UNIVERSITE DE LIEGE
Faculté des Sciences Appliquées

Symbolic Methods for Exploring Infinite State Spaces

Thèse présentée par

Bernard Boigelot

en vue de l'obtention du titre
de Docteur en Sciences Appliquées

Année Académique 1997–1998

Abstract

In this thesis, we introduce a general method for computing the set of reachable states of an infinite-state system. The basic idea, inspired by well-known state-space exploration methods for finite-state systems, is to propagate reachability from the initial state of the system in order to determine exactly which are the reachable states. Of course, the problem being in general undecidable, our goal is not to obtain an algorithm which is guaranteed to produce results, but one that often produces results on practically relevant cases.

Our approach is based on the concept of *meta-transition*, which is a mathematical object that can be associated to the model, and whose purpose is to make it possible to compute in a finite amount of time an infinite set of reachable states. Different methods for creating meta-transitions are studied. We also study the properties that can be verified by state-space exploration, in particular linear-time temporal properties.

The state-space exploration technique that we introduce relies on a symbolic representation system for the sets of data values manipulated during exploration. This representation system has to satisfy a number of conditions. We give a generic way of obtaining a suitable representation system, which consists of encoding each data value as a string of symbols over some finite alphabet, and to represent a set of values by a finite-state automaton accepting the language of the encodings of the values in the set. Finally, we particularize the general representation technique to two important domains: unbounded FIFO buffers, and unbounded integer variables. For each of those domains, we give detailed algorithms for performing the required operations on represented sets of values.

Acknowledgments

I would first like to thank my thesis advisor, Pierre Wolper, for getting me started in the field of verification, and sharing with me his contagious enthusiasm about finite automata. His insightful comments contributed significantly to the content and the presentation of this thesis. Thanks also to the other members of my jury, Daniel Ribbens, Pascal Gribomont, Véronique Bruyère, Pierre-Yves Schobbens, Alain Finkel and Bengt Jonsson, who have accepted to read and evaluate this thesis.

It has been a great pleasure for me to work in collaboration with several other people during these last four years. I wish to thank Patrice Godefroid, who significantly influenced much of the work contained in this thesis. Patrice also contributed to some of the results presented in Chapters 3 to 7, and made possible two exciting stays at Bell Laboratories in 1995 and 1996. Thanks to Bernard Willems, who introduced me to some areas of mathematics and most willingly helped me to tackle various problems. The results exposed in Chapter 8 would not be in the present form without the help of Bernard. Thanks also to Louis Bronne and to Stéphane Rassart, who contributed substantially to some of the results presented in Chapter 8 and provided me with valuable comments and suggestions. I am also grateful to Gérard Cécé and Philippe Louis for their careful reading of Chapter 7.

Thanks to all the people that shared with me their insightful ideas about symbolic verification at various conferences and meetings. In particular, I thank Pascal Gribomont, Ahmed Bouajjani, Alain Finkel, Yassine Lakhnech, Hubert Comon, and Véronique Bruyère.

I am grateful to the National Fund for Scientific Research of Belgium (FNRS), which supported me as research assistant ("aspirant") and research associate ("chargé de recherches").

Finally, thanks to my colleagues Philippe Lejoly, Didier Rossetto, Ulrich Nitsche, Didier Pirottin and Diana Tourko which provided me with an enjoyable work environment. Thanks also to Edie and Gary, to all my good friends for their support, and — last but not least — to Murielle for her ... infinite love.

Contents

List of Figures

Chapter 1

Introduction

Because of the rapid progress of computer technology over the last decades, computers are now present in a large variety of devices, ranging from home appliances driven by simple microcontrollers to phone switches controlled by massively parallel units. Even an increasing number of life-critical systems rely on computers: in modern *fly-by-wire* aircrafts, control surfaces are actuated by flight computers rather than being mechanically linked to the pilot controls. The consequences of computer system failures have thus become more and more severe. Over the last years, there have been numerous cases of major disturbances and even fatalities caused by computer problems [Neu96]. As a chilling example, there have been more than ten fatal computer-related aircraft incidents over the last fifteen years [Neu97].

It is therefore crucial for developers of computer systems to have at their disposal analysis techniques for detecting potential failures before those systems are used. Even for systems which are not life-critical, it is always economically sound to detect design flaws as early as possible in the development process.

A long promoted way of designing correct computer systems is to develop with the system a formal proof of its correctness. This proof is traditionally based on *invariants*, which are logic formulas whose truth value provably never changes during the possible runs of the system. The correctness of the system is expressed as a logical consequence of an invariant that is initially true. Invariants are written in dedicated logics such as Hoare's logic [Hoa69] or Dijkstra's programming calculus [Dij76]. Even though this mathematically appealing approach has occasionally been applied [vLS79, CE81, CM89, Gri93], it inherently suffers from major drawbacks:

- *It is costly.* Writing a formal proof, even with the assistance of a computerized tool, is not straightforward and may require a significant amount of time, ingenuity, and experience.
- *It is not practical.* For instance, it is not possible to reuse already existing code if this code was developed without a proof. Even if this does not seem to

be a major restriction for some specific applications, there are many domains in which it is not economically feasible not to reuse parts of existing systems (for instance, banking or phone switching software).

- *It is rigid.* Even if a system is proved correct with respect to some properties, obtaining a correction proof for other properties may require a complete new study of the system.

An alternative approach is automated verification. Given a computer system, one uses an automatic technique for checking that each execution of the system satisfies some correctness criteria. In practice, this cannot be done while taking into account all the details of the system; indeed, an analysis carried out up to the greatest level of precision would have to deal with the electrical and even chemical phenomena occurring inside the components of the computer. This is far beyond our ambition. The solution is to define some level of abstraction, and write a *model*, i.e., a formal description of the system at that level of abstraction. In addition, one must also define *properties* expressing the correctness of the model at that level of abstraction. Properties are often written in dedicated logic formalisms such as temporal logics [Wol86, Wol83, Eme90, MP92]. The analysis simply consists of checking if every execution of the model satisfies all the correctness properties.

The result of the analysis is either the detection of an error, or a guarantee that the model is correct with respect to the properties. In practice, results of the former type are the most interesting ones, because it is often easy to check whether an error in the model corresponds or not to an error in the original system. On the contrary, a guarantee of correctness for the model does not translate into a certainty of correctness for the system, unless lots of hypotheses are assumed. (Nevertheless, such a proof may increase the level of confidence in the system.)

In this thesis, we consider systems modeled as *state machines*. Intuitively, this modeling scheme is based on the assumption that each run of the system can be described by a (possibly infinite) sequence of discrete *state changes*. The model then consists of a finite amount of information defining the initial state of the system, as well as all the possible state changes.

A simple way of checking the correctness of such a model is to explore its state space. Roughly speaking, the idea is to check systematically all the possible situations that can occur during the possible executions of the model. If an execution violating a property is found, then a scenario proving that the model is erroneous is produced. If no such execution is found after exploring all the possible situations, then one can deduce that the model is "correct" [Hol88, Hol90]. The main drawback of this approach is that a model can have a very large number of states (meaning that there are a very large number of situations to check). This phenomenon is known as the *combinational explosion* of the number of states with respect to the size of the model. Tools have been developed for performing exhaustive state-space

exploration [HK90, Hol91, DDHY92, FGM+92], and have been successfully used to detect unsuspected errors in industrial systems [BG96a]. However, their applicability is still limited to small systems. Simple optimizations of the iterative state-space exploration technique have been proposed in order to broaden the set of analyzable systems [Mor68, WL93, PY97]. Despite some practical advantages inherent to those optimizations, their use does not significantly increase the order of magnitude of the size of the systems that can be handled.

On the other hand, techniques were developed for attacking directly the sources of state-space explosion. A first example is *partial-order methods* [Val91, GW93, God96], which attempt to limit the explosion caused by the modeling of concurrency by interleaving [Win84]. The idea consists of exploring only a part of the state space, this part being sufficient for checking the validity of the properties of interest. Another category of techniques tackling state-space explosion are *symbolic methods* [BCM+92, McM93]. There, the basic idea is to represent and manipulate sets of states implicitly (with the help of specific data structures), rather than explicitly (as enumerations of their components). In this approach, the improvement does not concern the number of states to be explored, but instead the total cost of this exploration. Symbolic methods have been successfully applied to different domains such as hardware circuits [KL93], real-time systems [ACD90, AHH93, HNSY94], and hybrid systems [HH94, Hen96].

The most widely used representation system for symbolic exploration is the *Binary Decision Diagram (BDD)* [Bry92]. The idea consists of encoding the elements of a set as fixed-length words of bits. The set is then represented by a canonical decision diagram – isomorphic to a directed acyclic graph – that recognizes the encodings of all the elements of the set. This simple and elegant representation has efficient implementations, and can easily be applied to a large class of domains. It does however suffer from an important drawback: BDDs only allow the representation of finite sets. As a consequence, symbolic exploration with BDDs is limited to the analysis of models with a finite state space.

It is however crucial to be able to analyze models with an infinite state space. Indeed, even though all physically constructible systems are finite in some sense, their size is often way larger than what can be handled by finite-state methods. Modeling such systems as infinite-state systems is then more realistic than artificially bounding their size well below reality. (For instance, a buffer with ten megabytes of capacity is more accurately modeled by an unbounded buffer than by a two-byte buffer.) Another reason is that verification methods can also be used to check the correctness of abstract systems from which real systems can then be derived. It is often more comfortable to reason independently from any limit than to impose an arbitrary upper bound on the size of a system. Finally, it should be stressed that techniques developed for infinite-state systems may remain very powerful for analyzing systems for which the state space is finite but very large. For instance,

```
program COUNTER;
1:   var i : unbounded integer;
2:   begin
3:       i := 0;
4:       repeat
5:           i := i + 2
6:       until i = 0
7:   end.
```

Figure 1.1: Simple infinite-state system.

there are systems whose state space is limited by an upper bound (such as the capacity of a communication object), for which the cost of state-space exploration appears to be independent of that bound.

Infinite-state models also have some disadvantages. The main one is that most elementary properties are undecidable for sufficiently expressive classes of models [EN94, Fin94, HKPV95, CFI96, ACJT96, AJ96, Esp97]. This implies that, in general, it is not possible to analyze such systems rigorously, and hence that only partial results can be obtained. Note however that this situation is not very different in practice from what occurs for finite-state systems, for which the analysis is often impossible due to excessive resource (time or memory) requirements, in spite of a theoretical guarantee that an analysis can always be carried out. Our point of view is that it is more useful to provide a partial solution to an important general problem rather than isolate elegant but not very meaningful subclasses of systems for which a complete analysis is theoretically always possible.

Another drawback of infinite-state models is that the result of their reachability analysis cannot be expressed as the explicit enumeration of all their reachable states. One has thus to resort to symbolic methods for representing implicitly sets of states, as well as to specific techniques for computing infinitely many reachable states in a finite amount of time. This is not very different from what is usually done during program analyses carried out by hand, as illustrated with the Pascal-like program given in Figure 1.1. Even though this program has an infinite state space, it is easily inferred that:

- Each execution of the main loop at Lines 2–6 has the effect of adding 2 to the value of i;
- The values that i can take just before executing Line 6 are exactly all the strictly positive even numbers;

- The program does not terminate.

This approach can be generalized and automatized. In this thesis, we address the problem of exploring infinite-state spaces with the help of symbolic methods. The results presented here extend and unify those appearing in previous publications [BW94, WB95, BG96b, BGWW97].

1.1 Overview of the Thesis

This thesis is organized as follows. In Chapter 2, we describe the formalism that is used throughout this thesis for modeling systems. This formalism, a variant of state machines, is based on the distinction between control and data, and assumes that the control is finite. The data domain can be chosen freely, and is the source of the infinite nature of the state space. After the presentation of the syntax and semantics of the formalism, an example of its use is given.

In Chapter 3, a general technique for exploring the state space of an infinite-state system modeled according to the principles introduced in Chapter 2 is described. The basic idea, inspired by well-known state-space exploration methods for finite-state systems, is to propagate reachability from the initial state of the model in order to determine exactly which are the reachable states. For fundamental reasons, this problem can not be fully solved in general, hence we provide only a partial solution. This solution consists a semi-algorithm, i.e., an algorithm without guarantee of termination. Our approach is based on the concept of *meta-transition*, which is a mathematical object that can be associated to the model, and whose purpose is to make it possible to compute in a finite amount of time an infinite set of reachable states. Different methods for creating meta-transitions are studied. An example of reachability analysis concludes the chapter.

In Chapter 4, we study the properties that can be verified by state-space exploration. For instance, it is possible to use the method discussed in Chapter 3 to verify some properties of infinite execution sequences. In particular, we show how to check ω-regular properties, and therefore properties expressed as Linear-time Temporal Logic formulas. Once again, due to the undecidability of the underlying problem, only a partial solution can be obtained.

In Chapter 5, we study the termination of the semi-algorithms proposed in Chapters 3 and 4. After proving that it is impossible to define syntactically the exact class of systems for which the reachability problem can be solved, we propose a lower approximation of this class. In other words, we give a sufficient syntactic criterion (on models) that guarantees the termination of the reachability analysis. We also show that model checking ω-regular properties is decidable for the class of systems satisfying the criterion.

The symbolic state-space exploration technique introduced in Chapter 3 relies on a symbolic representation system for sets of data values manipulated during exploration. This representation system has to satisfy a number of conditions. In Chapter 6, we give a generic way of obtaining a suitable representation system. The main idea is to encode each data value as a string of symbols over some finite alphabet, and to represent a set of values by a finite-state automaton accepting the language of the encodings of the values in the set.

In Chapters 7 and 8, we particularize the notions introduced in Chapter 6 to two important domains: unbounded FIFO buffers, and unbounded integer variables. For each of those domains, we give detailed algorithms for performing the required operations on represented sets of values. In particular, we introduce original decision procedures for determining whether the closure of some sequences of data operations preserves the representability of sets of data values.

In Chapter 9, we conclude this thesis by a comparison with related work, as well as some ideas for future work.

Chapter 2

Structured-Memory Automata

This chapter presents the formalism that will be used for modeling programs. After introducing its syntax and semantics, it discusses the motivations of the choice that has been made.

2.1 Modeling Programs

We consider programs composed of a *control part*, which controls the order according to which instructions are performed, and a *data part*, which defines the operations carried out by instructions. The control part is modeled by a *control graph*, whose edges are labeled by instructions. Each path in the control graph corresponds to a sequence of instructions that can possibly be performed. The data part is modeled by *variables* whose values can influence, and be influenced by, the execution of instructions. In this thesis, we require that programs have a finite control graph; however, we do not impose any restriction on the domains of variables.

Formally, a program is modeled by a *Structured-Memory Automaton* (SMA), defined as follows.

Definition 2.1 *An SMA is a tuple* (C, c_0, M, m_0, Op, T)*, where*

- C *is a finite set of* control locations*;*
- c_0 *is an* initial control location*;*
- $M = D_1 \times D_2 \times \cdots \times D_n$ *(*$n \geq 0$*) is a* memory domain*, structured as the Cartesian product of* variable domains $D_1, D_2, \ldots, D_n$ *(which may be infinite). The dimension* n *of* M *defines the number of variables of the SMA; those variables are denoted* $x_1, x_2, \ldots, x_n$*. Each element* $m = (v_1, v_2, \ldots, v_n) \in M$ *is a* memory content*. For every* i *such that* $1 \leq i \leq n$*, the component* v_i *of* m *corresponds to the value of* x_i*;*
- $m_0 = (v_{1,0}, v_{2,0}, \ldots, v_{n,0}) \in M$ *is an* initial memory content*;*

- *Op is a (possibly infinite) set of* memory operations. *Each operation $\theta \in Op$ is a function $M \to M$. This function may be partial, i.e., it does not need to be defined for every memory content in M (the fact that θ is undefined for the memory content $m \in M$ is denoted $\theta(m) = \perp$);*

- *$T \subseteq C \times Op \times C$ is a finite set of* transitions. *Each transition is a triple (c, θ, c'), where c is the* origin, *c' the* end, *and θ the* label *of the transition.*

2.2 Semantics

The semantics of an SMA is defined in terms of a state-transition system. The execution of an SMA consists of a possibly infinite sequence of discrete state changes, starting from a distinguished initial state. At each step, the possible state changes are determined by the outgoing transitions from the current state. SMAs can be non-deterministic, i.e., there may be several possible state changes from any given state.

Formally, the semantics of an SMA $\mathcal{A} = (C, c_0, M, m_0, Op, T)$ is the state-transition system (Q, q_0, R), where:

- $Q = C \times M$ is the set of potential *states*. Each state $q = (c, m) \in Q$ is thus composed of a control location $c \in C$ and a memory content $m \in M$. Since M may be infinite, Q may be infinite as well;

- $q_0 = (c_0, m_0)$ is the *initial state*;

- $R \subseteq Q \times Q$ is the *one-step reachability relation*. A pair $((c, m), (c', m'))$ belongs to R, which is denoted $(c, m) \to_R (c', m')$, if there exists a transition $(c, \theta, c') \in T$ such that $m' = \theta(m)$. The state (c', m') is then said to be *reachable in one step* from the state (c, m).

Let $\mathbf{N}_0$ denote the set of strictly positive integers. A state $q' \in Q$ is *reachable from* a state $q \in Q$ if there exist $k \in \mathbf{N}_0$ and $q_1, q_2, \ldots q_k \in Q$ such that $q = q_1$, $q_k = q'$, and $q_i \to_R q_{i+1}$ for all $0 < i < k$. This is equivalent to stating that the pair (q, q') belongs to the transitive closure R^* of R, which is the *reachability relation* of $\mathcal{A}$. The fact that (q, q') belongs to R^* is denoted $q \to_R^* q'$. The fact that there exists $q'' \in Q$ such that $q \to_R q''$ and $q'' \to_R^* q'$ is denoted $q \to_R^+ q'$. As a particular case of the definition of reachability, every state in Q is reachable from itself. A state $q \in Q$ is *reachable* if it is reachable from the initial state q_0. The set of all the reachable states is denoted Q_R. The *state space* (Q_R, R_R) of $\mathcal{A}$ is the (possibly infinite) graph whose nodes are the reachable states of $\mathcal{A}$, and whose edges correspond to the one-step reachability relation $R_R = R \cap (Q_R \times Q_R)$ between those states. A *computation* of $\mathcal{A}$ is a finite or infinite maximal sequence of states $q_1, q_2, \ldots \in Q$ such that $q_1 = q_0$, and $q_i \to_R q_{i+1}$ for all $i > 0$.

2.3 Example

An example of an SMA $\mathcal{A} = (C, c_0, M, m_0, Op, T)$ is given in Figure 2.1. It has the following components:

- $C = \{c_1, c_2\}$ (there are two control locations c_1 and c_2, corresponding to the nodes of the control graph of Figure 2.1);
- $c_0 = c_1$ (c_1 is the initial control location);
- $M = \mathbf{Z}^2$ (there are two integer variables x_1 and x_2. A memory content is thus a pair of integers);
- $m_0 = (0, 0)$ (the initial value of both variables is 0);
- $Op = \{x_{1++}, x_{2++}, \text{even}(x_1)\}$, where
  - $x_{1++} : \mathbf{Z}^2 \to \mathbf{Z}^2 : (v_1, v_2) \mapsto (v_1 + 1, v_2)$ (this operation increments the value of the variable x_1);
  - $x_{2++} : \mathbf{Z}^2 \to \mathbf{Z}^2 : (v_1, v_2) \mapsto (v_1, v_2 + 1)$ (this operation increments the value of the variable x_2);
  - $\text{even}(x_1) : \mathbf{Z}^2 \to \mathbf{Z}^2 : (v_1, v_2) \mapsto \begin{cases} (v_1, v_2) & \text{if } v_1 \text{ is even} \\ \bot & \text{if } v_1 \text{ is odd} \end{cases}$ (this operation tests whether the value of the variable x_1 is even);
- $T = \{(c_1, x_{1++}, c_1), (c_1, \text{even}(x_1), c_2), (c_2, x_{2++}, c_1)\}$ (there are three transitions, each of them corresponding to an edge of the control graph of Figure 2.1).

The SMA $\mathcal{A}$ is non-deterministic. Indeed, from a state such as $q_0 = (c_1, (0, 0))$ (the initial state), one can follow either transition (c_1, x_{1++}, c_1) or transition $(c_1, \text{even}(x_1), c_2)$. Since each of them leads to a different state, there are two different states that are reachable in one step from q_0.

The example also shows how memory contents can influence, and be influenced by, the execution of instructions. It is always possible to follow the transition (c_1, x_{1++}, c_1) from the control location c_1, and doing so has the effect of adding 1 to the value of x_1. On the other hand, it is only possible to follow $(c_1, \text{even}(x_1), c_2)$ from c_1 if the value of x_1 is even, and doing so has the effect of turning the control location from c_1 into c_2 without modifying the value of x_1 and x_2.

A part of the (infinite) state space of $\mathcal{A}$ is depicted in Figure 2.2. Each state of the form $(c_1, (v_1, v_2))$ with $v_1, v_2 \in \mathbf{N}$ (the control location is c_1 and the value of each variable is an arbitrary positive integer) is reachable, for instance by following from the initial state v_2 times the transitions $(c_1, \text{even}(x_1), c_2)$ and (c_2, x_{2++}, c_1), and then v_1 times the transition (c_1, x_{1++}, c_1). Each state of the form $(c_2, (v_1, v_2))$ with $v_1 \in 2\mathbf{N}$ (the set of all the positive even numbers) and $v_2 \in \mathbf{N}$ is reachable as well,

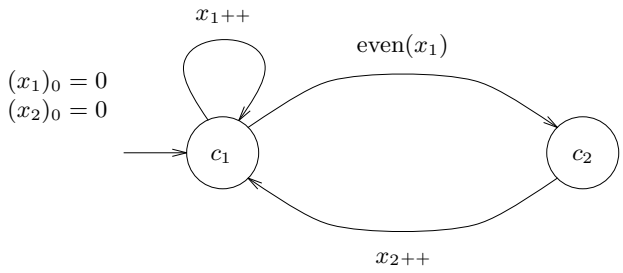

Figure 2.1: Example of SMA.

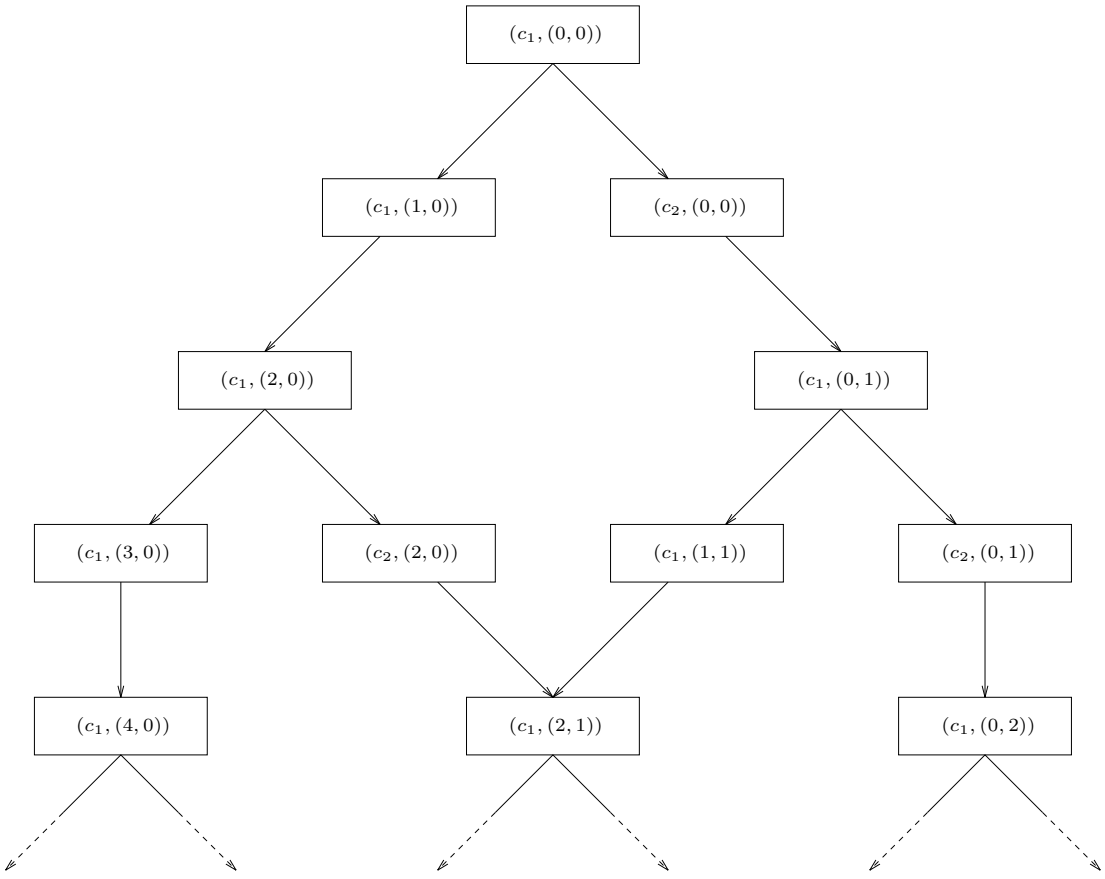

Figure 2.2: State space of the example.

by following the transition $(c_1, \text{even}(x_1), c_2)$ from the state $(c_1, (v_1, v_2))$. There is no other reachable state.

2.4 Discussion

SMAs are a simple yet powerful way of modeling programs. Expressing the control part as a finite graph makes it possible to model non-determinism as well as arbitrarily intricate control structures (such as for instance nested loops with multiple entry and exit points). There is no restriction on the data part of the program, since the memory domain and operations may be freely chosen. That makes it possible to model as SMAs systems such as Petri Nets[1] [Pet62, Pet81, Rei85] or FIFO nets [FR87], as well as programs expressed in imperative sequential programming languages such as C [KR78] or Pascal [Wir71]. In the next chapters (3–6), we present some results which hold independently from the memory domain and set of memory operations (provided that these satisfy some conditions which will be detailed). Next, in Chapters 7 and 8, the results are particularized to two important classes of SMAs, namely those using integer variables and linear operations, and those using FIFO channels and send/receive operations.

Other classes of systems can be indirectly modeled as SMAs. This is the case for concurrent systems based on the interleaving model of concurrency [Win84], provided that their control part is finite. There are simple algorithms for computing the *product* of all the components of the system, which is a sequential program equivalent to the whole system. Informally, this is done by grouping together the states of all the components into *global states*, and by associating to the product every transition corresponding to some transition of one of the components. Once computed, the product can be converted into an SMA. Although the product operation is usually costly, it can be efficiently implemented by performing the operation on-the-fly rather than globally. This consists of generating the set of outgoing transitions of a global state on demand rather than systematically, in order to avoid computing and storing useless information. Another approach is to compute only partly the product, the result being sufficient for verifying the property of interest. For instance, *partial-order* methods [God96] attempt to reduce the size of the product by not generating transitions for which it is known that they do not influence the result of the analysis. Two different interleavings of the same computations of the components will then correspond to one single computation of the product,

[1] A simple way of converting a Petri net into an SMA consists of building an SMA with only one control location, and with one natural variable for each place of the Petri net (this variable modeling the number of tokens at that place). Each transition of the Petri net is then converted into a transition of the SMA. The initial memory content of the SMA corresponds to the initial marking of the Petri net.

provided that they are shown to be equivalent with respect to the properties being checked. Those issues are not addressed in this thesis.

Modeling an actual system as an SMA is not always straightforward, for the distinction between control part and data part is somehow arbitrary. For instance, the SMA of Figure 2.1 could easily be turned into one with a single control location and an additional variable x_3 of domain $\{c_1, c_2\}$. Though a distinction between control part and data part sometimes appears naturally, there are rules that must be observed:

- *The infinite character of the state space must be entirely contained in the data part.* In other words, the control graph must be finite.

- *The set of memory operations should have a simple structure and have simple algebraic properties.* The purpose of this (informal) rule is to make easy the computation of the necessary operations on memory values. The concept is illustrated in the context of two particular memory domains with different properties in Chapters 7 and 8.

Chapter 3

Reachability Analysis

This chapter addresses the problem of computing the set of reachable states of an SMA. As it will be shown in Chapter 4, solving this problem makes it possible to decide various properties of programs modeled as SMAs. We propose a solution inspired by the algorithms developed for systems with a finite state space.

3.1 Finite-State Systems

Computing the set of reachable states of an SMA $\mathcal{A}$ is easy when this set is finite. Indeed, a simple solution consists of starting with a set containing only the initial state. Then, by following the one-step reachability relation R of $\mathcal{A}$, one obtains new reachable states which are added to the set. Since there are only a finite number of reachable states, repeating this operation iteratively will eventually produce a stable set, i.e., a set that can not be enlarged anymore by following R. At this point, the set contains exactly all the reachable states of the SMA.

Recall that the state space of an SMA is a graph whose nodes correspond to reachable states, and whose edges correspond to the one-step reachability relation between those states. It follows that the method outlined above can be seen as an *exploration* of the state-space graph, that is, a search visiting each node. There are various strategies that can be adopted for the search, differing from each other by the order according to which the nodes are visited. Figure 3.1 gives an algorithm based on a *breadth-first* search, whose strategy is to visit nodes in increasing order of depth (the depth of a node is the length of the shortest path from the initial state to that node). Figure 3.2 gives an algorithm using a different search order, the *depth-first* search. There, the strategy is to always follow an edge whose origin is the most recently visited node that has an unvisited successor. The two algorithms are equivalent, in the sense that they always yield the same result.

All the sets manipulated by the algorithms in Figures 3.1 and 3.2 are finite. This means that the algorithms can actually be implemented by representing sets

```
function REACHABLE-FINITE-B(SMA (C, c_0, M, m_0, Op, T)) : set of states
 1:  var visited-states, recent-states, new-states : sets of states;
 2:  begin
 3:      visited-states := ∅;
 4:      recent-states := {(c_0, m_0)};
 5:      repeat
 6:          visited-states := visited-states ∪ recent-states;
 7:          new-states := ∅;
 8:          for each (c, m) ∈ recent-states do
 9:              for each (c', θ, c'') ∈ T such that c' = c do
10:                  if θ(m) ≠ ⊥ and (c'', θ(m)) ∉ visited-states then
11:                      new-states := new-states ∪ {(c'', θ(m))};
12:          recent-states := new-states
13:      until recent-states = ∅;
14:      return visited-states
15:  end.
```

Figure 3.1: Breadth-first exploration of a finite state space.

```
function REACHABLE-FINITE-D(SMA (C, c_0, M, m_0, Op, T)) : set of states
 1:  var visited-states : set of states;
 2:  procedure explore(state (c, m))
 3:      begin
 4:          visited-states := visited-states ∪ {(c, m)};
 5:          for each (c', θ, c'') ∈ T such that c' = c do
 6:              if θ(m) ≠ ⊥ and (c'', θ(m)) ∉ visited-states then
 7:                  explore((c'', θ(m)))
 8:      end;
 9:  begin  (* REACHABLE-FINITE-D *)
10:      visited-states := ∅;
11:      explore((c_0, m_0));
12:      return visited-states
13:  end.
```

Figure 3.2: Depth-first exploration of a finite state space.

as finite lists of their elements. In practical applications, specific data structures such as hash tables can be used in order to speed up set operations. If implemented properly, both algorithms take $O(N_e)$ space and time, where N_e is the number of edges in the state space.

3.2 Infinite-State Systems

The main limit of the method presented in Section 3.1 is that it can only be applied to systems with a finite state space. Indeed, since each reachable state is visited individually, the exploration of an infinite state space by one of the algorithms of Figures 3.1 and 3.2 would never terminate.

It is possible though to follow the same approach, which consists of spreading the reachability information along the edges of the state-space graph, in order to explore infinite state spaces. In order to be able to do so, two items are needed:

- A technique for going through an infinite number of transitions in a finite amount of time;
- An algorithmically easy to handle finite representation of infinite sets of states.

3.2.1 Exploring Infinite Sets of Reachable States

In order to be able to explore infinite state spaces, one must be able to compute a possibly infinite set of reachable states in a finite number of steps. An idea is to generalize the basic operation for propagating reachability, so as to allow to deduce the reachability of an infinite set from the reachability of a finite set. This is done by introducing the concept of *meta-transition*.

Definition 3.1 *Let $\mathcal{A} = (C, c_0, M, m_0, Op, T)$ be an SMA. A* meta-transition $\bar{t}$ *for $\mathcal{A}$ is a triple (c, f, c'), where $c, c' \in C$ and $f : 2^M \to 2^M$, that satisfies the following property: for every set $U \subseteq M$ of memory contents, it is such that*

$$(\forall m' \in f(U))(\exists m \in U)((c, m) \to_R^* (c', m')),$$

where R is the one-step reachability relation of $\mathcal{A}$. The function f is called the memory function *of $\bar{t}$.*

Meta-transitions generalize the concept of transition. If $S \subseteq Q$ is a set of states, then *following* the meta-transition (c, f, c') from S leads to the set of states

$$S' = \mathit{states}(c', f(\mathit{values}(S, c))),$$

where $\mathit{values}(S, c)$ denotes the set $\{m' \in M \mid (c, m') \in S\}$ of all the memory contents associated to c in S, and for every $U \subseteq M$, $\mathit{states}(c, U)$ denotes the set $\{(c, m) \mid m \in U\}$ of all the states associating a memory content in U to c. As a consequence of Definition 3.1, the set S' contains only reachable states provided that S contains only reachable states. This means that meta-transitions propagate reachability information. However, unlike transitions, they are able to generate infinite sets of reachable states from finite sets of such states.

Exploring the state space of an SMA with the help of meta-transitions is done in the following way. The first step is to add meta-transitions to the SMA, which becomes an *Extended Structured-Memory Automaton (ESMA)*. The resulting ESMA has the same set of reachable states as the original SMA. The second step is to perform a state-space exploration of the ESMA, taking advantage of meta-transitions. The meta-transitions that are added to the SMA can be arbitrarily chosen as far as correctness is concerned. However, their choice clearly influences the termination of the state-space exploration.

Formally, an ESMA is defined as follows.

Definition 3.2 *An* Extended Structured-Memory Automaton *is a tuple $(C, c_0, M, m_0, Op, T, \bar{T})$, where*

- *(C, c_0, M, m_0, Op, T) is a Structured-Memory Automaton;*

- $\bar{T} \subseteq C \times F_M \times C$, *where* F_M *denotes the set of all the functions* $2^M \to 2^M$, *is a finite set of* meta-transitions. *Each element* $\bar{t} \in \bar{T}$ *is a meta-transition for the SMA* (C, c_0, M, m_0, Op, T).

The semantics of an ESMA $\mathcal{A}$ is derived from that of the underlying SMA. The set of potential states Q, the initial state q_0, the one-step reachability relation R and the state space (Q_R, R_R) of an ESMA $(C, c_0, M, m_0, Op, T, \bar{T})$ are identical to those of the underlying SMA (C, c_0, M, m_0, Op, T). If $q = (c, m)$, $q' = (c', m') \in Q$ are states and $t = (c_1, \theta, c_2) \in T$ is a transition such that q' is reachable from q by following t once, i.e., if $c = c_1 \wedge c' = c_2 \wedge m' = \theta(m)$, then we write $q \xrightarrow{t} q'$. Likewise, we write $q \stackrel{\bar{t}}{\Rightarrow} q'$ if q' is reachable from q by following once the meta-transition $\bar{t} = (c_1, f, c_2) \in \bar{T}$, i.e., if $c = c_1 \wedge c' = c_2 \wedge m' \in f(\{m\})$. Finally, we write $q \stackrel{\tilde{t}}{\rightharpoonup} q'$ if either $\tilde{t} \in T$ and $q \xrightarrow{\tilde{t}} q'$, or $\tilde{t} \in \bar{T}$ and $q \stackrel{\tilde{t}}{\Rightarrow} q'$.

For every reachable state $q \in Q_R$, there exist $k \in \mathbf{N}_0$, $q_1, q_2, \ldots, q_k \in Q_R$, and $\tilde{t}_1, \tilde{t}_2, \ldots, \tilde{t}_{k-1} \in T \cup \bar{T}$ such that $q_1 = q_0$, $q_k = q$, and $q_i \stackrel{\tilde{t}_i}{\rightharpoonup} q_{i+1}$ for every $i \in \{1, 2, \ldots, k-1\}$. The sequence $\pi = q_1, \tilde{t}_1, q_2, \tilde{t}_2, \ldots, \tilde{t}_{k-1}, q_k$ forms a *path* leading to q. This path is a *transition path* (resp. *meta-transition path*) if all the $\tilde{t}_i$ belong to T (resp. $\bar{T}$). Any subsequence $q_{i_1}, \tilde{t}_{i_1}, \ldots, \tilde{t}_{i_2-1}, q_{i_2}$ of π, with $1 \leq i_1 \leq i_2 \leq k$, is a *subpath*. The *length* of a path or subpath π is the number of transitions and meta-transitions appearing in π. Every reachable state has a *depth*, defined as the length of the shortest path leading to that state. Finally, two paths or subpaths $\pi = q_1, \ldots, q_{k_1}$ and $\pi = q'_1, \ldots, q'_{k_2}$ are said to be *equivalent* if $q_1 = q'_1$ and $q_{k_1} = q'_{k_2}$.

An algorithm for carrying out the state-space exploration of an ESMA by taking advantage of meta-transitions is presented in Section 3.3. Techniques for turning an SMA into an ESMA, i.e., for creating meta-transitions, are discussed in Section 3.4.

3.2.2 Representing Infinite Sets

An algorithm is only able to manipulate objects if their value can be encoded as a finite string of bits. It follows that the exploration of infinite state spaces requires a *representation system* for sets of states, that is, an encoding scheme transforming a set into a finite amount of information describing it unambiguously. All representation systems have a limited expressiveness, in the sense that they do not define an encoding for every possible infinite set. This is unavoidable, since there are uncountably many subsets of an infinite set of states, but only countably many finite strings of bits.

Since the infinite nature of the state space is a consequence of that of the data part of the program, it is natural to define representation systems for infinite sets of states in terms of representation systems for infinite sets of memory contents. Actually, since there are only a finite number of control locations, one can represent a (possibly infinite) set of states by associating to each control location the

representation of a set of memory contents.

From now on, we assume that sets of states are represented this way, and hence that a representation system for subsets of M is available. This system has to satisfy some conditions; in particular, one must be able to perform some elementary operations on represented sets of memory contents. The requirements are formalized in the following definition.

Definition 3.3 *Let $\mathcal{A} = (C, c_0, M, m_0, Op, T, \bar{T})$ be an ESMA. A representation system for subsets of M is* well suited *for $\mathcal{A}$ if:*

- *The following sets of memory contents are representable:*
  - *The empty set $\emptyset$;*
  - *The universal set M;*
  - *Every set $\{m\}$, where $m \in M$, and*
- *All the following operations can be performed algorithmically on every representable sets $U_1, U_2 \subseteq M$:*
  - *Computing the union $U_1 \cup U_2$, intersection $U_1 \cap U_2$, and difference $U_1 \setminus U_2$;*
  - *Testing the inclusion $U_1 \subseteq U_2$;*
  - *Testing the emptiness of U_1;*
  - *Computing the image $\theta(U_1) = \{\theta(m) \mid m \in U_1\}$ of U_1 by any operation $\theta \in Op$;*
  - *Computing the image $f(U_1)$ of U_1 by any function $f : 2^M \to 2^M$ labeling a meta-transition $(c, f, c') \in \bar{T}$.*

By extension, a representation system for sets of states is said to be *well suited* for an ESMA $\mathcal{A}$ if it represents sets of states as lists of pairs (control location, set of memory contents), where the sets of memory contents are represented in a representation system that is well suited for $\mathcal{A}$. If $S \subseteq Q$ is a set of states and $c \in C$ is a control location, then a representation of the set $values(S, c)$ is trivially computed from a representation of S by simply locating the pair (control location, representation of set of contents) corresponding to c. If $c \in C$ is a control location and $U \subseteq M$ is a set of memory contents, then a representation of the set $states(c, U)$ simply consists of the pair (c, representation of U). Implementations of elementary set-theory operations such as intersection, union, difference, test of inclusion and test of emptiness on representable sets of states are easily deduced from the corresponding operations on representable sets of memory contents.

A general method for obtaining representation systems well suited for some types of ESMAs is described in Chapter 6. The method is particularized to two important classes of ESMAs in Chapters 7 and 8.

3.3 Symbolic State-Space Exploration

The set of reachable states of an ESMA (or, more precisely, a finite and exact representation of this set) can be computed by the same approach as for finite-state SMAs. The idea is to propagate reachability information by following transitions, but also meta-transitions.

An algorithm formalizing this idea is given in Figure 3.3. It can be seen as a generalization of the breadth-first search of Figure 3.1. The main difference is that several states, as opposed to a single state, are now visited at each step. We assume that the sets of states manipulated by the algorithm are represented with the help of a well suited representation system, hence the name "*Symbolic State-Space Exploration*" of this technique, to highlight the fact that sets of states are not simply manipulated as enumerations of their elements.

Despite the fact that following meta-transitions makes it possible to compute an infinite number of reachable states in a finite amount of time, state-space exploration algorithms are not guaranteed to terminate when the state space is infinite. Indeed, there are classes of systems such as FIFO nets [FR87] that can be modeled as ESMAs, but for which it is known that their set of reachable states can generally not be computed. It follows that the algorithm of Figure 3.3 is actually a *semi-algorithm*, i.e., a procedure that does not necessarily terminate. This semi-algorithm is correct thanks to the following result.

Theorem 3.4 *Let $\mathcal{A}$ be an ESMA such that the computation of REACHABLE($\mathcal{A}$) terminates. The result of this computation contains exactly all the reachable states of $\mathcal{A}$.*

Proof

- *The result contains only reachable states.* This is a direct consequence of the fact that, at any time during the computation, the sets of states *visited-states*, *recent-states* and *new-states* contain only reachable states. Indeed, executing Lines 10–11 (resp. 12–13) adds to *new-states* states that are reachable by following a transition (resp. a meta-transition) from states in *recent-states*.

- *The result contains all the reachable states.* At any time during the computation, let N denote the number of times Line 15 has been executed. Prior to each execution of Line 15, the set *recent-states* contains exactly all the states whose depth is N (this is easily shown be induction on N). If the computation terminates, then the test *recent-states* $= \emptyset$ at Line 16 succeeds for some value of N. This means that all the reachable states of $\mathcal{A}$ have a depth less than N. Therefore, all of them belong to the set *visited-states* returned at the end of the computation.

```
function REACHABLE(ESMA (C, c_0, M, m_0, Op, T, T̄)) : set of states
 1:  var visited-states, recent-states, new-states : sets of states;
 2:  begin
 3:      visited-states := ∅;
 4:      recent-states := {(c_0, m_0)};
 5:      repeat
 6:          visited-states := visited-states ∪ recent-states;
 7:          new-states := ∅;
 8:          for each c ∈ C such that values(recent-states, c) ≠ ∅ do
 9:              begin
10:                  for each (c', θ, c'') ∈ T such that c' = c do
11:                      new-states := new-states ∪
                             states(c'', θ(values(recent-states, c))) \ visited-states;
12:                  for each (c', f, c'') ∈ T̄ such that c' = c do
13:                      new-states := new-states ∪
                             states(c'', f(values(recent-states, c))) \ visited-states
14:              end;
15:          recent-states := new-states
16:      until recent-states = ∅;
17:      return visited-states
18:  end.
```

Figure 3.3: Breadth-first exploration of an infinite state space.

□

The arguments developed in the second part of the proof have an important corollary.

Theorem 3.5 *Let $\mathcal{A}$ be an ESMA. The computation of REACHABLE($\mathcal{A}$) terminates if and only if there exists an upper bound on the depth of all the reachable states of $\mathcal{A}$.*

Proof If the computation terminates, then the number N of times Line 15 has been executed is a suitable upper bound. Reciprocally, if there exists an upper bound $N_{up} \in \mathbf{N}$ on the depth of all the reachable states of $\mathcal{A}$, then *visited-states* will eventually contain all the reachable states for some value of N less or equal to N_{up}. At the next execution of Line 16, the condition *recent-states* $= \emptyset$ is satisfied and the computation terminates. □

There exist other semi-algorithms than the one given in Figure 3.3 for computing the set of reachable states of an ESMA by following repeatedly transitions and meta-transitions. Like for finite-state systems, they differ from each other by the order according to which the states are visited. As an example, Figure 3.4 gives a semi-algorithm analogous to the depth-first search of Figure 3.2.

Unlike for finite-state systems, the different search strategies for exploring infinite state spaces are not equivalent. Although semi-algorithms based on different search strategies always give out the same result when they terminate, the class of ESMAs for which they terminate is generally different. In that context, Theorem 3.5 has an interesting corollary.

Corollary 3.6 *The breadth-first strategy used by the semi-algorithm of Figure 3.3 always terminates whenever there is some other search strategy that terminates.*

Proof If there exists a search strategy that terminates after a finite number of steps for the ESMA $\mathcal{A}$, then all the reachable states of $\mathcal{A}$ are reached from the initial state after following a finite number of times individual transitions and meta-transitions. Let N be this number. Since N is an upper bound on the depth of all the reachable states of $\mathcal{A}$, it follows from Theorem 3.5 that the state-space exploration of $\mathcal{A}$ by the semi-algorithm of Figure 3.3 terminates. □

Even though Theorem 3.5 gives a necessary and sufficient condition of termination for the semi-algorithm of Figure 3.3, it does not provide an effective procedure for deciding whether the state-space exploration of a given ESMA terminates or not. In Chapter 5, which addresses termination issues, we show that there does not exist such an effective procedure for most classes of ESMAs. It is nevertheless possible to give sufficient static conditions on ESMAs for ensuring that the exploration of their state space terminates; an example of such a condition is also given in Chapter 5.

```
function REACHABLE-D(ESMA (C, c_0, M, m_0, Op, T, T̄)) : set of states
 1:  var visited-states : set of states;
 2:  procedure explore(set of states current-states)
 3:      begin
 4:          if current-states ⊆ visited-states then return;
 5:          visited-states := visited-states ∪ current-states;
 6:          for each c ∈ C such that values(current-states, c) ≠ ∅ do
 7:              begin
 8:                  for each (c', f, c'') ∈ T̄ such that c' = c do
 9:                      explore(states(c'', f(values(current-states, c))));
10:                  for each (c', θ, c'') ∈ T such that c' = c do
11:                      explore(states(c'', θ(values(current-states, c))))
12:              end
13:     end;
14:  begin  (* REACHABLE-D *)
15:      visited-states := ∅;
16:      explore({(c_0, m_0)});
17:      return visited-states
18:  end.
```

Figure 3.4: Depth-first exploration of an infinite state space.

3.4 Creating Meta-Transitions

Meta-transitions are created during the transformation of an SMA into an ESMA. Since the presence of meta-transitions has no influence over the set of reachable states of an ESMA, meta-transitions can be arbitrarily chosen as far as the partial correctness of the state-space exploration is concerned. However, termination of the state-space exploration is usually influenced by the choice of meta-transitions.

For every SMA $\mathcal{A} = (C, c_0, M, m_0, Op, T)$ with an infinite state space, there are infinitely many potential meta-transitions. Indeed, if $\mathcal{A}$ has an infinite set S of reachable states, then there exists at least one control location $c \in C$ such that $values(S, c)$ is infinite. For every subset U of $values(S, c)$, one can create a meta-transition (c_0, f_U, c), where f_U is the function $2^M \to 2^M$ such that for every $U' \subseteq M$, $f_U(U') = U$ if $m_0 \in U'$, and $f_U(U') = \emptyset$ if $m_0 \notin U'$.

Since an ESMA can only have a finite number of meta-transitions, a restriction has to be imposed over the set of potential meta-transitions. There are various methods for imposing such a restriction.

3.4.1 Cycle Meta-Transitions

When a meta-transition is followed, an infinite number of states may be reached from a finite number of states. A natural idea is thus to associate meta-transitions to elements of SMAs that are responsible for the infinite nature of their state space.

The only cause of state-space infinity for an SMA (C, c_0, M, m_0, Op, T) is the presence of cycles in its control graph. A *cycle* is a sequence $\mathcal{C} = (c_1, \theta_1, c'_1), \ldots, (c_k, \theta_k, c'_k)$ $(k \geq 1)$ of transitions in T such that $c'_k = c_1$ and for every $0 < i < k$, $c'_i = c_{i+1}$. The sequence $\sigma = \theta_1, \theta_2, \ldots, \theta_k$ of all the operations labeling the transitions is the *body* of the cycle and is said to *label* $\mathcal{C}$; this is denoted $\sigma = body(\mathcal{C})$. The control location c_1 first visited by $\mathcal{C}$ is denoted $first(\mathcal{C})$. The cycle $\mathcal{C}$ is *simple* if it does not contain a subcycle, i.e., if there do not exist $1 \leq i < j \leq k$ such that $(c_i, \theta_i, c'_i), (c_{i+1}, \theta_{i+1}, c'_{i+1}), \ldots, (c_j, \theta_j, c'_j)$ is a cycle, and either $i > 1$ or $j < k$. The cycle $\mathcal{C}$ has k *rotations* denoted $rot(\mathcal{C}, 0), rot(\mathcal{C}, 1), \ldots, rot(\mathcal{C}, k-1)$, such that $rot(\mathcal{C}, 0) = \mathcal{C}$, and for every $i \in \{1, \ldots, k-1\}$,

$$rot(\mathcal{C}, i) = (c_{i+1}, \theta_{i+1}, c'_{i+1}), (c_{i+2}, \theta_{i+2}, c'_{i+2}), \ldots, (c_k, \theta_k, c'_k), (c_1, \theta_1, c'_1), \ldots, (c_i, \theta_i, c'_i).$$

Let $U \subseteq M$ be a set of memory contents. Following the cycle $\mathcal{C}$ from the set of states $states(c_1, U)$ amounts to following successively all the transitions composing $\mathcal{C}$, yielding the set of states $states(c_1, U')$, where $U' = \sigma(U) = \theta_k(\theta_{k-1}(\cdots\theta_1(U)\cdots))$ is the final set of memory contents and $\sigma = body(\mathcal{C})$. The set of contents obtained after following the cycle l times $(l \geq 0)$ from the set of contents U is denoted $\sigma^l(U)$. The set of all the contents that can be obtained by following the cycle any number

of times from the set of contents U is denoted $\sigma^*(U)$, it can be seen as the result of applying to U the function

$$\sigma^* : 2^M \to 2^M : U \mapsto \bigcup_{l \in \mathbf{N}} \sigma^l(U).$$

If $\mathcal{C}$ is a cycle, then the *cycle meta-transition* associated to $\mathcal{C}$ is a meta-transition whose effect is equivalent to following $\mathcal{C}$ any number of times. Formally, it is defined as follows.

Definition 3.7 *Let $\mathcal{A} = (C, c_0, M, m_0, Op, T)$ be an SMA, and $\mathcal{C} = (c_1, \theta_1, c_2), (c_2, \theta_2, c_3), \ldots, (c_k, \theta_k, c_1)$ $(k \geq 1)$ be a cycle in its control graph (C, T). The* cycle meta-transition *associated to $\mathcal{C}$ is the meta-transition (c_1, f, c_1), with $f : 2^M \to 2^M : U \mapsto body(\mathcal{C})^*(U)$.*

Cycle meta-transitions are valid meta-transitions, since their memory function satisfies the conditions of Definition 3.1. Indeed, let (c_1, f, c_1) be the cycle meta-transition associated to some cycle $\mathcal{C}$. If $U \subseteq M$ is a set of memory contents, and $U' = f(U)$, then for every $m' \in U'$, there exist $m \in U$ and $l \in \mathbf{N}$ such that (c_1, m') is reached from (c_1, m) by executing l times the body of $\mathcal{C}$. Since this implies $(c_1, m) \to_R^* (c_1, m')$, the conditions of Definition 3.1 are fulfilled.

Not all potential cycle meta-transitions are interesting to consider. First, cycles visiting control locations that are unreachable in the control graph of the SMA do not have to be considered.

Second, recall that the purpose of adding meta-transitions to an SMA is to allow the symbolic exploration of its state space, and that this exploration relies on a representation system for sets of memory contents. One should avoid to create meta-transitions such that the representation system will not be suited for the resulting ESMA (this only happens when the memory function f of the meta-transition cannot be computed on representable sets of memory contents). This rule is enforced as follows. Each representation system for sets of memory contents must define a predicate META? over the set of potential sequences of operations, whose purpose is to decide whether the corresponding meta-transition can be created or not. The predicate META? can be arbitrarily chosen, provided that it satisfies the following conditions:

- META? is computable over the sequences of operations in Op^*;
- There exists an algorithm for computing a representation of the set of memory contents $\sigma^*(U)$, given a sequence σ of operations such that META?(σ) is true and a represented set of memory contents U.

The restriction to cycles for which META? is true might not be strong enough to ensure that only a finite number of meta-transitions are created. There are different ways of imposing additional restrictions.

```
function META-SIMPLE(SMA 𝒜) : set of meta-transitions
1:  var meta-transitions : set of meta-transitions;
2:  begin
3:      meta-transitions := ∅;
4:      for each (c, σ) ∈ SIMPLE-CYCLES(𝒜) do
5:          if META?(σ) then meta-transitions := meta-transitions ∪ {(c, σ*, c)};
6:      return meta-transitions
7:  end.
```

Figure 3.5: Creation of simple-cycle meta-transitions.

Restriction to Simple Cycles

Since the control graph of an SMA is finite, it can only have a finite number of simple cycles. The idea is to create a meta-transition for every simple cycle in the control graph that is reachable and satisfies META? (a cycle is reachable in the control graph if it visits control locations for which there exists a sequence of transitions from the initial control location to these locations). The advantage of this approach is that there are classes of SMAs for which considering all the simple-cycle meta-transitions is sufficient for ensuring that symbolic state-space exploration terminates. The issue is discussed in detail in Chapter 5.

An algorithm for creating all the meta-transitions that can be derived from reachable simple cycles is given in Figure 3.5. It relies on a function SIMPLE-CYCLES that returns all the reachable simple cycles in the control graph of an SMA. An algorithm for computing this function is given in Figure 3.6[1]. This algorithm proceeds by performing a depth-first search in the control graph, without storing a table of the control locations already visited. This means that paths in the control graph are explored until they visit the same control location twice, rather than until they visit a control location already visited by a (possibly different) path. Whenever a control location occurs twice on the same path, the cycle corresponding to the subpath located between the two occurrences is added to the set computed so far, as well as are all the rotations of this cycle. The algorithm is correct thanks to the following result.

Theorem 3.8 *Let $\mathcal{A}$ be an SMA. SIMPLE-CYCLES($\mathcal{A}$) returns the set of all the pairs (c, σ) such that σ is the body of a simple cycle $\mathcal{C}$ that is reachable in the control graph of $\mathcal{A}$, and c is the first control location visited by $\mathcal{C}$.*

[1] In this algorithm, σ_1, σ_2 denotes the concatenation of the sequences of transitions σ_1 and σ_2.

```
function SIMPLE-CYCLES(SMA (C, c_0, M, m_0, Op, T)) : set of (control location,
                                                    sequence of memory operations)
 1:  var cycles : set of (control location, sequence of memory operations);
 2:      node : array[0, 1, ...] of control locations;
 3:      edge : array[0, 1, ...] of memory operations;
 4:  procedure generate(integer depth1, depth2)
 5:      var i, j : integers;
 6:          σ : sequence of memory operations;
 7:      begin
 8:          for i := depth1 to depth2 do
 9:              begin
10:                  σ := edge[i];
11:                  for j := i + 1 to depth2 do σ := σ, edge[j];
12:                  for j := depth1 to i − 1 do σ := σ, edge[j];
13:                  cycles := cycles ∪ {(node[i], σ)}
14:              end
15:      end;
16:  procedure explore(control location c, integer depth)
17:      begin
18:          node[depth] := c;
19:          for each (c', θ, c'') ∈ T such that c' = c do
20:              begin
21:                  edge[depth] := θ;
22:                  if (∃i, 0 ≤ i ≤ depth) such that node[i] = c'' then
23:                      generate(i, depth)
24:                  else explore(c'', depth + 1)
25:              end
26:      end;
27:  begin  (* SIMPLE-CYCLES *)
28:      cycles := ∅;
29:      explore(c_0, 0);
30:      return cycles
31:  end.
```

Figure 3.6: Computation of all the simple cycles in the control graph.

Proof The proof is in three parts. First we establish termination. Then, we show that the result of the computation contains all the simple cycles that are reachable in the control graph. Finally, we prove that this result contains only such simple cycles.

- *The computation of SIMPLE-CYCLES($\mathcal{A}$) terminates.* Since C is finite, any exploration path that does not visit the same control location twice has a length bounded by the number of control locations in C. It follows that the number of recursive calls to Procedure *explore* is bounded, hence that the computation terminates.

- *If $\mathcal{C} = (c_1, \theta_1, c_2), \ldots, (c_k, \theta_k, c_1)$ is a simple cycle that is reachable in the control graph of $\mathcal{A}$, then the result of the computation of SIMPLE-CYCLE($\mathcal{A}$) contains the pair $(c_1, \mathit{body}(\mathcal{C}))$.* Since $\mathcal{C}$ is reachable in the control graph of $\mathcal{A}$, there exists a finite path π of transitions from the initial control location c_0 of $\mathcal{A}$ to c_1. Since occurrences of cycles may be removed from π, we can assume that all the control locations visited by π are distinct. Let l $(1 \leq l \leq k)$ be such that c_l is the first control location of $\mathcal{C}$ visited by π, and π' be the prefix of π leading from c_0 to c_l. Appending to π' the part of $\mathcal{C}$ between c_l and c_1 followed by the part of $\mathcal{C}$ between c_1 and c_{l-1} (or c_k if $l = 1$) yields a path π'' from c_0 to c_{l-1} that visits only distinct control locations. By induction on the argument *depth* of Procedure *explore*, we have that at some time, this procedure is called with the value of *depth* equal to the length of π'', $\mathit{node}[0], \mathit{node}[1], \ldots \mathit{node}[\mathit{depth}-1], c$ are all the control locations visited by π'', and $\mathit{edge}[0], \mathit{edge}[1], \ldots \mathit{edge}[\mathit{depth}-1]$ are the first *depth* operations labeling the transitions of π'' (in the same order). During this call to *explore*, the transition leading from c_{l-1} (or c_k if $l = 1$) to c_l in $\mathcal{C}$ is explored at Line 19. Since, by construction of π'', we have $c_l \in \{\mathit{node}[0], \mathit{node}[1], \ldots \mathit{node}[\mathit{depth}-1], c\}$, the condition at Line 22 is satisfied and therefore *generate* is called. At this time, the arguments of *generate* are such that

$$\begin{aligned} (\mathit{node}[\mathit{depth1}], \ldots, \mathit{node}[\mathit{depth2}]) &= (c_l, c_{l+1}, \ldots, c_k, c_1, c_2, \ldots, c_{l-1}) \\ (\mathit{edge}[\mathit{depth1}], \ldots, \mathit{edge}[\mathit{depth2}]) &= (\theta_l, \theta_{l+1}, \ldots, \theta_k, \theta_1, \theta_2, \ldots, \theta_{l-1}), \end{aligned}$$

  where for every $1 \leq p \leq k$, θ_p denotes the operation labeling the transition outgoing from c_p in $\mathcal{C}$. After the loop at Line 8 reaches the value of i such that $\mathit{Node}[i] = c_1$, the value of σ at Line 13 becomes equal to $\mathit{body}(\mathcal{C})$. The pair (c_1, σ) is thus added to the set *cycles* returned at the end of the computation.

- *If (c', σ') belongs to the result of the computation of SIMPLE-CYCLES($\mathcal{A}$), then there exists a simple cycle $\mathcal{C}$ that is reachable in the control graph of $\mathcal{A}$ such that $c' = \mathit{first}(\mathcal{C})$ and $\sigma' = \mathit{body}(\mathcal{C})$.* If (c', σ') belongs to the result of

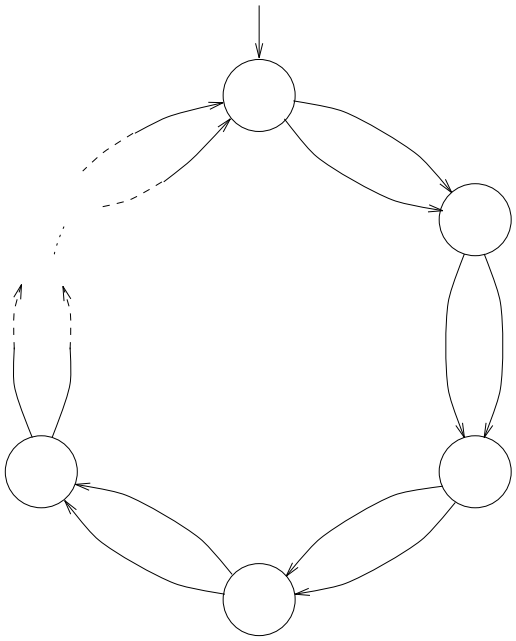

Figure 3.7: Control graph with $2N$ transitions and $N2^N$ simple cycles.

the computation of SIMPLE-CYCLES($\mathcal{A}$), then, by construction, there exists a control location $c \in C$ and a path π of transitions leading from the initial control location c_0 to c that is such that:

- c occurs exactly twice in π;
- σ' is the body of some rotation $\mathcal{C}$ of the simple cycle $\mathcal{C}'$ corresponding to the subpath of π located between the two occurrences of c;
- $c' = \mathit{first}(\mathcal{C})$.

It follows that $\mathcal{C}$ is a simple cycle that is reachable in the control graph of $\mathcal{A}$.

□

Computing all the simple cycles in the control graph can be a very inefficient strategy. This is illustrated in Figure 3.7 which shows how to build, for any $N \geq 1$, a control graph with $2N$ transitions and $N2^N$ distinct simple cycles. Executing the algorithm of Figure 3.6 on such a control graph would take $O(N^2 2^N)$ time, since each cycle is of length N and is produced transition by transition. It follows that this strategy can only be applied in practice if the control graph is small, or belongs to a specific class of graphs for which the cycle-search algorithm is more efficient than for arbitrary graphs.

Restriction to Syntactic Cycles

In some practical applications, SMAs are derived from specifications written in high-level modeling languages. Turning a high-level program into an SMA simply consists

```
function META-SYNTACTIC(SMA 𝒜) : set of meta-transitions
1:  var meta-transitions : set of meta-transitions;
2:  begin
3:      meta-transitions := ∅;
4:      for each (c, σ) ∈ SYNTACTIC-CYCLES(𝒜) do
5:          if META?(σ) then meta-transitions := meta-transitions ∪ {(c, σ*, c)};
6:      return meta-transitions
7:  end.
```

Figure 3.8: Creation of cycle meta-transitions from syntactic cycles.

of expressing its flow of control as a control graph, and converting its instructions into transitions. If the syntax of the high-level language contains specific constructs for defining loops, such as the "for", "while" and "repeat" statements of Pascal [Wir71], then cycles corresponding to loops defined that way can be identified with little additional cost during the translation of the program into an SMA.

The strategy consists of associating a meta-transition to each syntactically identified cycle (or, in short, syntactic cycle) satisfying the predicate META?. The procedure is formalized in Figure 3.8. In this program, SYNTACTIC-CYCLES($\mathcal{A}$) denotes the set of all the pairs (c, σ) such that σ is the sequence of operations labeling a syntactic cycle in the control graph of $\mathcal{A}$, and c is the first control location visited by that cycle. In actual implementations, SYNTACTIC-CYCLES is computed by the syntactic analyzer of the compiler used to translate programs into SMAs.

3.4.2 Multicycle Meta-Transitions

The concept of cycle meta-transitions can be generalized. If several cycles are starting from the same control location, then the *multicycle meta-transition* associated to those cycles is a meta-transition whose effect is equivalent to following any of them any number of times, in any order. Formally, it is defined as follows.

Definition 3.9 *Let $\mathcal{A} = (C, c_0, M, m_0, Op, T)$ be an SMA, and*

$$\begin{aligned}
\mathcal{C}_1 &= (c_{1,1}, \theta_{1,1}, c_{1,2}), \ldots, (c_{1,k_1}, \theta_{1,k_1}, c_{1,1}), \\
\mathcal{C}_2 &= (c_{2,1}, \theta_{2,1}, c_{2,2}), \ldots, (c_{2,k_2}, \theta_{2,k_2}, c_{2,1}), \\
&\vdots \\
\mathcal{C}_l &= (c_{l,1}, \theta_{l,1}, c_{l,2}), \ldots, (c_{l,k_l}, \theta_{l,k_l}, c_{l,1}) \subseteq T,
\end{aligned}$$

with $l \geq 1$, $k_1, k_2, \ldots k_l \geq 1$ *and* $c_{1,1} = c_{2,1} = \cdots = c_{l,1}$, *be cycles in its control graph. The* multicycle meta-transition *associated to the set* $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_l\}$ *is the meta-transition* $(c_{1,1}, f, c_{1,1})$, *with*

$$f : 2^M \to 2^M : U \mapsto \bigcup_{i \in \mathbf{N}} g^i(U),$$

and

$$g : 2^M \to 2^M : U \mapsto \bigcup_{1 \leq j \leq l} \mathit{body}(\mathcal{C}_j)(U).$$

Multicycle meta-transitions are valid meta-transitions, i.e., they satisfy the requirements of Definition 3.1. Indeed, let (c, f, c) be the multicycle meta-transition associated to the set of cycles $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_l\}$, with $l \geq 1$. If $U \subseteq M$ is a set of memory contents and $U' = f(U)$, then for every $m' \in U'$, there exist $m \in U$ and $p \in \mathbf{N}$ such that $\{m'\} = g^p(\{m\})$. From the definition of g, it follows that there exist $l_1, l_2, \ldots, l_p \in \{1, 2, \ldots, l\}$ such that $m' = \mathit{body}(\mathcal{C}_{l_p})(\mathit{body}(\mathcal{C}_{l_{p-1}})(\cdots \mathit{body}(\mathcal{C}_{l_1})(v)))$. Since each cycle is a sequence of transitions starting and ending at the control location c, we have $(c, m) \to_R^* (c, m')$. The conditions of Definition 3.1 are thus fulfilled.

Just as for cycle meta-transitions, not all possible multicycle meta-transitions can generally be considered, and thus a restriction needs to be imposed. One can use a similar strategy to the one proposed in Section 3.4.1 by considering only sets of cycles that are reachable in the control graph and generating only multicycle meta-transitions whose memory function is computable over representable sets of memory contents. In addition, one can also apply here the restrictions to simple or to syntactic cycles in order to obtain only a finite number of meta-transitions.

There is however a minor difference. Even if there are only a finite number of cycles to consider at a given control location, it is not convenient to test every subset of them in order to check if the corresponding multicycle meta-transition is computable (there may be exponentially many of them). In this case, the solution is not a predicate for testing whether a particular set of cycles leads to a computable meta-transition, but instead a function MULTI-META-SET that takes as arguments the sequences of operations labeling a set of cycles starting at the same control location, and returns a finite number of memory functions defining multicycle meta-transitions that can be associated to those cycles.

The function MULTI-META-SET may be arbitrarily chosen, provided that it satisfies the following conditions:

- MULTI-META-SET is computable over the finite sets of sequences of operations defined by the representation system;
- Let $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_l$ $(l \geq 1)$ be cycles starting at the same control location $c \in C$. MULTI-META-SET$(\{\mathit{body}(\mathcal{C}_1), \mathit{body}(\mathcal{C}_2), \ldots, \mathit{body}(\mathcal{C}_l)\})$ is finite;

- Every $f \in$ MULTI-META-SET($\{body(\mathcal{C}_1), body(\mathcal{C}_2), \ldots, body(\mathcal{C}_l)\}$) is the memory function of a multicycle meta-transition corresponding to the cycles $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_l$;

- There exists an algorithm for computing a representation of the set of memory contents $f(U)$, given a memory function f belonging to a set returned by MULTI-META-SET and a representation of a set of memory contents U;

- MULTI-META-SET is monotonous over the finite sets of sequences of operations labeling cycles that start at the same control location. This means that for every sets S_1, S_2, the inclusion $S_1 \subseteq S_2$ implies MULTI-META-SET(S_1) $\subseteq$ MULTI-META-SET(S_2). Intuitively this condition expresses the fact that if it is possible to obtain the set of meta-transitions $\bar{T}$ from the set of sequences S_1, then it should be possible to obtain at least all the elements of $\bar{T}$ from any superset of S_1.

An algorithm for computing multicycle meta-transitions that can be added to an SMA is given in Figure 3.9. This algorithm applies the simple-cycle restriction strategy, but other strategies can be implemented by replacing SIMPLE-CYCLES by the corresponding function at Line 5.

3.5 Dynamic Creation of Meta-Transitions

The state-space exploration technique presented in Section 3.3 proceeds by first adding meta-transitions to an SMA, and then exploring the state space of the resulting ESMA. Meta-transitions are created statically, i.e., without taking advantage of the reachability information obtained during the state-space exploration.

It is however possible to create cycle (or, more generally, multicycle) meta-transitions while the state-space exploration is being performed. This approach has an advantage: the selection performed among the potential meta-transitions guarantees that a finite set of multicycle meta-transitions for which the state-space exploration terminates will always be obtained if such a set exists. The main drawback is that the technique is computationally expensive.

An algorithm for exploring the state space of an SMA by creating dynamically multicycle meta-transitions is given in Figures 3.10 and 3.11[2]. This algorithm performs a breadth-first search in the state-space graph, while simultaneously carrying out cycle detection in order to create meta-transitions. The idea is to keep with each explored state a path leading to that state, provided that this path is only composed of transitions. Precisely, the algorithm maintains a list *explored-paths* of triples (σ, c, U) such that $states(c, U)$ is the set of states reached at the end of an

[2]In this algorithm, suf(σ) denotes the set of all the suffixes of the sequence of transitions σ.

```
function MULTI-META-SMA(SMA (C, c_0, M, m_0, Op, T)) : set of meta-transitions
 1:  var meta-transitions : set of meta-transitions;
 2:      cycles, current-cycles : sets of (control location,
                                          sequence of memory operations);
 3:  begin
 4:      meta-transitions := ∅;
 5:      cycles := SIMPLE-CYCLES((C, c_0, M, m_0, Op, T));
 6:      for each c ∈ C do
 7:          if (∃(c', σ) ∈ cycles) such that  c = c' then
 8:              begin
 9:                  current-cycles := {σ | (c, σ) ∈ cycles};
10:                  meta-transitions := meta-transitions ∪
                         {(c, f, c) | f ∈ MULTI-META-SET(current-cycles)}
11:              end;
12:      return meta-transitions
13:  end.
```

Figure 3.9: Creation of multicycle meta-transitions.

```
function REACHABLE-DYNAMIC(SMA (C, c_0, M, m_0, Op, T)) : set of states
 1:   var visited-states, recent-states : sets of states;
 2:       explored-paths, new-paths : sets of (sequence of transitions,
                                          control locations, set of memory contents);
 3:       cycles : array[C] of sequences of operations;
 4:       meta-transitions : set of meta-transitions;
 5:   procedure store-cycles(sequence of transitions σ)
 6:       begin
 7:           for each C = (c_1, θ_1, c'_1), ..., (c_l, θ_l, c'_l) ∈ suf(σ) such that c_1 = c'_l do
 8:               cycles[c_1] := cycles[c_1] ∪ {body(C)}
 9:       end;
10:   procedure explore-from(sequence of transitions σ, control location c,
                                                  set of memory contents U)
11:       begin
12:           for each (c', f, c'') ∈ meta-transitions such that c' = c do
13:               new-paths := new-paths ∪ {(⊥, c'', f(U))};
14:           for each (c', θ, c'') ∈ T such that c' = c do
15:               if σ = ⊥ then
16:                   new-paths := new-paths ∪ {(⊥, c'', θ(U))}
17:               else
18:                   begin
19:                       σ' := σ, (c', θ, c'');
20:                       new-paths := new-paths ∪ {(σ', c'', θ(U))};
21:                       store-cycles(σ')
22:                   end
23:       end;

    (...)
```

Figure 3.10: State-space exploration by dynamic creation of meta-transitions.

```
    (...)
24:  begin  (* REACHABLE-DYNAMIC *)
25:      visited-states := ∅;
26:      recent-states := {(c_0, m_0)};
27:      explored-paths := {(ε, c_0, {m_0})};
28:      for each c ∈ C do cycles[c] := ∅;
29:      meta-transitions := ∅;
30:      repeat
31:          visited-states := visited-states ∪ recent-states;
32:          new-paths := ∅;
33:          for each (σ, c, U) ∈ explored-paths do explore-from(σ, c, U);
34:          explored-paths := explored-paths ∪ new-paths;
35:          recent-states := ∅;
36:          for each (σ, c, U) ∈ explored-paths do
37:              recent-states := recent-states ∪ states(c, U);
38:          for each c ∈ C such that (∃(σ, c', U) ∈ explored-paths)(c = c') do
39:              meta-transitions := meta-transitions
                        ∪{(c, f, c) | f ∈ MULTI-META-SET(cycles[c])}
40:      until recent-states ⊆ visited-states;
41:      return visited-states
42:  end.
```

Figure 3.11: State-space exploration by dynamic creation of meta-transitions (continued).

exploration path π, and σ contains either the sequence of transitions corresponding to π, or the special value $\perp$ if π contains meta-transitions. Each time a control location c is reached, the current exploration path is checked for occurrences of cycles starting at c, and the possible meta-transitions corresponding to those cycles are added to the system[3]. The breadth-first strategy followed here is different in two points from the one implemented by the algorithm in Figure 3.3. First, reaching a state that has previously been visited does not prevent the path leading to that state from being further explored. Second, each iteration of the search attempts to append a transition or a meta-transition to every exploration path obtained so far, as opposed to only to the paths obtained as the result of the last iteration.

The motivation behind this modified strategy is twofold. First, since all the existing transition paths are searched for cycles in increasing order of length, every cycle whose detection is crucial for termination of state-space exploration will eventually be detected. Second, since no exploration path is ever discarded, the state-space exploration is guaranteed to terminate if there exists a finite set of multicycle meta-transition (not necessarily based on simple cycles) and a finite set of exploration paths containing those meta-transitions that lead to all the reachable states of the system (for those sufficient exploration paths will eventually be followed). These arguments will be used in the sequel of this section in order to show that dynamic state-space exploration is the most powerful strategy as far as termination is concerned. The algorithm of Figures 3.10 and 3.11, which is presented here as a proof that a theoretical "best" strategy exists from the point of view of termination, is however very inefficient and is thus not readily usable in practice. Indeed, there may be a large number of explored paths leading to the same reachable states. Some possible optimizations and approximations of this algorithm will be discussed in Section 3.7.

The algorithm is correct thanks to the following result.

Theorem 3.10 *Let $\mathcal{A} = (C, c_0, M, m_0, Op, T)$ be an SMA such that the computation of REACHABLE-DYNAMIC($\mathcal{A}$) terminates. The result of the computation contains exactly all the reachable states of $\mathcal{A}$.*

Proof

- *The result contains only reachable states.* It is sufficient to show that at any time during the computation, the variables *visited-states* and *recent-states* (if defined) only contain reachable states. This is a consequence of the following invariant:

[3]The meta-transitions are actually added at the end of each exploration step, in order to ensure that for every $k > 0$, only a finite number of paths of length k are explored.

  - The variables *new-paths* and *explored-paths* only contain triples (σ, c, U) such that $states(c, U)$ is a set of reachable states, and σ contains either a path of transitions leading to these states or the special value "$\perp$", and
  - For every $c \in C$, the variable $cycles[c]$ contains a set of sequence of operations labeling cycles of (C, T) starting at c, and
  - The variable *meta-transitions* only contains valid meta-transitions.

- *The result contains all the reachable states.* If the condition at Line 40 is satisfied, then all the states visited by the call to Procedure *explore-from* at Line 33 already belong to the set *visited-states* containing all the states visited during the previous iterations of the main loop. This means that, at Line 41, all the states that are reachable in one step from states in *visited-states* belong to *visited-states*. Since *visited-states* contains the initial state, it follows that this set contains all the reachable states of $\mathcal{A}$.

□

The main advantage of the dynamic approach, which is that the state-space search does always terminate if there exists a finite set of multicycle meta-transitions (not necessarily based on simple cycles) for which static state-space exploration terminates, is a consequence of the following theorem:

Theorem 3.11 *Let $\mathcal{A} = (C, c_0, M, m_0, Op, T)$ be an SMA. The symbolic state-space exploration performed by REACHABLE-DYNAMIC($\mathcal{A}$) terminates if and only if there exists a finite set $\bar{T}$ of multicycle meta-transitions such that computing REACHABLE($\mathcal{A}'$) by the algorithm of Figure 3.3, where $\mathcal{A}'$ is the ESMA $(C, c_0, M, m_0, Op, T, \bar{T})$, terminates.*

Proof

- *If the computation of REACHABLE-DYNAMIC($\mathcal{A}$) terminates, then there exists a terminating set of multicycle meta-transitions $\bar{T}$.* Every reachable state of $\mathcal{A}$ is reached during the computation of REACHABLE-DYNAMIC($\mathcal{A}$) by following a path of transitions and/or meta-transitions whose length is bounded. Let $\bar{T}$ be the (finite) set containing all the meta-transitions followed during the execution of REACHABLE-DYNAMIC. There exists an upper bound on the depth of all the reachable states of the ESMA $\mathcal{A}' = (C, c_0, M, m_0, Op, T, \bar{T})$. From Theorem 3.5, it follows that the computation of REACHABLE($\mathcal{A}'$) terminates.

- *If there exists a terminating set of multicycle meta-transitions $\bar{T}$, then the computation of REACHABLE-DYNAMIC($\mathcal{A}$) terminates.* Suppose that there exists such a set $\bar{T}$. Without loss of generality, we may assume that it contains

only multicycle meta-transitions derived from reachable cycles (a cycle is reachable if there exists a reachable state from which the entire sequence of transitions composing the cycle can be followed). Indeed, since unreachable cycles are never followed during state-space exploration, ignoring them while creating meta-transitions does not impact the computation of REACHABLE($\mathcal{A}'$).

Since $\bar{T}$ contains a finite number of multicycle meta-transitions, each derived from a finite number of reachable cycles, there exists an upper bound N on the length of the shortest paths of transitions in the state space of $\mathcal{A}$ in which the entire sequence of transitions composing the cycles is followed. If the computation of REACHABLE-DYNAMIC($\mathcal{A}$) does not terminate, then all such paths of transitions as well as all their prefixes can be found in the first component of the triples belonging to *explored-paths* after Line 34 has been executed N times. It follows that all the cycles that have been entirely followed by those paths have been detected by Procedure *store-cycles*, and that the sequences of operations labeling them belong to the variables *cycles* (which are indexed with respect to the control location at which cycles start). It follows that for each meta-transition $\bar{t} \in \bar{T}$, Line 39 will eventually be executed with *cycles*[c] containing at least all the cycles from which $\bar{t}$ can be derived. As a consequence, the variable *meta-transitions* will eventually contain all the meta-transitions in $\bar{T}$.

By Theorem 3.5, all the reachable states of $\mathcal{A}$ can be explored by following from the initial state a path of transitions and/or meta-transitions in $\bar{T}$ of bounded length (let N' be an upper bound on this length). After Line 34 has been executed N times, the variable *explored-path* still contains the triple $(\varepsilon, c_0, \{m_0\})$. It follows that after Line 34 has been executed $N + N'$ times, all the paths of transitions and/or meta-transitions in $\bar{T}$ of length less or equal to N' have been explored. As a consequence, all the reachable states have been explored, and the algorithm terminates at the next iteration of the main loop.

□

3.6 Example

The notions introduced in this chapter can be applied to the analysis of the example described in Section 2.3. There are three simple cycles in the control graph of the SMA depicted in Figure 2.1:

$$\begin{aligned}
\mathcal{C}_1 &= (c_1, x_{1{+}{+}}, c_1), \\
\mathcal{C}_2 &= (c_1, \text{even}(x_1), c_2), (c_2, x_{2{+}{+}}, c_1), \\
\mathcal{C}_3 &= (c_2, x_{2{+}{+}}, c_1), (c_1, \text{even}(x_1), c_2).
\end{aligned}$$

Suppose that we have at our disposal a representation system for subsets of $\mathbf{Z}^2$ for which META?($body(\mathcal{C}_1)$) = META?($body(\mathcal{C}_2)$) = **T** and META?($body(\mathcal{C}_3)$) = **F**. This means that it is possible to create two simple-cycle meta-transitions $\bar{t}_1 = (c_1, f_1, c_1)$ and $\bar{t}_2 = (c_1, f_2, c_1)$, with $f_1 = (x_{1++})^*$ and $f_2 = (\text{even}(x_1), x_{2++})^*$. Precisely, we have

$$\begin{array}{rcl} f_1 & : & 2^{\mathbf{Z}^2} \to 2^{\mathbf{Z}^2} \quad : \quad U \mapsto \{(v_1 + k, v_2) \mid (v_1, v_2) \in U \ \wedge \ k \in \mathbf{N}\}, \\ f_2 & : & 2^{\mathbf{Z}^2} \to 2^{\mathbf{Z}^2} \quad : \quad U \mapsto \{(v_1, v_2 + k) \mid (v_1, v_2) \in U \ \wedge \ k \in \mathbf{N} \\ & & \qquad\qquad\qquad\qquad \wedge \ (\text{even}(v_1) \ \vee \ k = 0)\}. \end{array}$$

Let $\mathcal{A}$ be the ESMA obtained by adding to the system the set of meta-transitions $\{\bar{t}_1, \bar{t}_2\}$. The details of the computation of REACHABLE($\mathcal{A}$) by the algorithm of Figure 3.3 are given in Figure 3.12. For each iteration of the main loop (Lines 6–15), the value of *recent-states* at the beginning of the loop is given, followed by the list of transitions and meta-transitions that are explored at Lines 10–13, and then by the sets of states obtained after following those transitions and meta-transitions.

Suppose now that the representation system allows to define a multicycle meta-transition associated to the cycles $\mathcal{C}_1$ and $\mathcal{C}_2$. We then have MULTI-SET($\{body(\mathcal{C}_1), body(\mathcal{C}_2)\}$) = $\{\bar{t}_3\}$, with $\bar{t}_3 = (c_1, f_3, c_1)$, and

$$\begin{array}{rcl} f_3 & : & 2^{\mathbf{Z}^2} \to 2^{\mathbf{Z}^2} \quad : \quad U \mapsto U \ \cup \ \{(v_1, v_2 + k_2) \mid (v_1, v_2) \in U \ \wedge \ \text{even}(v_1) \ \wedge \ k_2 \in \mathbf{N}\} \\ & & \qquad\qquad\qquad\qquad \cup \ \{(v_1 + k_1 + 1, v_2 + k_2) \mid (v_1, v_2) \in U \ \wedge \ k_1, k_2 \in \mathbf{N}\}. \end{array}$$

Let $\mathcal{A}'$ be the ESMA obtained by adding the meta-transition $\bar{t}_3$ to the system of Figure 2.1. The details of the computation of REACHABLE($\mathcal{A}'$) are given in Figure 3.13.

3.7 Discussion

In this chapter, we have studied cycle and multicycle meta-transitions. These aim at capturing the infinite nature of the state-space resulting from the presence of cycles in the control graph of the system.

Depending on the memory domain, there might be other types of meta-transitions which could be of interest. An important example is the case of *lossy systems*, which are systems whose memory contents may at any time undergo some change (usually modeling loss of information over transmission channels). Translating a non-lossy program into a lossy one can be done very simply by adding to each control location a meta-transition expressing the possible losses at that location. As an example, a technique for creating meta-transitions modeling the loss in systems with lossy FIFO channels is presented in Chapter 7.

Using the techniques for creating meta-transitions proposed in this chapter in an actual implementation is not straightforward. The main problem is efficiency, since

1. $(c_1, \{(0,0)\})$
$$\begin{array}{ll} \xrightarrow{(c_1, x_1{+}{+}, c_1)} & (c_1, \{(1,0)\}) \\ \xrightarrow{(c_1, \mathrm{even}(x_1), c_2)} & (c_2, \{(0,0)\}) \\ \xRightarrow{\bar{t}_1} & (c_1, \{(k_1, 0) \mid k_1 \in \mathbf{N}\}) \\ \xRightarrow{\bar{t}_2} & (c_1, \{(0, k_2) \mid k_2 \in \mathbf{N}\}) \end{array}$$

2. $(c_1, \{(k_1+1, 0) \mid k_1 \in \mathbf{N}\} \cup \{(0, k_2+1) \mid k_2 \in \mathbf{N}\}), (c_2, \{(0,0)\})$
$$\begin{array}{ll} \xrightarrow{(c_1, x_1{+}{+}, c_1)} & (c_1, \{(k_1+2, 0) \mid k_1 \in \mathbf{N}\} \cup \{(1, k_2+1) \mid k_2 \in \mathbf{N}\}) \\ \xrightarrow{(c_1, \mathrm{even}(x_1), c_2)} & (c_2, \{(2k_1+2, 0) \mid k_1 \in \mathbf{N}\} \cup \{(0, k_2+1) \mid k_2 \in \mathbf{N}\}) \\ \xRightarrow{\bar{t}_1} & (c_1, \{(k_1, k_2) \mid k_1, k_2 \in \mathbf{N} \wedge (k_1 \neq 0 \vee k_2 \neq 0)\}) \\ \xRightarrow{\bar{t}_2} & (c_1, \{(2k_1, k_2) \mid k_1, k_2 \in \mathbf{N} \wedge (k_1 \neq 0 \vee k_2 \neq 0)\} \\ & \quad \cup \{(k_1+1, 0) \mid k_1 \in \mathbf{N}\}) \\ \xrightarrow{(c_2, x_2{+}{+}, c_1)} & (c_1, \{(0,1)\}) \end{array}$$

3. $(c_1, \mathbf{N}_0^2), (c_2, \{(2k_1+2, 0) \mid k_1 \in \mathbf{N}\} \cup \{(0, k_2+1) \mid k_2 \in \mathbf{N}\})$
$$\begin{array}{ll} \xrightarrow{(c_1, x_1{+}{+}, c_1)} & (c_1, \{(k_1+2, k_2+1) \mid k_1, k_2 \in \mathbf{N}\}) \\ \xrightarrow{(c_1, \mathrm{even}(x_1), c_2)} & (c_2, \{(2k_1+2, k_2+1) \mid k_1, k_2 \in \mathbf{N}\}) \\ \xRightarrow{\bar{t}_1} & (c_1, \mathbf{N}_0^2) \\ \xRightarrow{\bar{t}_2} & (c_1, \mathbf{N}_0^2) \\ \xrightarrow{(c_2, x_2{+}{+}, c_1)} & (c_1, \{(2k_1+2, 1) \mid k_1 \in \mathbf{N}\} \cup \{(0, k_2+2) \mid k_2 \in \mathbf{N}\}) \end{array}$$

4. $(c_2, \{(2k_1+2, k_2+1) \mid k_1, k_2 \in \mathbf{N}\})$
$$\xrightarrow{(c_2, x_2{+}{+}, c_1)} \quad (c_1, \{(2k_1+2, k_2+2) \mid k_1, k_2 \in \mathbf{N}\})$$

Reachable states: $(c_1, \mathbf{N}^2), (c_2, \{(2k_1, k_2) \mid k_1, k_2 \in \mathbf{N}\})$.

Figure 3.12: Example of state-space exploration with simple-cycle meta-transitions.

1. $(c_1, \{(0,0)\})$
$\xrightarrow{(c_1, x_1{+}{+}, c_1)}$ $(c_1, \{(1,0)\})$
$\xrightarrow{(c_1, \mathrm{even}(x_1), c_2)}$ $(c_2, \{(0,0)\})$
$\xRightarrow{\bar{t}_3}$ $(c_1, \mathbf{N}^2)$

2. $(c_1, \{(k_1,k_2) \mid k_1,k_2 \in \mathbf{N} \wedge (k_1 \neq 0 \vee k_2 \neq 0)\}), (c_2, \{(0,0)\})$
$\xrightarrow{(c_1, x_1{+}{+}, c_1)}$ $(c_1, \{(k_1+1,k_2) \mid k_1,k_2 \in \mathbf{N} \wedge (k_1 \neq 0 \vee k_2 \neq 0)\})$
$\xrightarrow{(c_1, \mathrm{even}(x_1), c_2)}$ $(c_2, \{(2k_1,k_2) \mid k_1,k_2 \in \mathbf{N} \wedge (k_1 \neq 0 \vee k_2 \neq 0)\})$
$\xRightarrow{\bar{t}_3}$ $(c_1, \{(k_1,k_2) \mid k_1,k_2 \in \mathbf{N} \wedge (k_1 \neq 0 \vee k_2 \neq 0)\})$
$\xrightarrow{(c_2, x_2{+}{+}, c_1)}$ $(c_1, \{(0,1)\})$

3. $(c_2, \{(2k_1,k_2) \mid k_1,k_2 \in \mathbf{N} \wedge (k_1 \neq 0 \vee k_2 \neq 0)\})$
$\xrightarrow{(c_2, x_2{+}{+}, c_1)}$ $(c_1, \{(2k_1,k_2+1) \mid k_1,k_2 \in \mathbf{N} \wedge (k_1 \neq 0 \vee k_2 \neq 0)\})$

Reachable states: $(c_1, \mathbf{N}^2), (c_2, \{(2k_1,k_2) \mid k_1,k_2 \in \mathbf{N}\})$.

Figure 3.13: Example of state-space exploration with multicycle meta-transitions.

it is probably more interesting in practice to obtain quickly a limited set of meta-transitions than to wait essentially forever for a set that guarantees termination. A positive point is that meta-transitions can be freely chosen, as far as they are consistent with Definition 3.1. A pragmatic solution is to allow the user to define some bounds, such as for instance a maximum number of cycle or multicycle meta-transition allowed per strongly connected component of the control graph. Heuristics could then be used to select between potential meta-transitions. Another solution is to let the user interact with the creation of meta-transitions.

The problem has similarities with the case of optimizing compilers [AU72], for which loop detection provides a way of optimizing code. In this context, efficiency is unhesitatingly preferred to a guarantee that every potential optimization is performed. Although a comparison of the two domains might yield interesting ideas for creating meta-transitions, those issues extend well beyond the scope of this work and are not further discussed here.

In the same way, the algorithm REACHABLE-DYNAMIC was introduced in Section 3.5 as a proof that a best state-space exploration strategy exists as far as termination is the primary concern, but does not straightforwardly translate into a usable implementation of dynamic state-space exploration. Indeed, the guarantee that the algorithm will terminate at least as often as any static state-space exploration algorithm was obtained at the cost of detecting every cycle and exploring every path up to a given depth in the state space. This leads to excessive time and space requirements, since the state space usually contains a considerable number

of cycles of a given length as well as a large number of paths leading to the same reachable state. In practice, there are two approaches to reducing the cost of dynamic exploration. The first is to impose restrictions on the cycles that are detected and on the paths that are explored during the state-space search. For instance, one may create only a bounded number of meta-transitions for each strongly connected component of the state-space graph, and only explore paths that do not visit the same state more than a given number of times. The second approach is to restrict the set of memory operations that can label transitions of the system, in such a way that considering a restricted set of cycles and of exploration paths is then sufficient for exploring totally the state-space in a finite amount of time. Those approaches are not addressed in this thesis.

Chapter 4

Properties

This chapter shows that state-space exploration (as introduced in Chapter 3) can be used to check various types of properties of programs modeled as SMAs. These properties are studied here independently of the memory domain of the SMAs, under the hypothesis that a well suited representation system is available for the sets of memory contents. For each algorithm discussed in this chapter, if there are specific additional memory operations that are required for implementing the algorithm for a particular domain, they will be pointed out.

4.1 Reachability Properties

In Chapter 3, we presented semi-algorithms for computing the set Q_R of reachable states of an SMA $\mathcal{A} = (C, c_0, M, m_0, Op, T)$. This set is returned as a finite list of pairs (control location, set of memory contents), where the sets of memory contents are represented with the help of a representation system well suited for $\mathcal{A}$. If the computation of Q_R terminates, then there are properties of $\mathcal{A}$ that can be decided on the sole basis of the information contained in Q_R.

Let Q_P be a set of states represented as a finite list of pairs (control location, representation of a set of memory contents). The *general reachability problem* consists of determining, given Q_P, whether this set contains at least one reachable state. A simple solution to this problem consists of testing the emptiness of the intersection $Q_P \cap Q_R$ (this operation can be performed as explained in Section 3.2.2). Indeed, there exists at least one reachable state in Q_P if and only if $Q_P \cap Q_R \neq \emptyset$. The *universal reachability problem* consists of determining, given Q_P, whether all the states belonging to Q_P are reachable. This problem can be solved by testing the inclusion $Q_P \subseteq Q_R$. The *restricted reachability problem* consists of determining whether a given state q_P of $\mathcal{A}$ is reachable or not. This problem can be reduced to the general reachability by taking $Q_P = \{q_P\}$.

The *boundedness problem* consists of determining, given a control location c and

a variable x_i, whether the set of values that x_i can take at c is finite or not. Let $\pi_i(U)$ denote the *projection* of a set of memory contents $U \subseteq M$ over the domain D_i of x_i, i.e., let $\pi_i(U) = \{v_i \mid (\exists v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n)((v_1, \ldots, v_n) \in U)\}$. The boundedness problem can be solved by checking whether the set $\pi_i(\mathit{values}(Q_R, c))$ is finite or not. This requires the ability to compute projections of represented sets of memory contents, as well as to decide the finiteness of such sets.

4.2 Deadlock Detection

Let $\mathcal{A}$ be an SMA. A state of $\mathcal{A}$ is a *deadlock state* if it is reachable, and if there is no state that can be reached in one step from that state (in other words, if there is no transition that can be followed from that state). For systems that are not intended to halt, such as those controlled by reactive programs, the presence of deadlocks is the reflect of design flaws. Indeed, if there is a deadlock, then it is possible for the system to reach a state from which no further action can be performed.

The *general deadlock detection problem* consists of computing the set of deadlock states of an SMA $\mathcal{A} = (C, c_0, M, m_0, \mathit{Op}, T)$. This problem can be solved as follows. We require every transition $(c, \theta, c') \in T$ to be such that the domain complement $\overline{\mathit{dom}}(\theta) = \{v \in M \mid \theta(v) = \bot\}$ of the function θ is representable (intuitively, $\overline{\mathit{dom}}(\theta)$ is the set of all the memory contents from which the transition cannot be followed). In addition, it must be possible to compute a representation of $\overline{\mathit{dom}}(\theta)$ from the specification of θ. The set Q_D of all the deadlock states of $\mathcal{A}$ can then be obtained by computing the intersection of the set of reachable states and of the set of all the states from which no transition can be followed. Formally, we have

$$Q_D = Q_R \cap \bigcup_{c \in C} \bigcap_{(c,\theta,c') \in T} \mathit{states}(c, \overline{\mathit{dom}}(\theta)).$$

Since there are only finitely many transitions in T, this formula allows one to easily compute a representation of Q_D from a representation of Q_R.

4.3 Temporal Properties

The properties studied in Sections 4.1 and 4.2 can be decided on the sole basis of the set of reachable states of the model. In this section, we investigate whether more elaborate properties can be checked. More precisely, we study properties of *infinite computations* of SMAs. Recall that an infinite computation is an infinite sequence $q_0, q_1, \ldots$ of states such that q_0 is the initial state, and $q_i \rightarrow_R q_{i+1}$ for every $i \in \mathbf{N}$. In this context, a *property* $\mathcal{P}$ of an SMA $\mathcal{A}$ is a (possibly infinite) set of infinite sequences of states. The SMA $\mathcal{A}$ *satisfies* $\mathcal{P}$, which is denoted $\mathcal{A} \models \mathcal{P}$, if each of its infinite computations belongs to $\mathcal{P}$.

4.3.1 Linear-Time Temporal Logic

Linear-time Temporal Logic, or LTL, is a language for specifying properties of infinite computations [Eme90, MP92]. In LTL, one can express properties such as "some given condition will eventually be satisfied forever" or "some condition will be satisfied infinitely often", which are not expressible solely in terms of the set of reachable states of the system. LTL properties are expressed as *formulas*. Let Λ be a finite set of *atomic propositions*. The syntax of LTL formulas is defined as follows:

- Every atomic proposition $a \in \Lambda$ is an LTL formula;
- If ψ_1 and ψ_2 are LTL formulas, then so are $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$ and $\psi_1 \,\mathcal{U}\, \psi_2$;
- If ψ_1 is an LTL formula, then so are (ψ_1), $\neg\psi_1$, $\bigcirc \psi_1$, $\Box \psi_1$ and $\Diamond \psi_1$.

The semantics of an LTL formula is defined with respect to a structure (Γ, L), where Γ is a set of *states* and $L : \Gamma \to 2^{\Lambda}$ is a *labeling function* that associates to each state the set of atomic propositions that are true in that state. The truth value of an LTL formula is then defined with respect to pairs of the form (γ, i), where $\gamma : \mathbf{N}_0 \to \Gamma$ is an infinite sequence of states, and $i \in \mathbf{N}_0$ is the index of a state γ_i in γ. Intuitively, since an infinite sequence of states can be seen as a sequence of state changes occurring at discrete time points, the index i defines the *time instant* at which the truth value of the formula is evaluated. The fact that the LTL formula ψ is true at the time instant i in the infinite sequence of states γ is denoted $(\gamma, i) \models \psi$. For each infinite sequence of states $\gamma = \gamma_1, \gamma_2, \ldots$ and time instant $i \in \mathbf{N}_0$, we define that:

- An atomic proposition $a \in \Lambda$ is true at the time instant i in γ if and only if a is true in the state γ_i. Formally, we have $(\gamma, i) \models a$ iff $a \in L(\gamma_i)$;
- A formula of the form $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$, (ψ_1) or $\neg\psi_1$ is true whenever the appropriate Boolean combination of the truth values of its sub-formulas ψ_1 and ψ_2 is true. Formally, we have

$$\begin{aligned}
(\gamma, i) \models \psi_1 \wedge \psi_2 \quad &\text{iff} \quad ((\gamma, i) \models \psi_1) \text{ and } ((\gamma, i) \models \psi_2), \\
(\gamma, i) \models \psi_1 \vee \psi_2 \quad &\text{iff} \quad ((\gamma, i) \models \psi_1) \text{ or } ((\gamma, i) \models \psi_2), \\
(\gamma, i) \models (\psi_1) \quad &\text{iff} \quad (\gamma, i) \models \psi_1, \\
(\gamma, i) \models \neg\psi_1 \quad &\text{iff} \quad \neg((\gamma, i) \models \psi_1);
\end{aligned}$$

- A formula of the form $\psi_1 \,\mathcal{U}\, \psi_2$ is true at the time instant i in γ if and only if the formula ψ_2 is true at some time instant j greater or equal to i, and the formula ψ_1 is true at all the time instants greater or equal to i and less than j. The temporal operator "$\mathcal{U}$" is thus used to express a condition that

must always be satisfied *until* another condition becomes satisfied and is read "until". Formally, we have $(\gamma, i) \models \psi_1 \,\mathcal{U}\, \psi_2$ iff $(\exists j \geq i)((\gamma, j) \models \psi_2 \wedge (\forall k, i \leq k < j)((\gamma, k) \models \psi_1))$;

- A formula of the form $\bigcirc \psi_1$ is true at the time instant i in γ if and only if the formula ψ_1 is true at the time instant $i+1$ in γ. The temporal operator "$\bigcirc$" is thus used to express a condition that must be satisfied at the *next* time instant and is read "next". Formally, we have $(\gamma, i) \models \bigcirc \psi_1$ iff $(\gamma, i+1) \models \psi_1$;

- A formula of the form $\Box \psi_1$ is true at the time instant i in γ if and only if the formula ψ_1 is true at all the time instants greater or equal to i. The temporal operator "$\Box$" is thus used to express a condition that must *always* be satisfied at the present and future time instants and is read "always". Formally, we have $(\gamma, i) \models \Box \psi_1$ iff $(\forall j \geq i)((\gamma, j) \models \psi_1)$;

- A formula of the form $\Diamond \psi_1$ is true at the time instant i in γ if and only if the formula ψ_1 is true at some time instant greater or equal to i. The temporal operator "$\Diamond$" is thus used to express a condition that must *eventually* be satisfied at some present or future time instant and is read "eventually". Formally, we have $(\gamma, i) \models \Diamond \psi_1$ iff $\neg((\gamma, i) \models \Box \neg\psi_1)$ iff $(\exists j \geq i)((\gamma, j) \models \psi_1)$.

We say that an infinite sequence of states γ *satisfies* the LTL formula ψ, which is denoted $\gamma \models \psi$, if it is such that $(\gamma, 1) \models \psi$. The property expressed by ψ with respect to the structure (Γ, L) is the set of all the infinite sequences of states $\gamma : \mathbf{N}_0 \to \Gamma$ such that $\gamma \models \psi$.

LTL formulas can also be interpreted over the computations of an SMA $\mathcal{A} = (C, c_0, M, m_0, Op, T)$. This is done by defining a function $g : S \to \Gamma$ that associates to each state of $\mathcal{A}$ a state in Γ. The LTL formula ψ is then said to be *satisfied* by the infinite computation $q_0, q_1, \ldots \in S^\omega$ of $\mathcal{A}$ if it is such that $g(q_0), g(q_1), \ldots \models \psi$. The *property of* $\mathcal{A}$ expressed by ψ is the set of all the infinite computations of $\mathcal{A}$ satisfying ψ. In practice, one can simply define g and the structure (Γ, L) by giving for each atomic proposition $a \in \Lambda$ a computable predicate

$$P_a : C \times M \to \{\mathbf{T}, \mathbf{F}\} : q \mapsto \begin{cases} \mathbf{T} & \text{if } a \in L(g(q)), \\ \mathbf{F} & \text{if } a \notin L(g(q)). \end{cases}$$

mapping each state of $\mathcal{A}$ onto the truth value of a at that state.

4.3.2 Büchi Automata

Another way of specifying properties of infinite computations is to express them as finite-state automata on infinite words [Büc62, Mul63, Tho90]. Roughly speaking, the idea is to represent a property by an automaton accepting exactly all the infinite sequences of states that satisfy the property.

We use the finite automata on infinite words introduced by Büchi [Büc62]. A Büchi automaton $\mathcal{B}$ is a tuple $(\Sigma, S, \Delta, s_0, F)$, where

- Σ is a finite *alphabet*;
- S is a finite set of *states*;
- $\Delta \subseteq S \times \Sigma \times S$ is a non-deterministic *transition relation*;
- $s_0 \in S$ is an *initial state*;
- $F \subseteq S$ is a set of *accepting states*.

A *run* of $\mathcal{B}$ over an infinite word $w = a_0 a_1 \cdots \in \Sigma^\omega$ is an infinite sequence of states $s_0, s_1, \ldots$ such that s_0 is the initial state of $\mathcal{B}$, and $(s_i, a_i, s_{i+1}) \in \Delta$ for every $i \in \mathbf{N}$. A run $s_0, s_1, \ldots$ is *accepting* if there are infinitely many $i \in \mathbf{N}$ such that $s_i \in F$. The infinite word $w \in \Sigma^\omega$ is *accepted* by $\mathcal{B}$ if $\mathcal{B}$ has an accepting run on w. The set $L(\mathcal{B})$ of all the infinite words accepted by $\mathcal{B}$ is the *language* accepted by $\mathcal{B}$.

The property expressed by $\mathcal{B}$ is defined with respect to a finite set Λ of atomic propositions and a labeling function $L : \Sigma \to 2^\Lambda$ that associates to each symbol in the alphabet of $\mathcal{B}$ the set of atomic propositions that are true for that symbol. The property $\mathcal{P}$ expressed by $\mathcal{B}$ is then the set

$$\mathcal{P} = \{L(a_0), L(a_1), L(a_2), \ldots \in (2^\Lambda)^\omega \mid (\exists a_0, a_1, a_2, \ldots \in \Sigma)(a_0 a_1 a_2 \cdots \in L(\mathcal{B}))\}$$

of all the sequences of interpretations of atomic propositions that are associated to words accepted by $\mathcal{B}$. We say that an infinite sequence of interpretations $\lambda \in (2^\Lambda)^\omega$ *satisfies* the property $\mathcal{P}$ expressed by $\mathcal{B}$, which is denoted $\lambda \models \mathcal{P}$, if we have $\lambda \in \mathcal{P}$.

Properties expressed by Büchi automata can also be interpreted over the computations of an SMA $\mathcal{A} = (C, c_0, M, m_0, Op, T)$. This is done by defining a function $g : S \to 2^\Lambda$ that associates to each state of $\mathcal{A}$ a set of atomic propositions that are true at that state. The property $\mathcal{P}$ expressed by a Büchi automaton $\mathcal{B}$ is then said to be *satisfied* by the infinite computation $q_0, q_1, \ldots \in S^\omega$ of $\mathcal{A}$ if it is such that $g(q_0), g(q_1), \ldots \models \mathcal{P}$. The *property of* $\mathcal{A}$ expressed by $\mathcal{B}$ is the set of all the infinite computations of $\mathcal{A}$ satisfying $\mathcal{P}$. Like for LTL formulas, one can simply define g by giving for each atomic proposition $a \in \Lambda$ a computable predicate $P_a : C \times M \to \{\mathbf{T}, \mathbf{F}\}$ mapping each state of $\mathcal{A}$ onto the truth value of a at that state.

The properties of SMAs that can be expressed as Büchi automata are said to be *ω-regular*. It is shown in [VW94] that the class of properties that can be expressed in LTL is a strict subset of the class of ω-regular properties. The proof is constructive and yields an algorithm for converting any LTL property into a Büchi automaton. This result is very useful in practice, since it is often convenient for algorithms to work with automata, even though it can be more natural for humans to specify

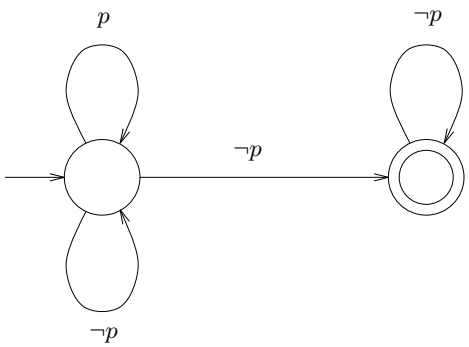

Figure 4.1: Büchi automaton.

properties as LTL formulas. In addition, this approach is compatible with arbitrary extensions of LTL, provided that they allow the translation of formulas into Büchi automata [Wol83]. The translation of an LTL formula ψ into a Büchi automaton is of size $O(2^{|\psi|})$ and can be computed in time $O(2^{|\psi|})$, where $|\psi|$ denotes the number of symbols composing ψ.

4.3.3 Example

Let us consider the SMA $\mathcal{A} = (C, c_0, M, m_0, Op, T)$ depicted in Figure 2.1. The property $\mathcal{P}$ = "x_2 *is not infinitely often odd*" is not satisfied by $\mathcal{A}$. Indeed, this automaton admits the infinite computation $(c_1, (0,0))$, $(c_2, (0,0))$, $(c_1, (0,1))$, $(c_1, (1,1))$, $(c_1, (2,1))$, $(c_1, (3,1))$, ..., in which the value of x_2 is infinitely often equal to 1.

The property $\mathcal{P}$ can be expressed as the LTL formula "$\neg \Box \Diamond p$", where the atomic proposition p is associated with the state predicate P_p, defined as

$$P_p : C \times M \to \{\mathbf{T}, \mathbf{F}\} : (c, (v_1, v_2)) \mapsto \begin{cases} \mathbf{T} & \text{if } v_2 \text{ is odd,} \\ \mathbf{F} & \text{if } v_2 \text{ is even.} \end{cases}$$

The property $\mathcal{P}$ can equivalently be expressed as the LTL formula "$\Diamond \Box \neg p$", which literally translates into "x_2 *will eventually be always even*". Indeed, a sequence of states $(c_1, (v_{1,1}, v_{2,1}))$, $(c_2, (v_{1,2}, v_{2,2}))$, ... satisfies $\mathcal{P}$ if and only if there exists $i \in \mathbf{N}_0$ such that $v_{2,j}$ is even for every $j > i$.

A Büchi automaton $\mathcal{B}$ expressing $\mathcal{P}$ is given in Figure 4.1. Its alphabet contains the symbols p and $\neg p$, which are respectively associated with the predicates P_p (as defined above) and $\neg P_p$ (the complement of P_p). The language $L(\mathcal{B})$ accepted by $\mathcal{B}$ is the set of all the infinite words over the alphabet $\{p, \neg p\}$ containing only finitely many p symbols.

4.4 Model Checking

Let $\mathcal{A}$ be an SMA. The *model-checking* problem consists of determining, given a property $\mathcal{P}$ expressed as an LTL formula or as a Büchi automaton, whether $\mathcal{A}$ satisfies $\mathcal{P}$. We will first present a classical solution to this problem, the *automata-theoretic* approach [VW86], which is applicable to systems with a finite state space. For systems with an infinite state space, it is known that model checking is undecidable if the memory domain is sufficiently expressive [EN94, Fin94, HKPV95, CFI96, ACJT96, AJ96, Esp97]. For such systems, we will give a partial solution based on an extension of the automata-theoretic approach.

4.4.1 Finite-State Systems

If $\mathcal{A}$ has a finite number of reachable states, then the model-checking problem can be solved by performing the following operations:

1. Building a Büchi automaton $\mathcal{B}_{\neg\mathcal{P}}$ accepting the *complement* of $\mathcal{P}$ (the set of all the words whose corresponding sequence of propositional interpretations does not belong to $\mathcal{P}$);

2. Computing the product $\mathcal{B}_{\mathcal{A},\neg\mathcal{P}} = \mathcal{A} \times \mathcal{B}_{\neg\mathcal{P}}$, which is a Büchi automaton such that each of its accepting runs corresponds to an infinite computation of $\mathcal{A}$ accepted by $\mathcal{B}_{\neg\mathcal{P}}$;

3. Checking whether the language accepted by $\mathcal{B}_{\mathcal{A},\neg\mathcal{P}}$ is empty or not. $L(\mathcal{B}_{\mathcal{A},\neg\mathcal{P}})$ is empty if and only if $\mathcal{A}$ does not have an infinite computation that is accepted by $\mathcal{B}_{\neg\mathcal{P}}$, hence if and only if $\mathcal{A}$ satisfies $\mathcal{P}$.

Carrying out the first step is easy. If $\mathcal{P}$ is specified as an LTL formula, then all one has to do is to translate $\neg\mathcal{P}$ (which is also an LTL formula) into a Büchi automaton (this can be done with the algorithm given in [GPVW95]). If $\mathcal{P}$ is expressed as a Büchi automaton $\mathcal{B}_{\mathcal{P}}$, then one has to *complement* this automaton, i.e., to build an automaton accepting the complement of $L(\mathcal{B}_{\mathcal{P}})$. Algorithms for complementing Büchi automata can be found in [Büc62, Péc86, SVW87, Saf88]. The complementation operation can be quite costly; indeed, the automata returned by the nearly optimal algorithm presented in [Saf88] are of size $2^{O(N \log N)}$, where N is the number of transitions of $\mathcal{B}_{\mathcal{P}}$.

The second step is performed as follows. Let $\mathcal{A} = (C, c_0, M, m_0, Op, T)$, $\mathcal{B}_{\neg\mathcal{P}} = (\Sigma_{\neg\mathcal{P}}, S_{\neg\mathcal{P}}, \Delta_{\neg\mathcal{P}}, s_{0\neg\mathcal{P}}, F_{\neg\mathcal{P}})$, let (Q, q_0, R) be the semantics of $\mathcal{A}$, and let Q_R be the set of reachable states of $\mathcal{A}$ (finite by hypothesis). The product $\mathcal{A} \times \mathcal{B}_{\neg\mathcal{P}}$ is the Büchi automaton $\mathcal{B} = (\Sigma, S, \Delta, s_0, F)$, where:

- $\Sigma = \{\tau\}$ (the alphabet of $\mathcal{B}$ contains a single dummy symbol τ. $\mathcal{B}$ can indeed be seen as an *inputless* Büchi automaton);

- $S = Q_R \times S_{\neg\mathcal{P}}$ (the set of states of $\mathcal{B}$ is the Cartesian product of the (finite) set of reachable states of $\mathcal{A}$ and of the set of states of $\mathcal{B}_{\neg\mathcal{P}}$);
- $\Delta = \{((c_1, m_1, s_1), \tau, (c_2, m_2, s_2)) \in S \times \Sigma \times S \mid (\exists a \in \Sigma_{\neg\mathcal{P}})((c_1, m_1) \rightarrow_R (c_2, m_2) \wedge (s_1, a, s_2) \in \Delta_{\neg\mathcal{P}} \wedge P_a((c_1, m_1)))\}$, where P_a is the state predicate of $\mathcal{B}_{\neg\mathcal{P}}$ associated to the symbol a (there is a transition between two states of $\mathcal{B}$ if and only if there are transitions between the corresponding states of $\mathcal{A}$ and of $\mathcal{B}_{\neg\mathcal{P}}$, and the appropriate state predicate is satisfied);
- $s_0 = (c_0, m_0, s_{0\neg\mathcal{P}})$ (the initial state of $\mathcal{B}$ is composed of the initial states of $\mathcal{A}$ and of $\mathcal{B}_{\neg\mathcal{P}}$);
- $F = Q_R \times F_{\neg\mathcal{P}}$ (a state of $\mathcal{B}$ is accepting if and only if the corresponding state of $\mathcal{B}_{\neg\mathcal{P}}$ is accepting).

This construction is correct thanks to the following result.

Theorem 4.1 *Let $(c_1, m_1), (c_2, m_2), \ldots$ be states of $\mathcal{A}$. The automaton $\mathcal{B} = \mathcal{A} \times \mathcal{B}_{\neg\mathcal{P}}$ has an accepting run of the form $(c_1, m_1, s_1), (c_2, m_2, s_2), \ldots$, where $s_1, s_2, \ldots$ are states of $\mathcal{B}_{\neg\mathcal{P}}$, if and only if the infinite sequence of states $(c_1, m_1), (c_2, m_2), \ldots$ is an infinite computation of $\mathcal{A}$ that does not satisfy $\mathcal{P}$.*

Proof The relationship between the runs of $\mathcal{B}$ and the infinite computations of $\mathcal{A}$ is immediate by construction. □

As a direct consequence of the previous construction, the number of transitions composing $\mathcal{B}$ can be as large as $O(N_{\mathcal{A}} N_{\mathcal{B}_{\neg\mathcal{P}}})$, where $N_{\mathcal{A}}$ and $N_{\mathcal{B}_{\neg\mathcal{P}}}$ denote respectively the number of edges in the state space of $\mathcal{A}$, and the number of transitions of $\mathcal{B}_{\neg\mathcal{P}}$.

It remains to show how to perform the third step of the model-checking procedure, which consists of determining whether the language accepted by the Büchi automaton $\mathcal{B} = (\Sigma, S, \Delta, s_0, F)$ is empty or not. This can be done by performing a reachability analysis in the graph (S, Δ). First, one computes the maximal strongly connected components of this graph (a strongly connected component is a set of nodes such that any of them is reachable from all of them). Next, one checks whether there exists an accepting state reachable from s_0 which belongs to a non trivial[1] strongly connected component. Indeed, $\mathcal{B}$ has an accepting run if and only if there exists an accepting state that is reachable from the initial state, and reachable from itself by following at least one transition. If properly implemented, those operations can be performed in $O(|\Delta|)$ time, where $|\Delta|$ is the number of transitions of $\mathcal{B}$ [Tar83]. A nice optimization of this method that avoids explicitly constructing the maximal strongly connected components and uses only $O(|S|)$ space, where $|S|$ is the number of states of $\mathcal{B}$, is presented in [CVWY92].

[1] A strongly connected component is *non trivial* if its nodes are linked to each other by at least one transition.

In practice, there is no need for building and storing the product automaton $\mathcal{B} = \mathcal{A} \times \mathcal{B}_{\neg\mathcal{P}}$ in its entirety. Instead, this automaton can be generated *on-the-fly*, meaning that the states and transitions that are needed during the test of emptiness at Step 3 can be produced on demand rather than systematically.

4.4.2 Infinite-State Systems

The automata-theoretic approach cannot be applied straightforwardly to the model checking of infinite-state systems. The main problem is that the product of an SMA with an infinite number of reachable states by a Büchi automaton does not necessarily have a finite number of states, and thus cannot always be expressed as a Büchi automaton. The solution is to express this product as a *Structured-Memory Büchi Automaton*, which is an SMA associated with an accepting condition on its control locations. Structured-Memory Büchi Automata are finite representations of infinite-state machines accepting infinite words, just like Structured-Memory Automata are finite representations of infinite-state systems.

Formally, a *Structured-Memory Büchi Automaton* (or *SMBA* in short), is a tuple $(C, c_0, M, m_0, Op, T, F)$, where:

- (C, c_0, M, m_0, Op, T) is an SMA;
- $F \subseteq C$ is a finite set of *accepting control locations*.

The notions of control location, memory domain, memory content, state, initial state, one-step reachability and reachable states of an SMBA $\mathcal{B} = (C, c_0, M, m_0, Op, T, F)$ are defined identically to those of its underlying SMA (C, c_0, M, m_0, Op, T). A *run* of $\mathcal{B}$ is an infinite sequence of states $(c_0, m_0), (c_1, m_1), \ldots$ such that (c_0, m_0) is the initial state of $\mathcal{B}$, and $(c_i, m_i) \rightarrow_R (c_{i+1}, m_{i+1})$ for every $i \in \mathbf{N}$, where R is the one-step reachability relation of $\mathcal{B}$. The run is said to be *accepting* if there are infinitely many $i \in \mathbf{N}$ such that $c_i \in F$. The set $L(\mathcal{B})$ of all the accepting runs of $\mathcal{B}$ is the *language* accepted by $\mathcal{B}$.

Checking whether an infinite-state SMA $\mathcal{A}$ satisfies a property $\mathcal{P}$ can be done as follows. The procedure is an extension of the automata-theoretic approach to model checking for finite-state SMAs.

1. One builds a Büchi automaton $\mathcal{B}_{\neg\mathcal{P}}$ accepting the complement of $\mathcal{P}$;
2. One computes the product $\mathcal{B}_{\mathcal{A},\neg\mathcal{P}} = \mathcal{A} \times \mathcal{B}_{\neg\mathcal{P}}$, which is an SMBA such that each of its accepting runs corresponds to an infinite computation of $\mathcal{A}$ accepted by $\mathcal{B}_{\neg\mathcal{P}}$;
3. One checks whether the language accepted by $\mathcal{B}_{\mathcal{A},\neg\mathcal{P}}$ is empty or not. This language is empty if and only if $\mathcal{A}$ satisfies $\mathcal{P}$.

The first step is identical to the first step of the model-checking method for finite-state systems introduced in Section 4.4.1.

The second step is performed as follows. Let $\mathcal{A} = (C_\mathcal{A}, c_{0\mathcal{A}}, M_\mathcal{A}, m_{0\mathcal{A}}, Op_\mathcal{A}, T_\mathcal{A})$ and $\mathcal{B}_{\neg\mathcal{P}} = (\Sigma_{\neg\mathcal{P}}, S_{\neg\mathcal{P}}, \Delta_{\neg\mathcal{P}}, s_{0\neg\mathcal{P}}, F_{\neg\mathcal{P}})$. The product $\mathcal{A} \times \mathcal{B}_{\neg\mathcal{P}}$ is the SMBA $\mathcal{B} = (C, c_0, M, m_0, Op, T, F)$, where:

- $C = C_\mathcal{A} \times S_{\neg\mathcal{P}}$ (each potential state of $\mathcal{B}$ is composed of a control location of $\mathcal{A}$, and a state of $\mathcal{B}_{\neg\mathcal{P}}$);
- $c_0 = (c_{0\mathcal{A}}, s_{0\neg\mathcal{P}})$ (the initial control location of $\mathcal{B}$ is composed of the initial control location of $\mathcal{A}$ and the initial state of $\mathcal{B}_{\neg\mathcal{P}}$);
- $M = M_\mathcal{A}$ (the memory domain of $\mathcal{B}$ is identical to the one of $\mathcal{A}$);
- $m_0 = m_{0\mathcal{A}}$ (the initial memory content of $\mathcal{B}$ is identical to the one of $\mathcal{A}$);
- $Op = \{P_{a,c} \cap \theta \mid c \in C_\mathcal{A} \wedge a \in \Sigma_{\neg\mathcal{P}} \wedge \theta \in Op_\mathcal{A}\}$, where $P_{a,c}$ denotes the predicate satisfied by every memory content $m \in M$ such that (c, m) satisfies the state predicate P_a. The symbol "$\cap$" denotes the intersection of functions. Formally, for every control location $c \in C_\mathcal{A}$, symbol $a \in \Sigma_{\neg\mathcal{P}}$ and operation $\theta \in Op_\mathcal{A}$, we have

  $$P_{a,c} \cap \theta : M \to M : m \mapsto \begin{cases} \theta(m) & \text{if } P_a((c,m)), \\ \bot & \text{if } \neg P_a((c,m)). \end{cases}$$

  Intuitively, the operations labeling the transitions of $\mathcal{B}$ are conjunctions of operations labeling edges of the control graph of $\mathcal{A}$ and of state predicates of $\mathcal{B}_{\mathcal{A},\neg\mathcal{P}}$;
- $T = \{((c_1, s_1), P_{a,c_1} \cap \theta, (c_2, s_2)) \mid (\exists a \in \Sigma_{\neg\mathcal{P}})((c_1, \theta, c_2) \in T_\mathcal{A} \wedge (s_1, a, s_2) \in \Delta_{\neg\mathcal{P}})\}$ (there is a transition between two states of $\mathcal{B}$ if and only if there is an edge between the corresponding control locations of $\mathcal{A}$ and a transition between the corresponding states of $\mathcal{B}_{\neg\mathcal{P}}$. The memory operation performed while following that transition consists of executing the operation labeling the edge of $\mathcal{A}$, provided that the memory content satisfies the state predicate associated to the transition of $\mathcal{B}_{\neg\mathcal{P}}$);
- $F = C_\mathcal{A} \times F_{\neg\mathcal{P}}$ (a control location of $\mathcal{B}$ is accepting if and only if the corresponding state of $\mathcal{B}_{\neg\mathcal{P}}$ is accepting).

This construction is correct thanks to the following result.

Theorem 4.2 *Let $(c_1, m_1), (c_2, m_2), \ldots$ be states of $\mathcal{A}$. The automaton $\mathcal{B} = \mathcal{A} \times \mathcal{B}_{\neg\mathcal{P}}$ has an accepting run of the form $((c_1, s_1), m_1), ((c_2, s_2), m_2), \ldots$, where $s_1, s_2, \ldots$ are states of $\mathcal{B}_{\neg\mathcal{P}}$, if and only if the infinite sequence of states $(c_1, m_1), (c_2, m_2), \ldots$ is an infinite computation of $\mathcal{A}$ that does not satisfy $\mathcal{P}$.*

Proof The relationship between the runs of $\mathcal{B}$ and the infinite computations of $\mathcal{A}$ is immediate by construction. □

It remains to show how to perform the third step of the model-checking procedure, which consists of determining whether the language accepted by the SMBA $\mathcal{B}$ is empty or not. This problem, which is a crucial issue of infinite-state model checking, is discussed in Section 4.5. As will be shown in that section, the test of emptiness for SMBAs is undecidable for sufficiently expressive memory domains. The solution will thus consist of a semi-decision procedure. The approach we will propose is based on the state-space exploration of SMAs (as introduced in Chapter 3). Systems for which a full decision procedure can be obtained will be studied in Chapter 5.

As in the case of finite-state systems, there is no need for storing explicitly the product automaton computed at the second step of the model-checking procedure; the components of this automaton can be generated on demand by an *on-the-fly* algorithm.

4.5 Testing the Emptiness of SMBAs

The problem investigated here consists of checking whether an SMBA has an accepting run or not. As we will show, this problem cannot be fully solved if the memory domain of the SMBA is sufficiently expressive. We first define precisely this notion of expressiveness.

4.5.1 Expressiveness of Memory Domains

We characterize expressiveness by relating SMAs to Turing machines. A *Turing machine* [Tur36, HU79] is a state machine with a finite control and an infinite memory, the latter being structured as an unbounded *tape*. The tape is divided into an infinite number of *locations*, each containing a symbol from a finite alphabet. The content of the tape can thus be seen as a word over this alphabet. The tape is accessed by means of a *head*, which can move forwards and backwards across the tape reading and writing symbols. The location of the head together with the tape content characterize the state of the memory of the Turing machine. We use *inputless* Turing machines, which are machines whose tape content is initially empty.

Formally, a Turing machine is a tuple $(Q, \Gamma, \delta, q_0, \#, F)$, where

- Q is a finite set of *states*;
- Γ is a finite *tape alphabet*;
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is a *transition function* ("L" and "R" respectively indicate a left and a right move of the head);

- $q_0 \in Q$ is an *initial state*;
- $\# \in \Gamma$ is a *blank symbol*;
- $F \subseteq Q$ is a set of *accepting states*.

The semantics of a Turing machine $(Q, \Gamma, \delta, q_0, \#, F)$ is defined in terms of *configurations*. A configuration contains all the information needed for continuing the execution of the machine, i.e., it is composed of the current state, the tape content, and the position of the head. The initial configuration corresponds to the initial state, a tape containing only blank symbols, and the head positioned at the leftmost location of the tape. At any time, only a finite number of tape locations have been accessed by the machine, and thus the tape content can be unambiguously characterized by one of its finite prefixes (such that the remaining tape locations only contain blank symbols). Formally, a configuration is a triple (q, w_L, w_R), where

- $q \in Q$ is a state;
- $w_L \in \Gamma^*$ is the content of the tape between the leftmost location and the last location before the head (those locations included);
- $w_R \in \Gamma^*$ is the content of the tape between the head location and the last location containing a non-blank symbol (those locations included), or the empty string ε if, from the head location, the tape content is only composed of blank symbols.

Let $(q_1, w_{L,1}, w_{R,1})$ be a configuration, $a \in \Gamma$ be the rightmost symbol of $w_{L,1}$ (such that $w_{L,1} = w'_{L,1}a$, a being undefined if $w_{L,1} = \varepsilon$), and b be the leftmost symbol of $w_{R,1}$ (such that $w_{R,1} = bw'_{R,1}$, or such that $b = \#$ and $w'_{R,1} = \varepsilon$ if $w_{R,1} = \varepsilon$). A configuration $(q_2, w_{L,2}, w_{R,2})$ is *reachable in one step* from $(q_1, w_{L,1}, w_{R,1})$, which is denoted $(q_1, w_{L,1}, w_{R,1}) \vdash (q_2, w_{L,2}, w_{R,2})$, if $q_1 \notin F$ and

- $\delta(q_1, b) = (q_2, b', R)$, $w_{L,2} = w_{L,1}b'$ and $w_{R,2} = w'_{R,1}$, or
- $\delta(q_1, b) = (q_2, b', L)$, $w_{L,1} \neq \varepsilon$, $w_{L,2} = w'_{L,1}$ and
  - $w_{R,2} = ab'w'_{R,1}$ if $b' \neq \#$ or $w'_{R,1} \neq \varepsilon$;
  - $w_{R,2} = a$ if $a \neq \#$, $b' = \#$ and $w'_{R,1} = \varepsilon$;
  - $w_{R,2} = \varepsilon$ if $a = \#$, $b' = \#$ and $w'_{R,1} = \varepsilon$.

The configuration (q, w_L, w_R) is *reachable* if there exist $k \in \mathbf{N}_0$ and $(q_1, w_{L,1}, w_{R,1})$, $(q_2, w_{L,2}, w_{R,2})$, …, $(q_k, w_{L,k}, w_{R,k}) \in Q \times \Gamma^* \times \Gamma^*$ such that $q_1 = q_0$, $w_{L,1} = w_{R,1} = \varepsilon$, $q_k = q$, $w_{L,k} = w_L$, $w_{R,k} = w_R$, and $(q_i, w_{L,i}, w_{R,i}) \vdash (q_{i+1}, w_{L,i+1}, w_{R,i+1})$ for every i such that $0 < i < k$. The longest (possibly infinite) sequence $(q_1, w_{L,1}$,

$w_{R,1})$, $(q_2, w_{L,2}, w_{R,2})$, $(q_3, w_{L,3}, w_{R,3})$, ... of configurations such that $q_1 = q_0$, $w_{L,1} = w_{R,1} = \varepsilon$ and $(q_i, w_{L,i}, w_{R,i}) \vdash (q_{i+1}, w_{L,i+1}, w_{R,i+1})$ for every $i \in \mathbf{N}_0$ is the *execution* of the Turing machine. If this execution is finite (either because its last configuration contains an accepting state, or because the transition function is not defined for the last configuration), then the machine is said to *halt*.

We are now ready to relate SMAs to Turing machines. Roughly speaking, we consider a class of SMAs to be "sufficiently expressive" if it is possible to simulate every arbitrary Turing machine by an SMA belonging to the class. Formally, we have the following definition.

Definition 4.3 *Let M be a memory domain, and $Op \subseteq M \to M$ be a set of memory operations. The pair (M, Op) is* Turing-expressive *if there exists a computable function α converting every Turing machine $\mathcal{M} = (Q, \Gamma, \delta, q_0, \#, F)$ into an SMA $\mathcal{A} = \alpha(\mathcal{M}) = (C, c_0, M, m_0, Op, T)$ such that:*

- *Every state $q \in Q$ is associated with a unique control location $\alpha(q) \in C$;*
- *Every potential tape content $w \in \Gamma^*(\#)^\omega$ is associated with a set of corresponding memory contents $\alpha(w) \subseteq M$. For every $w_1, w_2 \in \Gamma^*(\#)^\omega$ such that $w_1 \neq w_2$, we have $\alpha(w_1) \cap \alpha(w_2) = \emptyset$;*
- *$\alpha(q_0) = c_0$ (the initial state of $\mathcal{M}$ is associated with the initial control location of $\mathcal{A}$);*
- *$\alpha((\#)^\omega) = \{m_0\}$ (the initial tape content of $\mathcal{M}$, which is only composed of blank symbols, is associated with the initial memory content of $\mathcal{A}$);*
- *A configuration $(q_2, w_{L,2}, w_{R,2}) \in Q \times \Gamma^* \times \Gamma^*$ is* reachable in one step *from a configuration $(q_1, w_{L,1}, w_{R,1}) \in Q \times \Gamma^* \times \Gamma^*$ if and only if a state of $\mathcal{A}$ corresponding to the latter is reachable (not necessarily in one step) from a state associated to the former. In other words, $(q_1, w_{L,1}, w_{R,1}) \vdash (q_2, w_{L,2}, w_{R,2})$ if and only if there exist $m_1 \in \alpha(w_{L,1}w_{R,1}(\#)^\omega)$ and $m_2 \in \alpha(w_{L,2}w_{R,2}(\#)^\omega)$ such that*
$$(\alpha(q_1), m_1) \to_R^* (\alpha(q_2), m_2),$$
where R is the reachability relation of $\mathcal{A}$;
- *T is deterministic, i.e., from every state in $C \times M$, there is at most one state reachable in one step.*

By extension, the set of all the SMAs sharing the same Turing-expressive pair of memory domain and operations (M, Op) is called a Turing-expressive class *of SMAs. Turing-expressive classes of Structured-Memory Büchi Automata are defined similarly.*

Unsurprisingly, most classes of SMAs used for modeling real-life systems are Turing-expressive. In Chapters 7 and 8, we will recall well-known results stating that SMAs using FIFO channels with send and receive operations and those using integer variables with linear operations are both Turing-expressive.

4.5.2 Undecidability of the Emptiness Problem

Let us show that the emptiness problem cannot be solved for Turing-expressive classes of SMBAs.

Theorem 4.4 *Let (M, Op) be a Turing-expressive pair of memory domain and operations. The problem of determining whether an arbitrary SMBA using those domain and operations has an accepting run is undecidable.*

Proof The proof is by reduction from the halting problem for Turing machines. Let $\mathcal{M} = (Q, \Gamma, \delta, q_0, \#, F)$ be a Turing machine. From this Turing machine, we construct the SMBA $\mathcal{B} = (C, c_0, M, m_0, Op, T, F')$ such that

- (C, c_0, M, m_0, Op, T) is the SMA obtained as a result of converting $\mathcal{M}$ as described in Definition 4.3. Since (M, Op) is Turing-expressive, this conversion can be performed algorithmically;
- F' contains all the control locations in C that are the image of a state of $\mathcal{M}$ by the conversion function.

As a consequence of this construction and of Definition 4.3, $\mathcal{B}$ has an accepting run if and only if $\mathcal{M}$ does not halt. Indeed, if $\mathcal{B}$ has an accepting run $(c_0, m_0), (c_1, m_1), \ldots$, then this run contains infinitely many c_i associated to states of $\mathcal{M}$. The run can thus be translated into an infinite execution of $\mathcal{M}$, and therefore $\mathcal{M}$ does not halt. Reciprocally, if $\mathcal{M}$ does not halt, then it has an infinite execution which can be translated into an accepting run of $\mathcal{B}$.

The empty word halting problem for Turing machines (or, more precisely, its complement) has thus been reduced to the emptiness problem for SMBAs. Since the former is undecidable [HU79], the latter is undecidable as well. □

4.5.3 Semi-Decision Procedure

From now on, we will assume that the SMBAs that we consider belong to Turing-expressive classes. Theorem 4.4 implies that it is impossible to check algorithmically all the runs of such an SMBA in order to determine whether its accepted language is empty or not.

It is nonetheless possible to obtain a partial decision procedure, by only searching for runs belonging to a restricted set (as opposed to all of them). The result would

be an algorithm which might give out false negatives (failing to discover an accepting run), but no false positives (finding an accepting run when there is none). Using such an algorithm in the model-checking procedure introduced in Section 4.4.2 would yield an algorithm that would be able to detect the presence of some errors (by establishing that the property is not satisfied by the model), but not their absence. This is consistent with our view of verification: we know that a proof of correctness for a model does not translate into a certainty of correctness for the actual system unless lots of hypotheses are assumed. As a full decision procedure cannot be obtained for the class of models we consider, we argue that looking for as many types of errors as possible is practically more useful than struggling to prove absolute correctness, which is anyway hypothetical.

Let $\mathcal{B} = (C, c_0, M, m_0, Op, T, F)$ be an SMBA. A run $(c_0, m_0), (c_1, m_1), \ldots$ of $\mathcal{B}$ is accepting if there are infinitely many c_i that belong to F. Since F is finite, this implies that there is an accepting control location $c \in F$ which is visited infinitely often by the run, i.e., such that there are infinitely many $i \in \mathbf{N}$ for which $c_i = c$. The partial decision procedure we propose is based on the concept of *meta-transition* introduced in Section 3.2.1. Sketchily, the idea consists of first associating some set of meta-transitions to the underlying SMA $\mathcal{A} = (C, c_0, M, m_0, Op, T)$ of $\mathcal{B}$, obtaining the ESMA $\mathcal{A}' = (C, c_0, M, m_0, Op, T, \bar{T})$ and ESMBA[2] $\mathcal{B}'$. Then, one searches only for sequences of states of $\mathcal{B}'$ in which an accepting control location is visited infinitely often by repeatedly following the same meta-transition.

There is a small problem with this approach, in that the existence of such a sequence does not always imply that $\mathcal{B}$ has an accepting run. Let us consider an example. Suppose that $F = \{c_0\}$, and that T is empty. We can associate to $\mathcal{A}$ the set $T' = \{\bar{t}\,\}$ containing the trivial meta-transition $\bar{t} = (c_0, \mathrm{id}, c_0)$, where

$$\mathrm{id} \,:\, 2^M \to 2^M \,:\, U \mapsto U$$

is the identity function (the meta-transition $\bar{t}$ clearly satisfies the conditions of Definition 3.1). The infinite sequence of states $(c_0, m_0), (c_0, m_0), \ldots$ can then be obtained by following repeatedly $\bar{t}$ from the initial state of $\mathcal{A}'$, even though this sequence does not correspond to an accepting run of $\mathcal{B}$. Indeed, there is no way of reaching (c_0, m_0) from itself by following a nonempty path of transitions.

The solution is to impose that every time a state q_2 is obtained from a state q_1 by following a meta-transition in $\bar{T}$, there must exist a nonempty sequence of transitions in T going from q_1 to q_2. This leads to the following definitions.

Definition 4.5 *A meta-transition $(c_1, f, c_2) \in \bar{T}$ of $\mathcal{A}'$ is* open *for the memory content $m_1 \in M$ if there exists $m_2 \in M$ such that $m_2 \in f(\{m_1\})$ and $(c_1, m_1) \to_R^+ (c_2, m_2)$, where R is the one-step reachability relation of $\mathcal{A}'$.*

[2] An *Extended SMBA*, or ESMBA, is simply an SMBA associated with a finite set of meta-transitions.

Definition 4.6 *A meta-transition* $(c_1, f, c_2) \in \bar{T}$ *of* $\mathcal{A}'$ *is* repeatedly open *for the memory content* $m_1 \in M$ *if*

- $c_1 = c_2$, *and*
- *There exist* $m_2, m_3, \ldots \in M$ *such that for every* $i \in \mathbf{N}_0$, $m_{i+1} \in f(\{m_i\})$ *and* $(c_i, m_i) \rightarrow_R^+ (c_{i+1}, m_{i+1})$, *where* R *is the one-step reachability relation of* $\mathcal{A}'$.

The partial decision procedure for checking the emptiness of $\mathcal{B}$ simply consists of checking if there is a run in which a repeatedly open meta-transition is followed from a reachable state whose control location is accepting. Indeed, if there exist $c \in F$ and $m \in M$ such that (c, m) is reachable as well as a meta-transition $(c, f, c) \in \bar{T}$ repeatedly open for m, then the previous definitions imply that there exist $m_2, m_3, \ldots \in M$ such that

$$(c_0, m_0) \rightarrow_R^* (c, m) \rightarrow_R^+ (c, m_2) \rightarrow_R^+ (c, m_3) \cdots.$$

Since c is accepting, this sequence defines an accepting run of $\mathcal{B}$.

Testing the emptiness of SMBAs has thus been reduced to performing a reachability analysis, followed by a search for repeatedly open meta-transitions. The former problem has been studied in Chapter 3. Let us now address the latter one.

Checking exactly whether a meta-transition is repeatedly open for some given memory content is in general impractical. Indeed, according to Definitions 4.5 and 4.6, the entire transition relation of the SMBA has to be taken into account in order to check whether following the meta-transition amounts to following a nonempty sequence of transitions. A more convenient approach consists of considering only the part of the transition relation that is relevant to the meta-transition. For instance, in the case of a cycle (see Section 3.4.1) or a multicycle (see Section 3.4.2) meta-transition, only the transitions composing the cycle(s) from which the meta-transition is derived may be taken into account. This leads to a sufficient condition for repeated openness. The fact that this condition may be not necessary is not at all problematic, since we know that checking for all potential accepting runs is impossible anyway.

In practice, the search for repeatedly open meta-transitions is performed as follows. We associate to each meta-transition $\bar{t}$ belonging to $\bar{T}$ a set OPEN-SET$(\bar{t}) \subseteq M$ such that $\bar{t}$ is known to be repeatedly open for each memory content belonging to that set (this set can be chosen arbitrarily as far as correctness is concerned, provided that the previous condition is satisfied). If no such set can be determined at the time $\bar{t}$ is created, or if the origin and destination control locations of the meta-transition differ, then we have OPEN-SET$(\bar{t}) = \emptyset$.

Checking whether there exists a meta-transition in T' that is repeatedly open for a reachable memory content can now be done by first computing the set Q_R of reachable states of $\mathcal{A}'$, and then testing the emptiness of each set of the form

```
function SMBA-EMPTY?(SMBA (C, c_0, M, m_0, Op, T, F), set of meta-transitions T̄) : {F, ?}
 1:  var Q_R : set of states;
 2:      c : control location;
 3:      (c_1, f, c_2) : meta-transition;
 4:  begin
 5:      Q_R := REACHABLE((C, c_0, M, m_0, Op, T, T̄));
 6:      for each c ∈ F such that values(Q_R, c) ≠ ∅ do
 7:          for each (c_1, f, c_2) ∈ T̄ such that c_1 = c_2 = c do
 8:              if values(Q_R, c) ∩ OPEN-SET((c_1, f, c_2)) ≠ ∅ then
 9:                  return F;
10:      return ?
11:  end.
```

Figure 4.2: Test of emptiness for SMBAs.

$values(Q_R, c) \cap$ OPEN-SET$(\bar{t})$, where $c \in F$ is an accepting control location and $\bar{t} = (c, f, c) \in T'$ is a meta-transition.

A semi-algorithm implementing this semi-decision procedure is given in Figure 4.2. Its output values "**F**" and "**?**" respectively correspond to nonemptiness, and inability to decide between emptiness and nonemptiness. Remark that termination is not guaranteed, since the computation of Q_R performed at Line 5 might not terminate (the issue is discussed in Chapter 5). Of course, the call to REACHABLE can be replaced by a call to any alternate function obtained in Chapter 3. The correctness of the semi-algorithm is established by the following result.

Theorem 4.7 *Let $\mathcal{B} = (C, c_0, M, m_0, Op, T, F)$ be an SMBA, and let $\bar{T}$ be a set of meta-transitions for its underlying SMA (C, c_0, M, m_0, Op, T). If the computation of SMBA-EMPTY?$((C, c_0, M, m_0, Op, T, F), \bar{T})$ returns* **F***, then $\mathcal{B}$ has an accepting run.*

Proof Immediate. □

It remains to show how to obtain the sets OPEN-SET$(\bar{t})$ during the creation of meta-transitions. Specifically, we study cycle (see Section 3.4.1) and multicycle (see Section 3.4.2) meta-transitions. We have the following definition.

Definition 4.8 *Let $\mathcal{A} = (C, c_0, M, m_0, Op, T)$ be an SMA. The sequence of operations $\sigma \in Op^*$ is* iterable *from a memory content $m_1 \in M$ if there exist $m_2, m_3, \ldots \in M$ such that for every $i \in \mathbf{N}_0$, $m_{i+1} = \sigma(m_i)$. By extension, if $\mathcal{C}$ is a cycle in the*

control graph of $\mathcal{A}$ *and* $\bar{t}$ *is the cycle meta-transition associated to* $\mathcal{C}$*, then* $\bar{t}$ *is* iterable *from* $m_1 \in M$ *if the sequence* $body(\mathcal{C})$ *is iterable from* m_1*.*

Intuitively, a cycle meta-transition is iterable from a given memory content if its corresponding cycle can be followed repeatedly an unbounded number of times, starting from that memory content. Clearly, if a cycle meta-transition is iterable from a given memory content, then it is repeatedly open for that content. Indeed, in Definition 4.8, if $m_{i+1} = body(\mathcal{C})(m_i)$ then $(c, m_i) \rightarrow_R^+ (c, m_{i+1})$, where $c \in C$ is the control location at which $\mathcal{C}$ starts and R is the one-step reachability relation of $\mathcal{A}$. The result is then a consequence of Definitions 4.5 and 4.6.

It follows that a way of obtaining the set OPEN-SET($\bar{t}$) associated to a cycle meta-transition $\bar{t}$ consists of computing the set of memory contents from which $\bar{t}$ is iterable. The advantage is that this computation can be performed on the sole basis of the transitions composing the cycle to which $\bar{t}$ is associated, as opposed to the entire transition relation of the SMA.

Practically, we require that the representation system used for sets of memory contents defines a function ITERABLE : $Op^* \rightarrow 2^M$ mapping every sequence of operations onto a representable set of memory contents from which this sequence is known to be iterable. Once again, completeness is not essential, the only requirements being that the sequence must be iterable from every memory content belonging to the returned set, and that this set of memory contents must be representable.

A similar approach can be followed with multicycle meta-transitions. The next definition generalizes the notion of iterability to finite sets of sequences of operations.

Definition 4.9 *Let* $\mathcal{A} = (C, c_0, M, m_0, Op, T)$ *be an SMA. The set of sequences of operations* $\{\sigma_1, \sigma_2, \ldots, \sigma_l\} \in 2^{Op^*}$ $(l \geq 1)$ *is* iterable *from a memory content* $m_1 \in M$ *if there exist* $m_2, m_3, \ldots \in M$ *and* $j_1, j_2, \ldots \in \{1, 2, \ldots, l\}$ *such that for every* $i \in \mathbf{N}_0$*,* $m_{i+1} = \sigma_{j_i}(m_i)$*. By extension, if* $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_l\}$ $(l \geq 1)$ *is a set of cycles in the control graph of* $\mathcal{A}$ *and* $\bar{t}$ *is the multicycle meta-transition associated to this set, then* $\bar{t}$ *is* iterable *from* $m_1 \in M$ *if the set of sequences* $\{body(\mathcal{C}_1), body(\mathcal{C}_2), \ldots, body(\mathcal{C}_l)\}$ *is iterable from* m_1*.*

Intuitively, a multicycle meta-transition is iterable from a given memory content if it is possible to follow repeatedly one of its corresponding cycles (not necessarily the same at each iteration) an unbounded number of times, starting from that memory content. Like in the case of cycle meta-transitions, it is clear that if a multicycle meta-transition is iterable from a given memory content, then it is repeatedly open for that content. The set OPEN-SET($\bar{t}$) associated to a multicycle meta-transition $\bar{t}$ can thus be obtained by computing the set of memory contents from which $\bar{t}$ is iterable.

In practice, we require that the representation system used for sets of memory contents defines a function MULTI-ITERABLE : $2^{Op^*} \to 2^M$ mapping every finite set of sequences of operations onto the representable set of memory contents from which that set of sequences is known to be iterable. The requirements are here that the set of sequences must be iterable from every memory content belonging to the set returned by this function, and that this set of memory contents must be representable. Algorithms implementing the functions ITERABLE and MULTI-ITERABLE for two important memory domains (FIFO channels with send/receive operations, and integers with linear operations) will be given in Chapters 7 and 8.

Chapter 5

Termination

This chapter studies the termination conditions for the state-space exploration semi-algorithms introduced in Chapter 3. It first shows that there does not exist a decidable necessary and sufficient condition for termination, under some mild assumptions on the class of systems being considered. Then, it gives an approximate condition, in the form of a sufficient condition for termination. This condition is presented here independently from any memory domain, the memory operations that one should be able to perform in order to decide this condition being clearly pointed out. Since the sufficient condition can be decided from the syntax of the system, it defines an algorithmically recognizable class of infinite-state systems for which an exact reachability analysis can always be carried out. This chapter next gives additional static conditions under which the LTL model-checking semi-algorithm presented in Chapter 4 becomes a full algorithm (i.e., an exact decision procedure which always terminates). Finally, this chapter describes a technique for optimizing the control graph of a state machine in order to increase the possibility of satisfying the terminating condition, while preserving other properties of interest.

5.1 Undecidability of Termination

In this section, we show that termination of symbolic state-space exploration by the semi-algorithms of Chapter 3 is undecidable for sufficiently expressive classes of infinite-state systems. Let us first consider the semi-algorithms REACHABLE and REACHABLE-D introduced in Section 3.3. We have the following result.

Theorem 5.1 *Let (M, Op) be a Turing-expressive pair of memory domain and operations. The problems which consist of determining, given an arbitrary ESMA $\mathcal{A} = (C, c_0, M, m_0, Op, T, \bar{T})$ using these memory domain and operations, whether the computations of REACHABLE($\mathcal{A}$) and REACHABLE-D($\mathcal{A}$) terminate are both undecidable.*

Proof The proof is by reduction from the halting problem for Turing machines. Since (M, Op) is Turing expressive, it follows from Definition 4.3 that there exists an algorithm for converting any Turing machine $\mathcal{M}$ into an equivalent SMA (C, c_0, M, m_0, Op, T). Let $\bar{T} = \emptyset$, and $\mathcal{A}$ be the ESMA $(C, c_0, M, m_0, Op, T, \bar{T})$. According to Theorem 3.5, the computation of REACHABLE($\mathcal{A}$) terminates if and only if there exists an upper bound on the depth of all the reachable states of $\mathcal{A}$. Since $\mathcal{A}$ has a deterministic transition function and does not have meta-transitions, such an upper bound exists if and only if $\mathcal{A}$ has a finite number of reachable states. This is the case if and only if $\mathcal{M}$ has a finite number of reachable configurations. The problem that consists of determining whether $\mathcal{M}$ has a finite number of reachable configurations is thus reduced to deciding the termination of REACHABLE($\mathcal{A}$). Deciding whether $\mathcal{M}$ halts can now be done as follows:

- If $\mathcal{M}$ has an infinite number of reachable configurations, then it does not halt;
- If $\mathcal{M}$ has a finite number of reachable configurations, then one can simulate its execution up to termination or to the first repetition of a previously reached configuration. $\mathcal{M}$ halts if and only if its execution does not reach the same configuration more than once.

The halting problem for Turing machines is thus reduced to deciding termination of REACHABLE($\mathcal{A}$). Since the former is undecidable [HU79], the latter is undecidable as well. The case of REACHABLE-D is handled in the same way. It is actually sufficient to notice that Theorem 3.5 also holds for REACHABLE-D whenever T is deterministic and $\bar{T}$ is empty. Indeed, in this case, it is easily seen that there will be as many calls to Procedure *explore* as there are reachable states in the state space of $\mathcal{A}$. □

Let us now consider the semi-algorithm REACHABLE-DYNAMIC introduced in Section 3.5. A result similar to Theorem 5.1 can be obtained, the only difference being an additional requirement on the representation system used for sets of memory contents.

Theorem 5.2 *Let (M, Op) be a Turing-expressive pair of memory domain and set of operations. The problem which consists of determining, given an arbitrary SMA $\mathcal{A} = (C, c_0, M, m_0, Op, T)$ using this pair of memory domain and set of operations, whether the computation of REACHABLE-DYNAMIC($\mathcal{A}$) terminates is undecidable, provided that the representation system used for subsets of M allows a procedure for deciding the finiteness of representable sets.*

Proof The proof is by reduction from the halting problem for Turing machines. Since (M, Op) is Turing expressive, it follows from Definition 4.3 that there exists an algorithm for converting any Turing machine $\mathcal{M}$ into an equivalent SMA (C, c_0, M, m_0, Op, T). By hypothesis, there exists a computable predicate FINITE

over the set of subsets of M, such that for every $U \subseteq M$, FINITE(U) is true if and only if U is finite. There are three possible situations:

- *The computation of REACHABLE-DYNAMIC($\mathcal{A}$) terminates, returning the set Q_R, and there exists $c \in C$ such that FINITE($values(Q_R, c)$) = $\mathbf{F}$.* In this case, $\mathcal{A}$ has an infinite number of reachable states, and thus $\mathcal{M}$ has an infinite number of reachable configurations.

- *The computation of REACHABLE-DYNAMIC($\mathcal{A}$) terminates, returning the set Q_R, and there does not exist $c \in C$ such that FINITE($values(Q_R, c)$) = $\mathbf{F}$.* In this case, $\mathcal{A}$ has a finite number of reachable states, and thus $\mathcal{M}$ has a finite number of reachable configurations.

- *The computation of REACHABLE-DYNAMIC($\mathcal{A}$) does not terminate.* In this case, $\mathcal{A}$ has an infinite number of reachable states (otherwise, by Theorem 3.5, the computation of REACHABLE($(C, c_0, M, m_0, Op, T, \bar{T})$) with $\bar{T} = \emptyset$ would terminate, which contradicts, as a consequence of Theorem 3.11, the fact that REACHABLE-DYNAMIC($\mathcal{A}$) does not terminate). Therefore, $\mathcal{M}$ has an infinite number of reachable configurations.

Since there are only a finite number of control locations in C, it is possible to decide which of these situations applies. Determining whether the set of reachable configurations of $\mathcal{M}$ is finite or infinite has thus been reduced to deciding termination of REACHABLE-DYNAMIC for SMAs. The reduction from the halting problem is then identical to the one performed in the proof of Theorem 5.1. □

5.2 Sufficient Conditions

In the previous section, we have shown that there is no decidable sufficient and necessary condition for the termination of state-space exploration. Here, we investigate whether one can obtain *sufficient* static conditions for the termination of the semi-algorithms of Chapter 3. By "static condition", we mean a condition that must be decidable from the syntax of the system without requiring a reachability analysis or any type of state-space search.

The main goal is to obtain a sufficient condition for the termination of REACHABLE (see Section 3.3). We do not address the case of REACHABLE-D for two reasons. First, this algorithm is non-deterministic (the choice of an outgoing transition from an explored state is arbitrary), which makes a termination study difficult. Indeed, two different state-space explorations of the same ESMA by REACHABLE-D are not necessarily equivalent from the point of view of termination. Additional parameters such as the ordering of the outgoing transitions from a control location would thus have to be taken into account by the study. The second reason is that

REACHABLE-D is never better than REACHABLE if termination is the primary concern. Finally, the reason for which the case of REACHABLE-DYNAMIC (see Section 3.5) is not explicitly addressed is that this semi-algorithm is based on a dynamic rather than static computation of meta-transitions. This makes a termination analysis relying on static properties difficult to perform.

It has been established in Section 3.3 that the state-space exploration by the procedure REACHABLE of an ESMA $\mathcal{A}$ terminates if and only if there is an upper bound on the depth of all the reachable states of $\mathcal{A}$, this bound depending only on $\mathcal{A}$ (see Theorem 3.5). In order to get a sufficient termination condition, one must therefore obtain a static condition which guarantees that all the reachable states of $\mathcal{A}$ have a bounded depth. In other words, such a condition would imply that all the reachable states of $\mathcal{A}$ are reachable from the initial state by following a path of transitions and/or meta-transitions whose length is bounded. The solution proposed here consists of designing a condition under which any exploration path can be transformed into an equivalent one of bounded length (recall that two paths or subpaths are said to be *equivalent* if they start from the same state and end in the same state).

Let π be a finite path of transitions and/or meta-transitions from the initial state of $\mathcal{A}$ to a given state (c, m). In general, this path is composed of an alternation of sequences of transitions and sequences of meta-transitions. We call each such sequence a *segment*. Precisely, a *segment* of π is a subpath of π entirely composed either of transitions (in which case it is a *transition segment*) or of meta-transitions (in which case it is a *meta-transition segment*), that is maximal, meaning that it cannot be enlarged by appending or prepending an additional transition or meta-transition. A *subsegment* is a subsequence of consecutive transitions or meta-transitions belonging to the same segment. The methodology we will follow in order to bound the length of exploration paths consists of bounding successively the length of each type of segment, and then bounding the number of segments in a path. We address successively the cases of ESMAs with only cycle and then with only multicycle meta-transitions.

5.3 Machines with Only Cycle Meta-Transitions

5.3.1 Transition Segments

The first step is to bound the length of the transition segments. In other words, given a path from the initial state of $\mathcal{A} = (C, c_0, M, m_0, Op, T)$ to a state (c, m), the goal is to show that there exists an equivalent path such that the length of each of its transition segments is bounded by some value depending only on $\mathcal{A}$. A natural solution is to take advantage of cycle meta-transitions. Roughly speaking, the idea is that for every simple-cyclic subsegment (i.e., beginning and ending at the same

control location, and not visiting twice the same control location) of the path, there should exist a finite sequence of meta-transitions in $\bar{T}$ equivalent to that subsegment. Formally, consider an exploration path of the form

$$\pi = \overbrace{(c_0, m_0) \cdots (c_i, m_i)}^{\pi_1} \xrightarrow{t_1} (c_{i+1}, m_{i+1}) \xrightarrow{t_2} \cdots \xrightarrow{t_j} \overbrace{(c_i, m_{i+j}) \cdots (c, m)}^{\pi_2},$$

where $i \geq 0$, $j \geq 1$, $c_1, c_2, \ldots \in C$, $m_1, m_2, \ldots \in M$, $t_1, t_2, \ldots, t_j \in T$, π_1, π_2 are subpaths, and the cycle labeled by $(t_1; t_2; \ldots; t_j)$ is simple. Since this path contains the simple-cyclic subsegment

$$(c_i, m_i) \xrightarrow{t_1} (c_{i+1}, m_{i+1}) \xrightarrow{t_2} \cdots \xrightarrow{t_j} (c_i, m_{i+j}),$$

there should exist an equivalent path

$$\pi' = \overbrace{(c_0, m_0) \cdots (c_i, m_i)}^{\pi_1} \stackrel{\bar{t}_1}{\Rightarrow} (c'_1, m'_1) \stackrel{\bar{t}_2}{\Rightarrow} (c'_2, m'_2) \stackrel{\bar{t}_3}{\Rightarrow} \cdots \stackrel{\bar{t}_{j'}}{\Rightarrow} \overbrace{(c_i, m_{i+j}) \cdots (c, m)}^{\pi_2},$$

where $j' \geq 1$, $c'_1, c'_2, \ldots, c'_{j'-1} \in C$, $m'_1, m'_2, \ldots, m'_{j'-1} \in M$, $\bar{t}_1, \bar{t}_2, \ldots, \bar{t}_{j'} \in \bar{T}$. A simple static sufficient condition for this requirement is the following.

Condition 1 *For every simple cycle $\mathcal{C}$ in the control graph of $\mathcal{A}$, the cycle meta-transition corresponding to $\mathcal{C}$ must belong to $\bar{T}$.*

If this condition is fulfilled, then it is possible to turn every exploration path into an equivalent one in which all the transition segments are acyclic. Indeed, if a transition segment contains a cycle, then it necessarily contains a simple cycle. By replacing in the path the occurrence of this simple cycle by its corresponding meta-transition, one obtains an equivalent path composed of strictly less transitions and one more meta-transition. Since the original path is finite and has a finite number of transitions, this simple-cycle replacement operation can only be repeated a finite number of times. The final result is a path equivalent to the original one in which all the transition segments are acyclic and hence of length bounded by the number of control locations of $\mathcal{A}$.

5.3.2 Meta-Transition Segments

The second step is to bound the length of the meta-transition segments. There is one small difficulty here: in order to keep the benefit of the first step, one must preserve the acyclic nature of transition segments. The goal is here to make sure that for any exploration path in which all the transition segments are acyclic, there exists an equivalent path in which all the transition segments are acyclic as well, but also in which all the meta-transition segments have a bounded length (the bound depending only on $\mathcal{A}$).

The idea is the following. Since there are only finitely many meta-transitions in $\bar{T}$, it is sufficient to ensure that for every meta-transition segment, there exists an equivalent segment in which each meta-transition appears at most once. This can be done as follows. Since $\bar{T}$ contains only cycle meta-transitions, the control location is constant throughout a meta-transition segment. It is sufficient to require an ordering between all the meta-transitions associated to that control location, such that for any subpath entirely composed of those meta-transitions, there exists an equivalent subpath in which the meta-transitions appear in order. The condition is based on the following definition.

Definition 5.3 *Let $\bar{T}_c = \{t_1, t_2, \ldots, t_k\}$ $(k \geq 0)$ be a set of meta-transitions beginning and ending at the same control location $c \in C$. This set is* serializable *if there exists a permutation $(\bar{t}_{i_1}, \bar{t}_{i_2}, \ldots, \bar{t}_{i_k})$ of $\bar{T}_c$ such that for every set of memory contents $U \subseteq M$ and sequence $\bar{t}_{j_1}; \bar{t}_{j_2}; \ldots; \bar{t}_{j_{k'}}$ of meta-transitions of $\bar{T}_c$, with $k' \geq 0$ and $j_1, j_2, \ldots, j_{k'} \in \{1, 2, \ldots, k\}$, $(\bar{t}_{j_1}; \bar{t}_{j_2}; \ldots; \bar{t}_{j_{k'}})(U) \subseteq (\bar{t}_{i_1}; \bar{t}_{i_2}; \ldots; \bar{t}_{i_k})(U)$. By extension, any meta-transition segment exclusively composed of meta-transitions of a serializable set $\bar{T}_c$ is also said to be serializable.*

We are now ready to state formally the condition.

Condition 2 *For every control location $c \in C$, the set $\bar{T}_c = \{\bar{t}_1, \bar{t}_2, \ldots, \bar{t}_k\}$ $(k \geq 0)$ of all the meta-transitions in $\bar{T}$ beginning and ending in c must be serializable.*

The implementation of this condition by an actual decision procedure will be discussed in Section 5.3.5.

5.3.3 Number of Segments

The last step is to bound the number of segments in exploration paths. This is done in the following way. Since the control graph (C, T) of $\mathcal{A}$ is finite, it contains a finite number of maximal strongly connected components (a strongly connected component is a set of nodes such that any of them is reachable[1] from all of them). By the definition of strongly connected components, an exploration path can visit successively several maximal strongly connected components, but can never get back to a previously visited component. Formally, if

$$\pi = (c_0, m_0) \xrightarrow{t_1} (c_1, m_1) \xrightarrow{t_2} \cdots \xrightarrow{t_j} (c_j, m_j),$$

[1]Recall that a node n_2 of a graph is reachable from a node n_1 is there exists a finite path of edges leading from n_1 to n_2. In the present context, reachability between control locations — which does not depend on transition labels — should not be confused with reachability between states.

where $j \geq 0$, $c_0, c_1, \ldots, c_j \in C$, $m_0, m_1, \ldots, m_j \in M$, $t_1, t_2, \ldots, t_j \in T$, is a path of transitions, then one can break this path into subpaths $\pi_1, \pi_2, \ldots, \pi_k$ $(0 \leq k \leq j)$ satisfying $\pi = \pi_1 \pi_2 \cdots \pi_k$, such that the control locations visited by each π_i all belong to the same maximal strongly connected component of (C, T), and this strongly connected component is different for each π_i. Assume that we first break π into the π_i, and then transform each π_i into an equivalent sequence of transition and meta-transition segments. The total number of segments that would be obtained is at least equal to the number of segments that would be obtained by directly transforming π into a sequence of segments. Since there are only finitely many strongly connected components in (C, T), it is thus sufficient to bound the number of segments in each π_i. We give two ways of obtaining such a bound.

The first one consists of a restriction on the form of the control graph. We simply require that each maximal strongly connected component contains a node visited by every simple cycle belonging to that component. Formally, the condition on every maximal strongly connected components S of (C, T) is as follows.

Condition 3a *The strongly connected component $S \subseteq C$ in the control graph of $\mathcal{A}$ must be such that there exists a control location $c \in S$ such that for every simple cycle $\mathcal{C} = (c_1, \theta_1, c_2), (c_2, \theta_2, c_3), \ldots, (c_k, \theta_k, c_1)$ ($k \geq 1$, $c_1, \ldots, c_k \in C$, $\theta_1, \ldots, \theta_k \in Op$) of S, there exists $i \in \{1, 2, \ldots, k\}$ such that $c_i = c$.*

Let us show that this condition implies a bound on the number of segments composing a subpath entirely contained in a strongly connected component. In order to take into account Conditions 1 and 2, we proceed by starting from an exploration path in which all the transition and meta-transition segments are respectively acyclic and serializable, and we turn this path into an equivalent one that fulfills the same requirements, but whose length is bounded. Precisely, we have the following result.

Theorem 5.4 *Let $\mathcal{A}$ be an ESMA satisfying Conditions 1 and 2, π be an exploration path in which all the transition and meta-transition segments are respectively acyclic and serializable, and π_i be a subpath of π entirely contained in a strongly connected component of (C, T) satisfying Condition 3a. There exists a subpath equivalent[2] to π_i composed of at most two transition segments, which are acyclic, and one meta-transition segment, which is serializable.*

Proof Note that the theorem is trivial if π_i is only composed of transitions. If π_i contains at least one meta-transition, we first transform π_i into π_i' as follows. For each occurrence $(c, m) \stackrel{\bar{t}}{\Rightarrow} (c, m')$ ($c \in C$, $m, m' \in M$, $\bar{t} \in \bar{T}$) of a meta-transition in π_i, we know from Definitions 3.1 and 3.7 that there exist $k \geq 0$ and a simple

[2]Recall that two paths or subpaths are said to be *equivalent* if they start from the same state and end in the same state.

cycle $\mathcal{C}$ in (C, T) such that $m' = \mathit{body}(\mathcal{C})^k(m)$. We then replace the occurrence of $\bar{t}$ in π_i by k successive copies of the sequence of transitions labeling $\mathcal{C}$. Repeatedly performing this operation for all the meta-transitions in π_i, we finally obtain an equivalent subpath π_i' only composed of transitions.

Since Condition 3a is satisfied by hypothesis, there exists $c' \in C$ such that every simple cycle occurring in π_i' visits c'. We can thus split π_i' into three subpaths π_a, π_b and π_c satisfying $\pi_i' = \pi_a\pi_b\pi_c$, such that π_b starts and ends in c' and π_a, π_c either are empty or do not visit c'. The subpaths π_a and π_c are acyclic transition segments. The subpath π_b is a succession of simple cycles. Since Condition 1 is fulfilled, one can replace in π_b every occurrence of a simple cycle by its corresponding meta-transition. The result is a subpath π_b' equivalent to π_b entirely composed of meta-transitions beginning and ending in c'. Since Condition 2 is satisfied, π_b' is a serializable segment of meta-transitions. □

The second way of bounding the number of segments in an exploration path does not involve the control structure, but consists of a restriction on the memory operations. The idea is to require some type of independence between operations carried out by meta-transitions and those performed by transitions. Intuitively, by allowing to reorder the memory operations labeling an exploration path, it will be easier to prove the existence of an equivalent path whose length is bounded. We have the following definition.

Definition 5.5 *Let $\sigma_1, \sigma_2 \in \mathit{Op}^*$ be two finite sequences of operations. The sequence σ_1* precedes favorably *σ_2, which is denoted $\sigma_1 \triangleleft \sigma_2$, if for every set $U \in M$ of memory contents, we have $(\sigma_2; \sigma_1)(U) \subseteq (\sigma_1; \sigma_2)(U)$.*

Intuitively, if $\sigma_1 \triangleleft \sigma_2$, then the sequence $\sigma_1; \sigma_2$ generates at least all the states generated by the sequence $\sigma_2; \sigma_1$. In practice, the procedure for deciding if $\sigma_1 \triangleleft \sigma_2$ has to be implemented together with the representation system for sets of memory contents.

The number of segments contained in a maximal strongly connected component S of (C, T) can now be bounded if the following condition is satisfied.

Condition 3b *The strongly connected component $S \subseteq C$ must be such that for every simple cycle $\mathcal{C} = (c_1, \theta_1, c_2), \ldots, (c_k, \theta_k, c_1)$ ($k \geq 1$, $c_1, \ldots, c_k \in S$, $\theta_1, \ldots, \theta_k \in \mathit{Op}$) and transition $(c, \theta, c') \in T$ which is part of either*

- *a simple cycle $\mathcal{C}' \neq \mathcal{C}$ starting at the location c_1, or*
- *a simple cycle $\mathcal{C}'$ that does not visit the location c_1,*

we have $\mathit{body}(\mathcal{C}) \triangleleft \theta$.

Let us show that this condition implies a bound on the number of segments composing subpaths entirely contained in a strongly connected component. We first need a lemma.

Lemma 5.6 *Let $\mathcal{A}$ be an ESMA, π be an exploration path, and π_i be a subpath of π entirely contained in a strongly connected component of (C,T) satisfying Condition 3b, with π_i of the form*

$$\pi_i = (c_1, m_1) \xrightarrow{t_1} (c_2, m_2) \xrightarrow{t_2} \cdots (c_k, m_k) \xrightarrow{t_k} (c_1, m_{1'}) \overset{body(\mathcal{C})}{\rightarrow} (c_1, m_{1''}),$$

where $k \geq 1$, $c_1, \ldots, c_k \in C$, $m_1, \ldots, m_k, m_{1'}, m_{1''} \in M$, $t_1, t_2, \ldots, t_k \in T$ and $\mathcal{C}$ is a simple cycle of (C,T). The occurrence of $\mathcal{C}$ can be moved to the beginning of π_i, i.e., there exist $m'_1, m'_2, \ldots, m'_k \in M$ such that

$$(c_1, m_1) \overset{body(\mathcal{C})}{\rightarrow} (c_1, m'_1) \xrightarrow{t_1} (c_2, m'_2) \xrightarrow{t_2} \cdots (c_k, m'_k) \xrightarrow{t_k} (c_1, m_{1''})$$

is equivalent to π_i.

Proof For every $j \in \{1, 2, \ldots, k\}$, let θ_j denote the memory operation labeling the transition t_j. Let $\theta = body(\mathcal{C})$.

It is sufficient to show that we have

$$(\theta \cdot \theta_k \cdot \theta_{k-1} \cdots \theta_1)(\{m_1\}) \subseteq (\theta_k \cdot \theta_{k-1} \cdots \theta_1 \cdot \theta)(\{m_1\}),$$

where "$\cdot$" denotes the composition of functions. This would mean that following the sequence $\mathcal{C}; t_1; t_2; \ldots; t_k$ from m_1 would produce at least the memory content that would be obtained by following the sequence $t_1; t_2; \ldots; t_k; \mathcal{C}$ from m_1.

This result is established by proving the existence of functions $\bar{\theta}_1, \bar{\theta}_2, \ldots, \bar{\theta}_k : M \rightarrow M$ such that

$$\begin{array}{rl}
 & (\theta \cdot \theta_k \cdot \theta_{k-1} \cdots \theta_1)(\{m_1\}) \\
\subseteq & (\theta_k \cdot \bar{\theta}_k \cdot \theta_{k-1} \cdots \theta_1)(\{m_1\}) \\
\subseteq & (\theta_k \cdot \theta_{k-1} \cdot \bar{\theta}_{k-1} \cdots \theta_1)(\{m_1\}) \\
 & \vdots \\
\subseteq & (\theta_k \cdot \theta_{k-1} \cdots \bar{\theta}_2 \cdot \theta_1)(\{m_1\}) \\
\subseteq & (\theta_k \cdot \theta_{k-1} \cdots \theta_1 \cdot \bar{\theta}_1)(\{m_1\}) \\
\subseteq & (\theta_k \cdot \theta_{k-1} \cdots \theta_1 \cdot \theta)(\{m_1\}).
\end{array}$$

In other words, the idea consists of moving the leading "θ" function across the sequence. The different $\bar{\theta}_j$ simply express the modifications undergone by θ at each step of the move. For notational convenience, we define $\bar{\theta}_{k+1} = \theta$.

There are various types of t_j in π_i. One type of particular interest are transitions that belong neither to a simple cycle starting at c_1 and different from $\mathcal{C}$, nor to a simple cycle that does not visit c_1 (this choice is motivated by the requirements of Condition 3b). We call such t_j *basic transitions*.

Basic transitions have nice properties. First, all of them are part of $\mathcal{C}$, as a direct consequence of their definition. More interestingly, we have that two successive basic transitions t_{j_1} and t_{j_2} in π_i (i.e., such that there is no basic transition t_j such that $j_1 < j < j_2$) are always consecutive in $\mathcal{C}$, i.e., the control location c_{j_1+1} at which t_{j_1} ends is equal to the control location c_{j_2} at which t_{j_2} begins. This is established by contradiction. Indeed, if t_{j_1} and t_{j_2} are not consecutive in $\mathcal{C}$, then the subsequence σ composed of all the t_j in π_i for which $j_1 < j < j_2$ is a sequence of transitions from c_{j_1+1} to c_{j_2}. Let $\sigma_{\leq j_1}$ denote the subsequence of $\mathcal{C}$ going from c_1 to c_{j_1}, and let $\sigma_{>j_2}$ denote the subsequence of $\mathcal{C}$ going from c_{j_2+1} to c_1. The sequence $(\sigma_{\leq j_1})\sigma(\sigma_{>j_2})$ is therefore a cycle $\mathcal{C}'$ starting at c_1. This cycle contains t_{j_1} and t_{j_2}, but no t_j such that $j_1 < j < j_2$. Therefore, $\mathcal{C}'$ contains an occurrence of a simple cycle different from $\mathcal{C}$ starting at c_1 and containing t_{j_1}. This contradicts the fact that t_{j_1} is a basic transition.

Let $\mathcal{C} = (c_1, \theta'_1, c'_2), (c'_2, \theta'_2, c'_3), \ldots, (c'_l, \theta'_l, c_1)$, with $l \geq 1$, $c'_2, c'_3, \ldots, c'_l \in C$ and $\theta'_1, \theta'_2, \ldots, \theta'_l \in Op$. For each $r \in \{0, 1, \ldots, l-1\}$, let $\mathcal{C}_r$ be the r-th rotation of this cycle, i.e., the cycle

$$\mathcal{C}_r = (c'_{r+1}, \theta'_{r+1}, c'_{r+2}), (c'_{r+2}, \theta'_{r+2}, c'_{r+3}), \ldots, (c'_{r-1}, \theta'_{r-1}, c'_r)(c'_r, \theta'_r, c'_{r+1})$$

(for notational convenience, we define $c'_1 = c'_{l+1} = c_1$). The functions $\bar{\theta}_j$ are computed in decreasing order of j, according to the following rules:

- Each $\bar{\theta}_j$ satisfies $\bar{\theta}_j = \mathit{body}(\mathcal{C}_{r_j})$, where r_j is determined according to the remaining rules;
- If t_j is basic, then $r_j = (r_{j+1} - 1) \bmod l$;
- If t_j is non-basic, then $r_j = r_{j+1}$.

Let us show that those rules are correct. There are two possible situations:

- *If t_j is basic.* Then, since all the basic transitions in π_i are consecutive in $\mathcal{C}$ and the value of $r_{j'}$ is only modified whenever the corresponding $t_{j'}$ is basic, t_j is the transition of $\mathcal{C}$ that leads from c'_{r_j+1} (the control location at which $\mathcal{C}_{r_j}$ starts) to $c'_{r_{j+1}+1}$ (the control location at which $\mathcal{C}_{r_{j+1}}$ starts). As a consequence, we have
$$\theta_j; \mathit{body}(\mathcal{C}_{r_{j+1}}) = \mathit{body}(\mathcal{C}_{r_j}); \theta_j.$$
Therefore, since $\bar{\theta}_{j+1} = \mathit{body}(\mathcal{C}_{r_{j+1}})$ and $\bar{\theta}_j = \mathit{body}(\mathcal{C}_{r_j})$, we have for every $m \in M$
$$(\bar{\theta}_{j+1} \cdot \theta_j)(\{m\}) = (\theta_j \cdot \bar{\theta}_j)(\{m\}).$$

- *If t_j is non-basic.* Then, t_j belongs either to a simple cycle not visiting c_1, or to a simple cycle different from $\mathcal{C}$ starting at c_1. Both cases implies that t_j belongs either to a simple cycle not visiting $c'_{r_{j+1}+1}$ (the control location at which $\mathcal{C}_{r_{j+1}}$ starts), or to a simple cycle different from $\mathcal{C}_{r_{j+1}}$ starting at $c'_{r_{j+1}+1}$. As a consequence, we have $body(\mathcal{C}_{r_{j+1}}) \triangleleft \theta_j$. From Definition 5.5, since $\bar{\theta}_{j+1} = \bar{\theta}_j = body(\mathcal{C}_{r_{j+1}})$, we obtain for every $m \in M$

$$(\bar{\theta}_{j+1} \cdot \theta_j)(\{m\}) \subseteq (\theta_j \cdot \bar{\theta}_j)(\{m\}).$$

□

This lemma has a useful corollary.

Corollary 5.7 *Let $\mathcal{A}$ be an ESMA, π be an exploration path, and π_i be a subpath of π entirely contained in a strongly connected component of (C, T) satisfying Condition 3b, with π_i of the form*

$$\pi_i = (c_1, m_1) \stackrel{\bar{t}}{\Rightarrow} (c_1, m_{1'}) \stackrel{t_1}{\rightarrow} (c_2, m_2) \stackrel{t_2}{\rightarrow} \cdots (c_k, m_k) \stackrel{t_k}{\rightarrow} (c_1, m_{1''}) \stackrel{\bar{t}}{\Rightarrow} (c_1, m_{1'''}),$$

where $k \geq 1$, $c_1, \ldots, c_k \in C$, $m_1, \ldots, m_k, m_{1'}, m_{1''}, m_{1'''} \in M$, $\bar{t} \in \bar{T}$ is a simple-cycle meta-transition, and $t_j \in T$ for every $j \in \{1, 2, \ldots, k\}$. The second occurrence of $\bar{t}$ can be removed from π_i, i.e., there exist $m'_2, \ldots, m'_k, m'_{1'}, m'_{1''} \in M$ such that

$$(c_1, m_1) \stackrel{\bar{t}}{\Rightarrow} (c_1, m'_{1'}) \stackrel{t_1}{\rightarrow} (c_2, m'_2) \stackrel{t_2}{\rightarrow} \cdots (c_k, m'_k) \stackrel{t_k}{\rightarrow} (c_1, m_{1'''})$$

is equivalent to π_i.

Proof Let $\mathcal{C}$ be the simple cycle corresponding to $\bar{t}$. There exists $n \geq 0$ such that $body(\mathcal{C})^n(m_{1''}) = m_{1'''}$. Thus, there exists a subpath equivalent to π_i of the form

$$(c_1, m_1) \stackrel{\bar{t}}{\Rightarrow} (c_1, m_{1'}) \stackrel{t_1}{\rightarrow} (c_2, m_2) \stackrel{t_2}{\rightarrow} \cdots (c_k, m_k) \stackrel{t_k}{\rightarrow} (c_1, m_{1''}) \stackrel{body(\mathcal{C})^n}{\rightarrow} (c_1, m_{1'''}).$$

Applying Lemma 5.6 n times to this subpath, we obtain that there exist $m'_2, \ldots, m'_k$, $m'_{1'} \in M$ such that

$$(c_1, m_1) \stackrel{\bar{t}}{\Rightarrow} (c_1, m_{1'}) \stackrel{body(\mathcal{C})^n}{\rightarrow} (c_1, m'_{1'}) \stackrel{t_1}{\rightarrow} (c_2, m'_2) \stackrel{t_2}{\rightarrow} \cdots (c_k, m'_k) \stackrel{t_k}{\rightarrow} (c_1, m_{1'''})$$

is equivalent to π_i, which implies that

$$(c_1, m_1) \stackrel{\bar{t}}{\Rightarrow} (c_1, m'_{1'}) \stackrel{t_1}{\rightarrow} (c_2, m'_2) \stackrel{t_2}{\rightarrow} \cdots (c_k, m'_k) \stackrel{t_k}{\rightarrow} (c_1, m_{1'''})$$

is equivalent to π_i as well. □

We are now ready to show that Condition 3b implies a bound on the number of segments composing a subpath entirely contained in a strongly connected component.

Theorem 5.8 *Let $\mathcal{A}$ be an ESMA satisfying Conditions 1 and 2, π be an exploration path in which all the transition and meta-transition segments are respectively acyclic and serializable, and π_i be a subpath of π entirely contained in a maximal strongly connected component of (C,T) satisfying Condition 3b (let S denote this strongly connected component). There exists a subpath equivalent to π_i composed of at most $N+1$ transition segments, which are acyclic, and N meta-transition segments, which are serializable, where N denotes the number of meta-transitions in $\bar{T}$ that begin and end in control locations belonging to S.*

Proof We can assume that all the meta-transitions that appear in π_i correspond to simple cycles. Indeed, if this is not the case, then π_i can easily be turned into an equivalent subpath in which every occurrence of a meta-transition corresponding to a non-simple cycle has been replaced by a sequence of acyclic transition segments and of simple-cycle meta-transitions. Once this operation is performed, we simply prove that multiple occurrences of the same simple-cycle meta-transition can be removed from π_i. Indeed, suppose that π_i is of the form

$$\cdots \overbrace{(c_1,m_1) \stackrel{\bar{t}}{\Rightarrow} (c_1,m_{1'}) \stackrel{\tilde{t}_1}{\rightharpoonup} (c_2,m_2) \stackrel{\tilde{t}_2}{\rightharpoonup} \cdots (c_k,m_k) \stackrel{\tilde{t}_k}{\rightharpoonup} (c_1,m_{1''}) \stackrel{\bar{t}}{\Rightarrow} (c_1,m_{1'''})}^{\pi'_i} \cdots,$$

where $k \geq 1$, $c_1,\ldots,c_k \in C$, $m_1,\ldots,m_k,m_{1'},m_{1''},m_{1'''} \in M$, each $\tilde{t}_j$ is either a transition (in which case "$\rightharpoonup$" should be read as "$\rightarrow$") or a meta-transition (in which case "$\rightharpoonup$" should be read as "$\Rightarrow$"), and π'_i is the subpath of π_i beginning and ending with two successive occurrences in π_i of the meta-transition $\bar{t} \in \bar{T}$.

Let us show that the second occurrence of $\bar{t}$ can be removed from π'_i, i.e., that there exist $m'_2,\ldots,m'_k,m'_{1'} \in M$ such that

$$(c_1,m_1) \stackrel{\bar{t}}{\Rightarrow} (c_1,m'_{1'}) \stackrel{\tilde{t}_1}{\rightharpoonup} (c_2,m'_2) \stackrel{\tilde{t}_2}{\rightharpoonup} \cdots (c_k,m'_k) \stackrel{\tilde{t}_k}{\rightharpoonup} (c_1,m_{1'''}).$$

Remark that for each $\tilde{t}_j$ that is a meta-transition, its occurrence in π'_i can be replaced by an equivalent sequence of transitions. Indeed, if we have

$$(c_j,m'_j) \stackrel{\tilde{t}_j}{\Rightarrow} (c_{j+1},m'_{j+1}),$$

then there exists $n \geq 0$ such that

$$(c_j,m'_j) \stackrel{(\sigma_j)^n}{\rightarrow} (c_{j+1},m'_{j+1}),$$

where σ_j is the sequence of transitions labeling the cycle corresponding to $\tilde{t}_j$. After all the substitutions have been made, each $\tilde{t}_j$ is a transition, and the fact that the second occurrence of $\bar{t}$ can be removed is a direct consequence of Corollary 5.7.

By suppressing iteratively redundant meta-transitions from π_i, one thus finally obtains an equivalent subpath containing at most one occurrence of each meta-transition in $\bar{T}$. This subpath is thus composed of at most N meta-transition segments and $N+1$ transition segments.

The serializable character of meta-transition segments is not affected by the removal of meta-transitions. It remains to show that one can always obtain a suitable subpath in which the transition sequences are acyclic. This is done as follows. If π_i contains a cyclic subsequence of transitions, then it is of the form

$$\cdots (c_1, m_1) \xrightarrow{t_1} (c_2, m_2) \xrightarrow{t_2} \cdots (c_k, m_k) \xrightarrow{t_k} (c_1, m_{1'}) \cdots,$$

where $k \geq 1$, $c_1, \ldots, c_k \in C$, $m_1, \ldots, m_k, m_{1'} \in M$, $t_1, \ldots, t_k \in T$, and $t_1 t_2 \cdots t_k$ is a simple cycle (let $\mathcal{C}$ be this cycle). We can replace the occurrence of $\mathcal{C}$ in π_i by

$$(c_1, m_1) \stackrel{\bar{t}}{\Rightarrow} (c_1, m_{1'}),$$

where $\bar{t} \in \bar{T}$ is the cycle meta-transition corresponding to $\mathcal{C}$.

If one performs alternatively the two operations discussed in this proof (removing in π_i all the redundant meta-transitions, and replacing a cyclic subsequence of transitions by a meta-transition) then the final result will be a subpath composed of at most N meta-transition segments and $N+1$ acyclic transition segments. This subpath is always obtained after a finite number of steps. Indeed, the number of transitions in π_i is not affected by the former operation, and is strictly decreased by the latter. □

5.3.4 Summary of Conditions

Let us summarize the necessary conditions obtained in Sections 5.3.1, 5.3.2 and 5.3.3.

Definition 5.9 *An ESMA with only cycle meta-transitions is* safe *if it satisfies Conditions 1 and 2, and if each maximal strongly connected component of its control graph satisfies either Condition 3a or Condition 3b (the satisfied condition may differ for each strongly connected component).*

We are now ready to state the main result of this section.

Theorem 5.10 *Let $\mathcal{A}$ be a safe ESMA with only cycle meta-transitions. The computation of REACHABLE(A) terminates.*

Proof The elements of the proof have already been developed during the discussion of each condition. Given an exploration path π, one first transforms its transition segments into acyclic ones, then serializes its meta-transition segments, and finally reduces the total number of segments. The result π' is an exploration path equivalent to π, for which:

- The length of each transition segment is less than the number of control states in C;

- The length of each meta-transition segment is less than the number of meta-transitions in $\bar{T}$;
- The number of segments is less than three times the number of maximal strongly connected components in the control graph (C, T).

Each reachable state of $\mathcal{A}$ can thus be reached by an exploration path whose length is bounded by a value depending only on $\mathcal{A}$. According to Theorem 3.5, the computation of REACHABLE(A) terminates. □

5.3.5 Implementation

Let us now study how to check algorithmically that the conditions discussed in Sections 5.3.1, 5.3.2 and 5.3.3 are satisfied by an ESMA $\mathcal{A}$.

Deciding whether Condition 1 is satisfied can be done by testing the inclusion of the set of cycles returned by SIMPLE-CYCLES (see Section 3.4.1) into the set of cycles from which the meta-transitions of $\mathcal{A}$ are created. If the goal is to ensure that Condition 1 is satisfied while creating meta-transitions, then one can simply call SIMPLE-CYCLES and then create one meta-transition for each returned cycle. If there are cycles for which no meta-transition can be created, then Condition 1 cannot possibly be satisfied.

Making sure that Condition 2 is satisfied is tougher. The problem consists of checking whether a finite set of cycle meta-transition is serializable. We use the following sufficient criterion.

Definition 5.11 *Let $c \in C$ be a control location and $\bar{T}_c = \{\bar{t}_1, \bar{t}_2, \ldots, \bar{t}_k\}$ $(k \geq 0)$ be a set of cycle meta-transitions beginning and ending in c. For each $i \in \{1, 2, \ldots, k\}$, let $\mathcal{C}_i$ be the cycle corresponding to $\bar{t}_i$. The set $\bar{T}_c$ is* strongly serializable *if there exists a permutation $\{i_1, i_2, \ldots, i_k\}$ of $\{1, 2, \ldots, k\}$ such that for any j, j' for which $1 \leq j < j' \leq k$, we have $body(\mathcal{C}_{i_j}) \triangleleft body(\mathcal{C}_{i_{j'}})$.*

The correctness of this sufficient condition is established by the following result.

Theorem 5.12 *Let $c \in C$ be a control location and $\bar{T}_c = \{\bar{t}_1, \bar{t}_2, \ldots, \bar{t}_k\}$ $(k \geq 0)$ be a set of cycle meta-transitions beginning and ending in c. If $\bar{T}_c$ is strongly serializable, then it is serializable.*

Proof Let $\sigma = \bar{t}_{j_1}; \bar{t}_{j_2}; \ldots; \bar{t}_{j_{k'}}$ be a sequence of meta-transitions of $\bar{T}_c$, with $k' \geq 0$ and $j_1, j_2, \ldots, j_{k'} \in \{1, 2, \ldots, k\}$. For every $i \in \{1, 2, \ldots, k'-1\}$ such that the cycles $\mathcal{C}_{j_i}$ and $\mathcal{C}_{j_{i+1}}$ corresponding respectively to $\bar{t}_{j_i}$ and $\bar{t}_{j_{i+1}}$ satisfy $body(\mathcal{C}_{j_{i+1}}) \triangleleft body(\mathcal{C}_{j_i})$, we have for every set of memory contents $U \subseteq M$

$$(\bar{t}_{j_1}; \ldots; \bar{t}_{j_{i-1}}; \bar{t}_{j_i}; \bar{t}_{j_{i+1}}; \ldots; \bar{t}_{j_{k'}})(U) \subseteq (\bar{t}_{j_1}; \ldots; \bar{t}_{j_{i-1}}; \bar{t}_{j_{i+1}}; \bar{t}_{j_i}; \bar{t}_{j_{i+2}}; \ldots; \bar{t}_{j_{k'}})(U).$$

If $\bar{T}_c$ is strongly serializable, then there exists a permutation $P = \{i_1, i_2, \ldots, i_k\}$ of $\{1, 2, \ldots, k\}$ such that for any j, j' for which $1 \leq j < j' \leq k$, we have $\mathit{body}(\mathcal{C}_{i_j}) \lhd \mathit{body}(\mathcal{C}_{i_{j'}})$.

By repeatedly permuting successive $\bar{t}_{j_i}$ and $\bar{t}_{j_{i+1}}$ in σ for which j_{i+1} precedes j_i in P, and collapsing successive $\bar{t}_{j_i}$ and $\bar{t}_{j_{i+1}}$ such that $j_i = j_{i+1}$ (since, in that case, we have $(\bar{t}_{j_i}; \bar{t}_{j_{i+1}})(U) = \bar{t}_{j_i}(U)$ for every $U \subseteq M$), one obtains in a finite number of steps that for every set of memory contents $U \subseteq M$

$$(\bar{t}_{j_1}; \bar{t}_{j_2}; \ldots; \bar{t}_{j_{k'}})(U) \subseteq (\bar{t}_{i_1}; \bar{t}_{i_2}; \ldots; \bar{t}_{i_k})(U).$$

The set $\bar{T}_c$ is thus serializable. □

The advantage of strong over plain serializability is that it is much easier to be checked algorithmically. There is however one small difficulty: the relation "$\lhd$" is not transitive and hence does not correspond to an order relation. Thus, checking whether a set $\bar{T}$ of cycle meta-transitions is strongly serializable does not reduce to simply sorting the set.

We solve the problem in the following way. First, if $\bar{T}$ contains two meta-transitions such that their corresponding cycles $\mathcal{C}$ and $\mathcal{C}'$ are labeled by sequences of operations which cannot be compared by "$\lhd$" (in other words, if $\mathit{body}(\mathcal{C}) \not\lhd \mathit{body}(\mathcal{C}')$ and $\mathit{body}(\mathcal{C}') \not\lhd \mathit{body}(\mathcal{C})$), then $\bar{T}$ is not serializable. Suppose now that this is not the case. We build a graph whose nodes are associated to the meta-transitions in $\bar{T}$ and whose edges correspond to the relation "$\lhd$" between the sequences of operations labeling the cycles corresponding to those meta-transitions. After this graph has been build, we remove all its reciprocal edges, i.e., we remove all the edges linking two nodes such that their corresponding meta-transitions are associated to cycles $\mathcal{C}$ and $\mathcal{C}'$ for which $\mathit{body}(\mathcal{C}) \lhd \mathit{body}(\mathcal{C}')$ and $\mathit{body}(\mathcal{C}') \lhd \mathit{body}(\mathcal{C})$.

After this operation has been performed, we test whether the resulting graph is acyclic. If the graph contains a cycle, then $\bar{T}$ is not serializable. Indeed, regardless of the order according to which the meta-transitions in $\bar{T}$ are considered, the presence of a cycle implies that Definition 5.11 is violated. On the other hand, if the graph is acyclic, then it expresses a partial order between the meta-transitions in $\bar{T}$. Any total order between those meta-transitions consistent with that partial order is such that $\mathit{body}(\mathcal{C}) \lhd \mathit{body}(\mathcal{C}')$ for every cycles $\mathcal{C}$ and $\mathcal{C}'$ such that the meta-transition corresponding to $\mathcal{C}$ precedes the one corresponding to $\mathcal{C}'$ in that order. Therefore, $\bar{T}$ is serializable.

The total cost of building the graph and then testing whether it is acyclic is $O(N_\lhd |\bar{T}|)$, where $|\bar{T}|$ is the number of meta-transitions in $\bar{T}$, and $N_\lhd$ is the cost of comparing two sequences of operations with respect to "$\lhd$".

Let us now discuss the case of Condition 3a. Deciding whether it is satisfied for a given maximal strongly connected component of the control graph can be done by simply computing the intersection of the sets of control locations visited by all

the simple cycles of the component. Indeed, we have that Condition 3a is satisfied if and only if this intersection is not empty.

Finally, a decision procedure for Condition 3b can straightforwardly be derived from its definition. All one has to do is to test successively all the pairs $(\mathcal{C}, \theta)$ composed of a simple cycle $\mathcal{C}$ and a memory operation θ belonging to the strongly connected component being checked.

5.4 Machines with Only Multicycle Meta-Transitions

In this section, we investigate whether the results of Section 5.3 can be adapted to SMAs associated with only multicycle meta-transitions. We follow the same methodology, which consists of bounding successively the lengths of the transition and meta-transition segments composing exploration paths, and then the number of those segments.

5.4.1 Transition Segments

Roughly speaking, Condition 1 can be adapted to the case of an ESMA with only multicycle meta-transitions $\mathcal{A}$ by simply requiring that for every simple cycle $\mathcal{C}$ in the control graph of $\mathcal{A}$, there is a meta-transition corresponding to at least this cycle. Precisely, the condition is as follows.

Condition 1' *For every simple cycle $\mathcal{C}$ in the control graph of $\mathcal{A}$, there must exist a multicycle meta-transition $\bar{t} \in \bar{T}$ such that its set of corresponding cycles $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$ $(k > 0)$ contains $\mathcal{C}$.*

This condition implies that every exploration path can be translated into an equivalent one in which all the transition segments are acyclic. Indeed, if a transition segment contains an occurrence of a simple cycle $\mathcal{C}$, then this occurrence can be replaced by any multicycle meta-transition associated to a set of cycles containing $\mathcal{C}$. This operation can be repeated until all the transition segments composing the path are acyclic.

5.4.2 Meta-Transition Segments

Let us now study the case of Condition 2. Instead of requiring that each meta-transition segment in an exploration path is reducible to a segment containing at most one occurrence of each meta-transition, we now go further and impose that for each meta-transition segment, there exists a single multicycle meta-transition equivalent to that segment. Precisely, the condition is the following.

Condition 2' *For every control location $c \in C$ visited by at least one cycle in the control graph of $\mathcal{A}$, there must exist a multicycle meta-transition $\bar{t}_c \in \bar{T}$ such that its corresponding set of cycles $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$ $(k > 0)$ contains at least all the simple cycles starting at c.*

This condition makes it possible to reduce the length of any meta-transition segment to one. Indeed, one can simply replace the segment by one occurrence of any meta-transition whose associated set of cycles contains all the cycles to which the meta-transitions of the segment correspond.

It is worth mentioning that Condition 2' implies Condition 1'. From an algorithmic perspective, a simple way of ensuring that Condition 2' is satisfied is to impose that the function MULTI-META-SET (introduced in Section 3.4.2) returns at least all the meta-transitions required by the condition.

5.4.3 Number of Segments

We now address the problem of bounding the number of segments composing exploration paths. Interestingly enough, Conditions 3a and 3b are applicable in this context without any modification. We have the following results.

Theorem 5.13 *Let $\mathcal{A} = (C, c_0, M, m_0, Op, T, \bar{T})$ be an ESMA satisfying Condition 2', π be an exploration path whose transition and meta-transition segments are respectively acyclic and of length one, and π_i be a subpath of π entirely contained in a strongly connected component of (C, T) satisfying Condition 3a. It is possible to transform π_i into an equivalent subpath composed of at most two transition segments, which are acyclic, and one meta-transition segment, which is of length one.*

Proof The proof is very similar to the one of Theorem 5.4. First, the theorem trivially holds if π_i is only composed of transitions. If π_i contains at least one meta-transition, then we first transform π_i by replacing every occurrence of a meta-transition by the corresponding sequence of transitions. Let π'_i be the subpath obtained after having replaced all the meta-transitions in π_i.

Since Condition 3a is satisfied by hypothesis, there exists $c' \in C$ such that every simple cycle occurring in π'_i visits c'. We can thus split π'_i into three subpaths π_a, π_b and π_c such that $\pi'_i = \pi_a\pi_b\pi_c$, π_b starts and ends in c', and π_a and π_c either are empty or do not visit c'. The subpaths π_a and π_c are acyclic transition segments. The subpath π_b is a sequence of simple cycles starting and ending in c'. Since Condition 2' is fulfilled, one can replace this subpath by a single occurrence of the appropriate multicycle meta-transition. □

Theorem 5.14 *Let $\mathcal{A} = (C, c_0, M, m_0, Op, T, \bar{T})$ be an ESMA with only multicycle meta-transitions satisfying Condition 2', π be an exploration path whose transition*

and meta-transition segments are respectively acyclic and of length one, and π_i be a subpath of π entirely contained in a strongly connected component of (C, T) satisfying Condition 3b (let S denote this strongly connected component). It is possible to transform π_i into an equivalent subpath composed of at most $N + 1$ transition segments and N meta-transition segments, where N denotes the number of meta-transitions in $\bar{T}$ beginning and ending in control locations belonging to S.

Proof The proof is similar to the one of Theorem 5.8. First, we assume that all the meta-transitions that occur in π_i can be replaced by meta-transitions associated to sets of simple cycles (as opposed to sets of arbitrary cycles). Precisely, we assume that for every subpath of π_i of the form

$$(c, m) \stackrel{\bar{t}}{\Rightarrow} (c, m'),$$

where $c \in C$, $m, m' \in M$ and $\bar{t} \in \bar{T}$, there exist a finite number $k > 0$ of non necessarily distinct simple cycles $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$ in (C, T) such that

$$(\mathit{body}(\mathcal{C}_1); \mathit{body}(\mathcal{C}_2); \ldots; \mathit{body}(\mathcal{C}_k))(m) = m'.$$

This assumption can be made without loss of generality, since any subpath π_i can be turned into an equivalent one in which every occurrence of a meta-transition that can not be replaced by a meta-transition associated to a set of simple cycles has been replaced by a sequence of acyclic transitions and of other meta-transitions. After this operation has been performed, we simply prove that multiple occurrences of the same meta-transition can be deleted from π_i. Indeed, suppose that this subpath is of the form

$$\cdots \overbrace{(c_1, m_1) \stackrel{\bar{t}}{\Rightarrow} (c_1, m_{1'}) \stackrel{\tilde{t}_1}{\rightharpoonup} (c_2, m_2) \stackrel{\tilde{t}_2}{\rightharpoonup} \cdots (c_k, m_k) \stackrel{\tilde{t}_k}{\rightharpoonup} (c_1, m_{1''}) \stackrel{\bar{t}}{\Rightarrow} (c_1, m_{1'''})}^{\pi'_i} \cdots,$$

where $k \geq 1$, $c_1, \ldots, c_k \in C$, $m_1, \ldots, m_k, m_{1'}, m_{1''}, m_{1'''} \in M$, each $\tilde{t}_j$ is either a transition (in which case "$\rightharpoonup$" should be read as "$\rightarrow$") or a meta-transition (in which case "$\rightharpoonup$" should be read as "$\Rightarrow$"), and π'_i is the subpath of π_i beginning and ending with two successive occurrences in π_i of the meta-transition $\bar{t} \in \bar{T}$.

Let us show that the second occurrence of $\bar{t}$ can be deleted from π'_i. For the same reasons as in the proof of Theorem 5.8, we assume that each $\tilde{t}_j$ is a transition.

We know that $\bar{t}$ is a meta-transition associated to a set of simple cycles $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$ $(k > 0)$. Let $\bar{t}_1, \bar{t}_2, \ldots, \bar{t}_k$ be the cycle meta-transitions associated to those simple cycles[3]. From Definition 3.9, we have that there exist $l \geq 0$, $i_1, i_2, \ldots, i_l \in \{1, 2, \ldots, k\}$ and $m'_1, m'_2, \ldots, m'_{l-1} \in M$ such that

$$(c_1, m_{1''}) \stackrel{\bar{t}_{i_1}}{\Rightarrow} (c_1, m'_1) \stackrel{\bar{t}_{i_2}}{\Rightarrow} (c_1, m'_2) \cdots (c_1, m'_{l-2}) \stackrel{\bar{t}_{i_{l-1}}}{\Rightarrow} (c_1, m'_{l-1}) \stackrel{\bar{t}_{i_l}}{\Rightarrow} (c_1, m_{1'''}).$$

[3]The meta-transitions $\bar{t}_1, \bar{t}_2, \ldots, \bar{t}_k$ are not required to belong to $\bar{T}$, and may even be uncomputable with respect to the representation system used for sets of memory contents. They are only introduced for the purpose of the proof.

We can assume that each meta-transition $\bar{t}_j$ in $\{\bar{t}_1, \bar{t}_2, \ldots, \bar{t}_k\}$ appears at least once in this subpath from $(c_1, m_{1''})$ to $(c_1, m_{1'''})$. Indeed, since those are cycle meta-transitions, we have for every $m \in M$

$$(c_1, m) \stackrel{\bar{t}_j}{\Rightarrow} (c_1, m),$$

and therefore meta-transitions can be appended at will to the subpath.

We have just established that there is a subpath equivalent to π'_i that is of the form

$$(c_1, m_1) \stackrel{\bar{t}}{\Rightarrow} (c_1, m_{1'}) \stackrel{\tilde{t}_1}{\rightarrow} (c_2, m_2) \stackrel{\tilde{t}_2}{\rightarrow} \cdots (c_k, m_k) \stackrel{\tilde{t}_k}{\rightarrow} (c_1, m_{1''}) \stackrel{\bar{t}_{i_1}}{\Rightarrow} (c_1, m'_1) \stackrel{\bar{t}_{i_2}}{\Rightarrow} (c_1, m'_2)$$
$$\cdots (c_1, m'_{l-2}) \stackrel{\bar{t}_{i_{l-1}}}{\Rightarrow} (c_1, m'_{l-1}) \stackrel{\bar{t}_{i_l}}{\Rightarrow} (c_1, m_{1'''}).$$

We now remove successively each $\bar{t}_{i_j}$ from this subpath, for $j = 1, 2, \ldots l$, by performing the following operations:

1. Let $m \in M$ be the memory content reached prior to the occurrence of $\tilde{t}_1$ in the subpath. Before $\tilde{t}_1$, we insert into the subpath a dummy occurrence of $\bar{t}_{i_j}$:

$$(c_1, m) \stackrel{\bar{t}_{i_j}}{\Rightarrow} (c_1, m).$$

2. There is now in π'_i a subpath of the form

$$(c_1, m) \stackrel{\bar{t}_{i_j}}{\Rightarrow} (c_1, m) \stackrel{\tilde{t}_1}{\rightarrow} (c_2, m''_2) \stackrel{\tilde{t}_2}{\rightarrow} \cdots (c_k, m''_k) \stackrel{\tilde{t}_k}{\rightarrow} (c_1, m''_{k+1}) \stackrel{\bar{t}_{i_j}}{\Rightarrow} (c_1, m''_{k+2}),$$

with $m''_2, m''_3, \ldots, m''_{k+2} \in M$. Applying Corollary 5.7, we remove the second occurrence of $\bar{t}_{i_j}$ from this subpath.

One eventually obtains a subpath equivalent to π'_i in which all the occurrences of the meta-transition $\bar{t}_{i_j}$ appear after the initial $\bar{t}$ and before $\tilde{t}_1$. The initial segment of meta-transitions can be replaced by a single occurrence of $\bar{t}$, yielding a subpath equivalent to π'_i of the form

$$(c_1, m_1) \stackrel{\bar{t}}{\Rightarrow} (c_1, m'_{1'}) \stackrel{\tilde{t}_1}{\rightarrow} (c_2, m'''_2) \stackrel{\tilde{t}_2}{\rightarrow} \cdots (c_k, m'''_k) \stackrel{\tilde{t}_k}{\rightarrow} (c_1, m_{1'''}),$$

where $m'''_2, m'''_3, \ldots, m'''_k, m'_{1'} \in M$.

By suppressing iteratively redundant meta-transitions from π_i, one thus finally obtains an equivalent subpath containing at most one occurrence of each meta-transition in $\bar{T}$. The subpath is then composed of at most N meta-transition segments and $N + 1$ transition segments.

It remains to show that one can always obtain such a subpath in which all the transition segments are acyclic and all the meta-transition segments are of length one. The idea is similar to the corresponding part of the proof of Theorem 5.8. One

replaces every subpath of transitions forming a simple cycle by an occurrence of a meta-transition associated to a set containing this cycle. In addition, every meta-transition segment of length greater than one can be replaced by an occurrence of any meta-transition associated to a superset of the sets of cycles associated with the meta-transitions of the segment. Alternating those operations, and removing redundant meta-transitions after each step, one finally obtains a subpath composed of at most N meta-transition segments of length one and $N + 1$ acyclic transition segments. □

5.4.4 Summary of Conditions

Let us summarize the necessary conditions obtained in Sections 5.4.1, 5.4.2 and 5.4.3.

Definition 5.15 *An ESMA with only multicycle meta-transitions is* safe *if it satisfies Condition 2', and if each maximal strongly connected component of its control graph satisfies either Condition 3a or Condition 3b (the satisfied condition may differ for each strongly connected component).*

We are now ready to state the main result of this section.

Theorem 5.16 *Let $\mathcal{A} = (C, c_0, M, m_0, Op, T, \bar{T})$ be a safe ESMA with only multicycle meta-transitions. The computation of REACHABLE(A) terminates.*

Proof The elements of the proof have already been developed during the discussion of each condition. Given an exploration path π, one first transforms its transition segments into acyclic ones, then replaces its meta-transition segments by single meta-transitions, and finally reduces the total number of segments. The result π' is an exploration path equivalent to π, for which:

- The length of each transition segment is less than the total number of control states in C;
- The length of each meta-transition segment is one;
- The number of segments is less than three times the number of maximal strongly connected components in the control graph (C, T).

Each reachable state of $\mathcal{A}$ can thus be reached by an exploration path whose length is bounded by a value depending only on $\mathcal{A}$. According to Theorem 3.5, the computation of REACHABLE(A) terminates. □

5.5 LTL Model Checking

In the previous sections, we proposed sufficient conditions for termination of state-space exploration by the semi-algorithms introduced in Chapter 3. As a corollary, those conditions also guarantee the termination of semi-algorithms relying upon state-space exploration, such as the LTL model-checking semi-algorithm proposed in Section 4.4. Unfortunately, this does not imply that LTL model-checking is decidable for the class of ESMAs satisfying the sufficient termination conditions, since the semi-decision procedure introduced in Section 4.4 may sometimes terminate with a "don't know" answer.

Here, we go further and show that sufficient conditions can be obtained for a full decision procedure. Precisely, since LTL model checking relies on a test of emptiness for Structured-Memory Büchi Automata, we will give sufficient static conditions on SMBAs that guarantee that the emptiness problem can be decided. In practice, since the SMBAs which are tested for emptiness are constructed by computing the product of a system (modeled as an SMA) and of a property (modeled as a Büchi automaton), this means that the sufficient conditions will be evaluated over pairs of the forms (system, property).

The methodology we follow in order to obtain the conditions consists of first considering SMBAs such that their underlying SMA satisfies the terminating conditions of Sections 5.3 and 5.4 (after having been associated with a set of meta-transitions), and then examining whether additional restrictions need to be imposed.

5.5.1 Systems with Only Cycle Meta-Transitions

We first consider the case of an SMBA $\mathcal{B} = (C, c_0, M, m_0, Op, T, F)$ associated with a finite set of cycle meta-transitions $\bar{T}$ such that the ESMA $(C, c_0, M, m_0, Op, T, \bar{T})$ is safe. For any accepting run of $\mathcal{B}$, there exists an accepting control location $c \in F$ visited infinitely often by the path π of transitions corresponding to the run. There are two possible situations, depending on the maximal strongly connected component S of (C, T) to which c belongs.

The first situation is when S satisfies Condition 3a. In this case, we know that there exists $c' \in S$ such that every simple cycle contained in S visits c'. As a consequence, we have that π is of the form

$$(c_0, m_0) \xrightarrow{\sigma} (c', m_1) \overset{body(\mathcal{C}_1)}{\rightarrow} (c', m_2) \overset{body(\mathcal{C}_2)}{\rightarrow} (c', m_3) \cdots,$$

where σ is an acyclic sequence of transitions of T, $m_1, m_2, \ldots \in M$, and $\mathcal{C}_1, \mathcal{C}_2, \ldots$ are simple cycles of (C, T). Since there are only finitely many such cycles, there must exist a $\mathcal{C}_j$ that visits an accepting control location, and that occurs an infinite number of times in the development of π. The question is to know whether this $\mathcal{C}_j$ can be repeatedly followed an unbounded number of times from a memory content

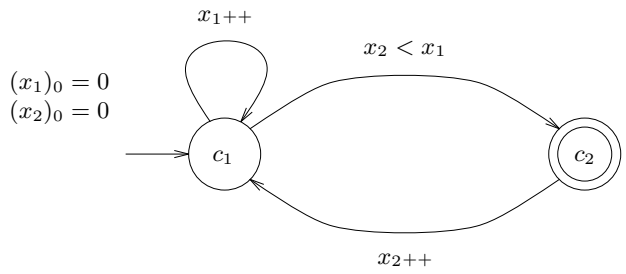

Figure 5.1: SMBA accepting a nonempty language.

reachable at c' (in other words, whether there exists an accepting run that will be discovered by the semi-algorithm SMBA-EMPTY? of Figure 4.2).

The answer to that question is unfortunately negative, as it is illustrated by the SMBA depicted in Figure 5.1. Indeed, let $\mathcal{C}_1 = (c_1, x_{1{+}{+}}, c_1)$ and $\mathcal{C}_2 = (c_1, x_2 < x_1, c_2); (c_2, x_{2{+}{+}}; c_1)$ in that figure. The SMBA admits the accepting run

$$(c_1,(0,0)),(c_1,(1,0)),(c_2,(1,0)),(c_1,(1,1)),(c_1,(2,1)),(c_2,(2,1)),(c_1,(2,2)),\ldots$$

corresponding to the path

$$(c_1,(0,0)) \xrightarrow{\mathcal{C}_1} (c_1,(1,0)) \xrightarrow{\mathcal{C}_2} (c_1,(1,1)) \xrightarrow{\mathcal{C}_1} (c_1,(2,1)) \xrightarrow{\mathcal{C}_2} (c_1,(2,2)) \xrightarrow{\mathcal{C}_1} (c_1,(3,2))\cdots,$$

in which $\mathcal{C}_2$ appears an infinite number of times. However, there is no memory content $(v_1, v_2) \in \mathbf{Z}^2$ from which $\mathcal{C}_2$ can be followed repeatedly an unbounded number of times.

We must therefore impose an additional restriction. A simple solution consists of strengthening the condition on the meta-transitions beginning and ending in the control location visited by all the simple cycles. Instead of requiring serializability between the sequences of operations labeling those cycles, we impose that each such sequence precedes favorably all of them. Formally, the condition is the following.

Condition 3a" *The strongly connected component $S \subseteq C$ in the control graph of $\mathcal{A}$ must be such that there exists a control location $c \in S$ such that for every simple cycle $\mathcal{C} = (c_1, \theta_1, c_2), (c_2, \theta_2, c_3), \ldots, (c_k, \theta_k, c_1)$ ($k \geq 1$, $c_1, \ldots, c_k \in S$, $\theta_1, \ldots, \theta_k \in Op$), there exists $i \in \{1, 2, \ldots, k\}$ such that $c_i = c$. In addition, for every pair $(\mathcal{C}_1, \mathcal{C}_2)$ of simple cycles starting at c, we must have $body(\mathcal{C}_1) \triangleleft body(\mathcal{C}_2) \wedge body(\mathcal{C}_2) \triangleleft body(\mathcal{C}_1)$.*

The correctness of this condition is established as follows. We consider an exploration path of the form

$$\pi = (c_0, m_0) \xrightarrow{\sigma} (c', m_1) \overset{body(\mathcal{C}_1)}{\longrightarrow} (c', m_2) \overset{body(\mathcal{C}_2)}{\longrightarrow} (c', m_3)\cdots,$$

where $c' \in C$, σ is an acyclic sequence of transitions of T, $m_1, m_2, \ldots \in M$, and $\mathcal{C}_1, \mathcal{C}_2, \ldots$ are simple cycles of (C, T).

Let us show that if c' belongs to a maximal strongly connected component of (C, T) that satisfies Condition 3a", then every $\mathcal{C}_j$ occurring an infinite number of times in π is repeatedly executable from at least one reachable memory content. The idea is to prove that for any $n \geq 0$, $\mathcal{C}_j$ can be followed at least n times from the state (c', m_1). Indeed, let π' be a subpath of π starting at (c', m_1) and containing n occurrences of $\mathcal{C}_j$ (not necessarily consecutive). By repeatedly permuting every occurrence of $\mathcal{C}_j$ in π' with all the occurrences of other simple cycles appearing before (this can be done because the sequences of operations labeling the cycles precede favorably each other), one eventually obtains a subpath equivalent to π' beginning with n consecutive occurrences of $\mathcal{C}_j$. The cycle $\mathcal{C}_j$ can thus be followed n times from the state (c', m_1).

From an algorithmic point of view, Condition 3a" can be evaluated directly from its definition with the cost $O(N_{\lhd}(N_{\mathcal{C}})^2)$, where $N_{\mathcal{C}}$ is the number of simple cycles in the strongly connected component, and $N_{\lhd}$ is the maximum cost of comparing two sequences of operations with respect to "$\lhd$".

Let us now study the case of a maximal strongly connected component of (C, T) satisfying Condition 3b. In this case, there is no need for imposing additional restrictions, since Condition 3b is sufficient for ensuring that the algorithm of Figure 4.2 will always find an accepting run whenever one exists. This result is a consequence of the following theorem.

Theorem 5.17 *Let $\mathcal{B} = (C, c_0, M, m_0, Op, T, F)$ be an SMBA associated with a finite set of cycle meta-transitions $\bar{T}$ such that the ESMA $(C, c_0, M, m_0, Op, T, \bar{T})$ is safe, π be a path of transitions corresponding to an accepting run of $\mathcal{B}$, and π_i be a subpath of π entirely contained in a maximal strongly connected component of (C, T) satisfying Condition 3b (let S denote this strongly connected component). If there exists an accepting control location $c \in S \cap F$ visited infinitely many times by π_i, then there exists a meta-transition $(c, f, c) \in \bar{T}$ which is iterable from some memory content $m \in M$ reachable at the location c.*

Proof Suppose that there exists an accepting control location $c \in S \cap F$ visited infinitely many times by π_i. Since there are only finitely many meta-transitions in $\bar{T}$, it is sufficient to show that there exists a memory content $m \in M$ such that (c, m) is reachable, and for every $n \in \mathbf{N}$ there exists a meta-transition $\bar{t} \in \bar{T}$ whose corresponding cycle $\mathcal{C}$ can be followed at least n times from (c, m).

Let C_{π_i} be the set of all the control locations visited infinitely many times by π_i. We split π_i into two paths of transitions π' and π'' such that $\pi_i = \pi'\pi''$ (π' is thus finite), and π' visits at least once every control location belonging to C_{π_i}.

Let $n \in \mathbf{N}$, and let N_c be the number of meta-transitions in $\bar{T}$ beginning at c. Since c appears an infinite number of times in π'', there exists a finite prefix π''' of π'' in which c appears at least $nN_c + 1$ times. We now apply the following

transformation to π_i: to the leftmost subpath of π''' (if one exists) of the form

$$(c_1, m_1) \xrightarrow{t_1} (c_2, m_2) \xrightarrow{t_2} \cdots (c_k, m_k) \xrightarrow{t_k} (c_1, m_{1'}),$$

where $k \geq 1$, $c_1, \ldots, c_k \in S$, $m_1, \ldots, m_k \in M$ and $t_1, \ldots, t_k \in T$, $c_j \neq c$ for every $j \in \{1, 2, \ldots, k\}$ and $t_1; t_2; \ldots; t_k$ is a simple cycle $\mathcal{C}$ of (C, T), we apply Lemma 5.6 and move this occurrence of $\mathcal{C}$ just after the rightmost appearance of c_1 in π'. In other words, we transform

$$\pi_i = \overbrace{(c'_1, m'_1) \xrightarrow{t'_1} \cdots \xrightarrow{t'_{l-1}} (c_1, m_{1''}) \xrightarrow{t'_l} \cdots\cdots}^{\pi'} \overbrace{\xrightarrow{t'_{l'-1}} (c_1, m_1) \xrightarrow{body(\mathcal{C})} (c_1, m_{1'}) \xrightarrow{t'_{l'}} \cdots\cdots}^{\pi'''},$$

where $l, l' \in \mathbf{N}_0$, $c'_1, \ldots \in S$, $m_{1''}, m'_1, \ldots \in M$ and $t'_1, t'_2, \ldots \in T$, into an equivalent path of the form

$$\overbrace{(c'_1, m'_1) \xrightarrow{t'_1} \cdots \xrightarrow{t'_{l-1}} (c_1, m_{1''}) \xrightarrow{body(\mathcal{C})} (c_1, m_{1'''}) \xrightarrow{t'_l} \cdots\cdots}^{\pi'} \overbrace{\xrightarrow{t'_{l'-1}} (c_1, m_{1''''}) \xrightarrow{t'_{l'}} \cdots\cdots}^{\pi'''},$$

where $m_{1'''}, m_{1''''} \in M$. Repeating this operation until a fixpoint is reached (i.e., until π_i does not change when the operation is subsequently performed), one eventually obtains a subpath equivalent to π_i containing at least nN_c successive occurrences of simple cycles beginning at c. This implies that there exists a simple cycle $\mathcal{C}_i$ starting at c that occurs at least n times in the subpath. Repeating again the move operation so as to shift left all the occurrences of simple cycles different from $\mathcal{C}_i$ in the subpath, we finally obtain a subpath equivalent to π_i in which $\mathcal{C}_i$ can repeatedly be followed at least n times from a reachable state. Applying Lemma 5.6 n times, one can move left the n occurrences of $\mathcal{C}_i$ just right of the leftmost appearance of c in the subpath, obtaining an equivalent subpath in which $\mathcal{C}_i$ can repeatedly be followed n times from a reachable state that does not depend on n. □

5.5.2 Summary of Conditions

Let us summarize the necessary conditions obtained in Section 5.5.1.

Definition 5.18 *An SMBA associated with a finite set of cycle meta-transitions is* safe *if it satisfies Conditions 1 and 2, and if each maximal strongly connected component of its control graph satisfies either Condition 3a" or Condition 3b (the satisfied condition may differ for each strongly connected component).*

We are now ready to state the main result of this section.

Theorem 5.19 *Let $\mathcal{B}$ be an SMBA which is safe when associated with a set of cycle meta-transitions $\bar{T}$. The problem which consists of determining whether $\mathcal{B}$ has an accepting run is decidable.*

function SMBA-EMPTY-SC?(SMBA $(C, c_0, M, m_0, Op, T, F)$, set of cycles $S_{\mathcal{C}}$) : $\{\mathbf{T}, \mathbf{F}\}$

```
1:  var Q_R : set of states;
2:      T̄ : set of meta-transitions;
3:      c : control location;
4:  begin
5:      T̄ := {(c, σ*, c) ∈ C × Op* × C | (∃C ∈ S_C)(σ = body(C) ∧ c = first(C)))};
6:      Q_R := REACHABLE((C, c_0, M, m_0, Op, T, T̄));
7:      for each c ∈ C such that values(Q_R, c) ≠ ∅ do
8:          for each (c_1, θ_1, c_2), (c_2, θ_2, c_3), ..., (c_k, θ_k, c_1) ∈ S_C such that c = c_1
                                                  and {c_1, c_2, ..., c_k} ∩ F ≠ ∅ do
9:              if values(Q_R, c) ∩ ITERABLE(θ_1; θ_2; ...; θ_k) ≠ ∅ then
10:                 return F;
11:     return T
12: end.
```

Figure 5.2: Test of emptiness for safe SMBAs (with only cycle meta-transitions).

Proof The problem can be decided by the algorithm of Figure 5.2, which is a variant of the semi-algorithm presented in Figure 4.2 to which the following minor modifications have been applied:

- The set $\bar{T}$ of meta-transitions associated to $\mathcal{B}$ is computed from a set of cycles supplied as an argument (instead of being an actual argument);
- The function OPEN-SET is implemented with the help of ITERABLE;
- The cycles that are checked are required to visit an accepting control location instead of beginning at such a location.

□

The first two changes are motivated by the fact that we are here dealing with cycle meta-transitions, as opposed to arbitrary ones. The purpose of the third modification is to take into account Condition 3a".

The correctness of this algorithm is established as follows. Let $S_{\mathcal{C}}$ be the set of cycles to which the meta-transitions in $\bar{T}$ are associated. The termination of the computation of SMBA-EMPTY-SC?$(\mathcal{B}, S_{\mathcal{C}})$ is a consequence of $(C, c_0, M, m_0, Op, T, \bar{T})$ being a safe ESMA. It remains to show that SMBA-EMPTY-SC?$(\mathcal{B}, S_{\mathcal{C}})$ returns **F** if and only if $\mathcal{B}$ has an accepting run.

One direction is trivial. Indeed, if the algorithm returns **F**, then there is a reachable state from which there exists a cycle that can be followed an unbounded number of times and that visits an accepting control location. This defines a path of transitions corresponding to an accepting run of $\mathcal{B}$.

Reciprocally, if $\mathcal{B}$ has an accepting run, then we have established in Section 5.5.1 that there exists such a run in which the same simple cycle is repeatedly followed from a reachable state, with this cycle visiting an accepting control location. Since $\mathcal{B}$ is safe, this simple cycle must correspond to a meta-transition, hence must belong to $S_{\mathcal{C}}$, and thus an accepting run will be detected by the algorithm. □

5.5.3 Systems with Only Multicycle Meta-Transitions

We now consider an SMBA $\mathcal{B} = (C, c_0, M, m_0, Op, T, F)$ associated with a finite set of multicycle meta-transitions $\bar{T}$, such that the ESMA $(C, c_0, M, m_0, Op, T, \bar{T})$ is safe. For any accepting run of $\mathcal{B}$, there exists an accepting control location $c \in F$ visited infinitely often by the path of transitions π corresponding to the run. Like in Section 5.5.1, we distinguish two possible situations depending on the maximal strongly connected component of (C, T) to which c belongs (let S be this strongly connected component).

The first situation is when S satisfies Condition 3a. In this case, we know that there exists $c' \in S$ such that every simple cycle contained in S visits c'. As a consequence, we have that π is of the form

$$(c_0, m_0) \xrightarrow{\sigma} (c', m_1) \overset{body(\mathcal{C}_1)}{\rightarrow} (c', m_2) \overset{body(\mathcal{C}_2)}{\rightarrow} (c', m_3) \cdots,$$

where σ is an acyclic sequence of transitions of T, $m_1, m_2, \ldots \in M$, and $\mathcal{C}_1, \mathcal{C}_2, \ldots$ are simple cycles of (C, T). Since Condition 2' is satisfied, there is a meta-transition $\bar{t} \in \bar{T}$ which is associated to a superset of $\{\mathcal{C}_1, \mathcal{C}_2, \ldots\}$. From Definition 4.9, it follows that $\bar{t}$ is iterable for the memory content m_1. As a consequence, if $\mathcal{B}$ has an accepting run, then it has a meta-transition which is iterable from a reachable state and such that its associated set contains at least one cycle that visits an accepting control location.

Unfortunately, the reciprocal is not true. Indeed, let $\bar{t}$ be a meta-transition associated to the set of cycles $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$ $(k > 1)$. Assume that $\mathcal{C}_1$ visits an accepting control location and that $\bar{t}$ is iterable from the reachable state $(c, m) \in C \times M$. This does not imply that there exists a run of $\mathcal{B}$ in which $\mathcal{C}_1$ appears an unbounded number of times. Actually, $\mathcal{C}_2$ may very well be the only $\mathcal{C}_j$ that can be followed from a reachable state of $\mathcal{B}$, and might visit only non-accepting control locations.

It is therefore necessary to introduce an additional requirement. We simply impose that if a maximal strongly connected component of (C, T) satisfies Condition 3a

and does not satisfy Condition 3b, then it must be such that if it contains an accepting control location, it also contains such a location visited by all the simple cycles of the component. Formally, this is expressed by the following condition.

Condition 4 *The strongly connected component $S \subseteq C$ in the control graph of $\mathcal{A}$ must be such that if $S \cap F \neq \emptyset$, then there exists $c \in S \cap F$ such that for every simple cycle $\mathcal{C} = (c_1, \theta_1, c_2), (c_2, \theta_2, c_3), \ldots, (c_k, \theta_k, c_1)$ ($k \geq 1$, $c_1, \ldots, c_k \in C$, $\theta_1, \ldots, \theta_k \in Op$), there exists $i \in \{1, 2, \ldots, k\}$ such that $c_i = c$.*

If this condition is satisfied by a strongly connected component S, then the SMBA admits an exploration path π that visits infinitely many accepting control locations in S if and only if S contains an accepting control location for which there is a reachable memory content as well as a meta-transition repeatedly open from that memory content.

From an algorithmic point of view, Condition 4 is easily checked by computing the intersection of the sets of control locations visited by every simple cycle and the set of accepting control locations of the strongly connected component. The condition is satisfied if and only if this intersection is not empty.

Let us now study the case of a maximal strongly connected component S satisfying Condition 3b. In this case, there is no need for imposing additional restrictions, since Condition 3b is sufficient for ensuring that the algorithm of Figure 4.2 will always find an accepting run whenever one exists. This result is formalized by the following theorem.

Theorem 5.20 *Let $\mathcal{B} = (C, c_0, M, m_0, Op, T, F)$ be an SMBA associated with a finite set of multicycle meta-transitions $\bar{T}$ such that the ESMA $(C, c_0, M, m_0, Op, T, \bar{T})$ is safe, π be a path of transitions corresponding to an accepting run of $\mathcal{B}$, and π_i be a subpath of π entirely contained in a maximal strongly connected component of (C, T) satisfying Condition 3b (let S denote this strongly connected component). If there exists an accepting control location $c \in S \cap F$ visited infinitely many times by π_i, then there exists a meta-transition $(c, f, c) \in \bar{T}$ which is iterable for some memory content $m \in M$ reachable at the location c.*

Proof The proof is similar to the one of Theorem 5.17. Suppose that there exists an accepting control location $c \in S \cap F$ visited infinitely many times by π_i. Since there are only finitely many meta-transitions in $\bar{T}$, it is sufficient to show that there exists a memory content $m \in M$ such that (c, m) is reachable and for every $n \in \mathbf{N}$, there exists a meta-transition $\bar{t} \in \bar{T}$ such that its corresponding cycles can be followed at least n times from (c, m) (the cycle that is followed may differ from one iteration to the other).

Let C_{π_i} be the set of all the control locations visited infinitely many times by π_i. We split π_i into two paths of transitions π' and π'' such that $\pi_i = \pi'\pi''$ (π' is thus finite) and π' visits at least once every control location belonging to C_{π_i}.

Let $n \in \mathbf{N}$. Since c appears an infinite number of times in π'', there exists a finite prefix π''' of π'' in which c appears at least $n+1$ times. We now apply the following transformation to π_i: to the leftmost subpath of π''' (if one exists) of the form

$$(c_1, m_1) \xrightarrow{t_1} (c_2, m_2) \xrightarrow{t_2} \cdots (c_k, m_k) \xrightarrow{t_k} (c_1, m_{1'}),$$

where $k \geq 1$, $c_1, \ldots, c_k \in S$, $m_1, \ldots, m_k \in M$, $t_1, \ldots, t_k \in T$, $c_j \neq c$ for every $j \in \{1, 2, \ldots, k\}$ and $t_1; t_2; \ldots; t_k$ is a simple cycle $\mathcal{C}$ of (C, T), we apply Lemma 5.6 and move the occurrence of $\mathcal{C}$ just after the rightmost appearance of c_1 in π'. In other words, we transform

$$\pi_i = \overbrace{(c'_1, m'_1) \xrightarrow{t'_1} \cdots \xrightarrow{t'_{l-1}} (c_1, m_{1''}) \xrightarrow{t'_l} \cdots\cdots}^{\pi'} \overbrace{\xrightarrow{t'_{l'-1}} (c_1, m_1) \xrightarrow{body(\mathcal{C})} (c_1, m_{1'}) \xrightarrow{t'_{l'}} \cdots\cdots}^{\pi'''},$$

where $l, l' \in \mathbf{N}_0$, $c'_1, \ldots \in S$, $m_{1''}, m'_1, \ldots \in M$ and $t'_1, t'_2, \ldots \in T$, into an equivalent path of the form

$$\overbrace{(c'_1, m'_1) \xrightarrow{t'_1} \cdots \xrightarrow{t'_{l-1}} (c_1, m_{1''}) \xrightarrow{body(\mathcal{C})} (c_1, m_{1'''}) \xrightarrow{t'_l} \cdots\cdots}^{\pi'} \overbrace{\xrightarrow{t'_{l'-1}} (c_1, m_{1''''}) \xrightarrow{t'_{l'}} \cdots\cdots}^{\pi'''},$$

where $m_{1'''}, m_{1''''} \in M$. Repeating this operation until a fixpoint is reached (i.e., until π_i does not change when the operation is subsequently performed), one eventually obtains a subpath equivalent to π_i containing at least n successive occurrences of simple cycles starting at c. Applying Lemma 5.6, one can move left the n occurrences of these cycles just right of the leftmost appearance of c in the subpath, obtaining an equivalent subpath in which the cycles can be followed n times from a reachable state that does not depend on n. □

As a consequence of this theorem, we have that if a maximal strongly connected component S satisfies Condition 3b, then the SMBA admits an exploration path π visiting infinitely often an accepting control location in S if and only if S contains an accepting control location for which there exist a reachable memory content as well as a meta-transition iterable from that memory content.

5.5.4 Summary of Conditions

Let us summarize the necessary conditions obtained in Section 5.5.3.

Definition 5.21 *An SMBA associated with a finite set of multicycle meta-transitions is* safe *if it satisfies Conditions 1 and 2, and if each maximal strongly connected component of its control graph satisfies either Conditions 3a" and 4, or Condition 3b (the satisfied condition(s) may differ for each strongly connected component).*

We are now ready to state the main result of this section.

function SMBA-EMPTY-MC?(SMBA $(C, c_0, M, m_0, Op, T, F)$, set of sets of cycles U) : $\{\mathbf{T}, \mathbf{F}\}$

1: **var** Q_R : **set of** states;
2: $\bar{T}$: **set of** meta-transitions;
3: $S_{\mathcal{C}}$: **set of** cycles;
4: c : control location;
5: **begin**
6: $\bar{T} := \{(c, S_{\mathcal{C}}^*, c) \in C \times Op^* \times C \mid (\exists S_{\mathcal{C}} \in U)(\forall \mathcal{C} \in S_{\mathcal{C}})(c = \mathit{first}(\mathcal{C}))\}$;
7: $Q_R :=$ REACHABLE$((C, c_0, M, m_0, Op, T, \bar{T}))$;
8: **for each** $c \in F$ **such that** $\mathit{values}(Q_R, c) \neq \emptyset$ **do**
9: **for each** $S_{\mathcal{C}} \in U$ **such that**$(\forall \mathcal{C} \in S_{\mathcal{C}})(\mathit{first}(\mathcal{C}) = c)$ **do**
10: **if** $\mathit{values}(Q_R, c) \cap$ MULTI-ITERABLE$(\{\mathit{body}(\mathcal{C}_i) \mid \mathcal{C}_i \in S_{\mathcal{C}}\}) \neq \emptyset$ **then**
11: **return F**;
12: **return T**
13: **end**.

Figure 5.3: Test of emptiness for safe SMBAs (multicycle meta-transitions).

Theorem 5.22 *Let $\mathcal{B}$ be an SMBA which is safe when associated with a set of multicycle meta-transitions $\bar{T}$. The problem which consists of determining whether $\mathcal{B}$ has an accepting run is decidable.*

Proof The problem can be decided by the algorithm of Figure 5.3, which is an adaptation of the algorithm of Figure 5.2 to the case of multicycle meta-transitions.

The correctness of this algorithm is established as follows. Let U be the set of sets of cycles to which the meta-transitions in $\bar{T}$ correspond. The termination of the computation of SMBA-EMPTY-MC?$(\mathcal{B}, U)$ is a consequence of $(C, c_0, M, m_0, Op, T, \bar{T})$ being a safe ESMA. It remains to show that SMBA-EMPTY-MC?$(\mathcal{B}, U)$ returns **F** if and only if $\mathcal{B}$ has an accepting run.

One direction is trivial. Indeed, if the algorithm returns **F**, then there exists a reachable state whose control location is accepting, and from which there is a group of cycles that can be followed an unbounded number of times. This defines an path of transitions corresponding to an accepting run of $\mathcal{B}$.

Reciprocally, if $\mathcal{B}$ has an accepting run, then we have established in Section 5.5.3 that there exists such a run in which a group of simple cycles is repeatedly followed from a reachable state whose control location is accepting. Since $\mathcal{B}$ is safe, a set containing those simple cycles must correspond to a meta-transition of $\bar{T}$, hence

must belong to U, and thus will be detected by the algorithm. □

5.6 Control Graph Optimization

In Sections 5.3, 5.4 and 5.5, two types of decidable conditions were obtained: conditions on the shape of the control graph (Conditions 3a, 3a" and 4), and conditions on the memory operations labeling transitions or meta-transitions (Conditions 1, 2, 3b, 1' and 2'). Conditions of the latter type depend strongly on the memory domain that is used, and are not very restrictive whenever this domain leads to a great amount of independence between memory operations. There are even domains for which those conditions are always trivially satisfied regardless of the particular system that is considered. On the other hand, conditions of the former type are more easily fulfilled whenever the formalism used for specifying models imposes restrictions on the shape of the control graph. For instance, this is the case of some structured languages for which loops can only be nested within each other rather than arbitrarily intertwined.

In this section, we go further and show that it is sometimes possible to modify the control graph of an SMA or SMBA in such a way that their behavior is not affected (i.e., their set of accepting states or the emptiness or their accepting language stays unchanged), so as to guarantee that sufficient conditions for termination are satisfied, or at least facilitate state-space exploration.

5.6.1 Introduction

The type of optimization proposed here concerns nested loops, i.e., loops containing loops. Roughly speaking, the idea is that the effect of following repeatedly a cycle in the control graph of an SMA or SMBA is sometimes equivalent to following a single transition. If the parameters of this transition can be computed, replacing the cycle by its equivalent transition may make it possible to obtain new simple cycles, and thus possibly new simple-cycle meta-transitions. Another advantage (which will appear later) is that cycle replacement splits the strongly connected components of the control graph. This increases the possibility of satisfying Conditions 3a, 3a" and 4.

Let us illustrate on a simple example how meta-transitions can be associated to cycles containing other cycles. The program given in Figure 5.4 is composed of two nested loops. Its control graph is depicted in Figure 5.5 (in this figure, the indices of control locations correspond to line numbers in the program). The only useful simple-cycle meta-transitions that can be created correspond to the cycle $(c_{4'}, j \leq i, c_5), (c_5, k := k+1, c_{5'}), (c_{5'}, j := j+1, c_{4'})$ (or one of its rotations). Indeed, since there is no reachable memory content from which the cycle $(c_{3'}, i \leq$

```
program NESTED-LOOPS()
1:    var i, j, k : integers;
2:    begin
3:          k := 0;
4:          for i := 1 to 1000 do
5:                for j := 1 to i do
6:                      k := k + 1
7:    end.
```

Figure 5.4: Example of program with nested loops.

$1000, c_4), (c_4, j := 1, c_{4'}), (c_{4'}, j > i, c_{4''}), (c_{4''}, i := i + 1, c_{3'})$ can be followed, meta-transitions corresponding to this cycle or its rotations would be useless for state-space exploration.

It is however easy to show that the overall effect of the inner loop at Lines 5–6, i.e., the transformation undergone by the variables values whenever those lines are completely executed, is equivalent to the sequence of instructions

```
if j <= i then
    begin
        j := i + 1;
        k := k + i
    end
```

As a consequence, any subpath in the graph of Figure 5.5 that begins with the transition $(c_{3'}, i \leq 1000, c_4)$, then visits states belonging to $\{c_4, c_{4'}, c_{4''}, c_5, c_{5'}\}$, and finally ends with the transition $(c_{4''}, i := i+1, c_{3'})$ is equivalent to the cycle depicted at Figure 5.6. One can therefore associate with the SMA of Figure 5.5 the meta-transition

$$(c_{3'}, (i \leq 1000; j := 1; j \leq i; j := i + 1; k := k + i; j > i; i := i + 1)^*, c_{3'})$$

corresponding to that equivalent cycle.

5.6.2 Loop Optimization

The idea behind loop optimization is to replace in the control graph of an SMA or SMBA some cycles by their equivalent transition. The first step is to characterize precisely the cycles that can be replaced.

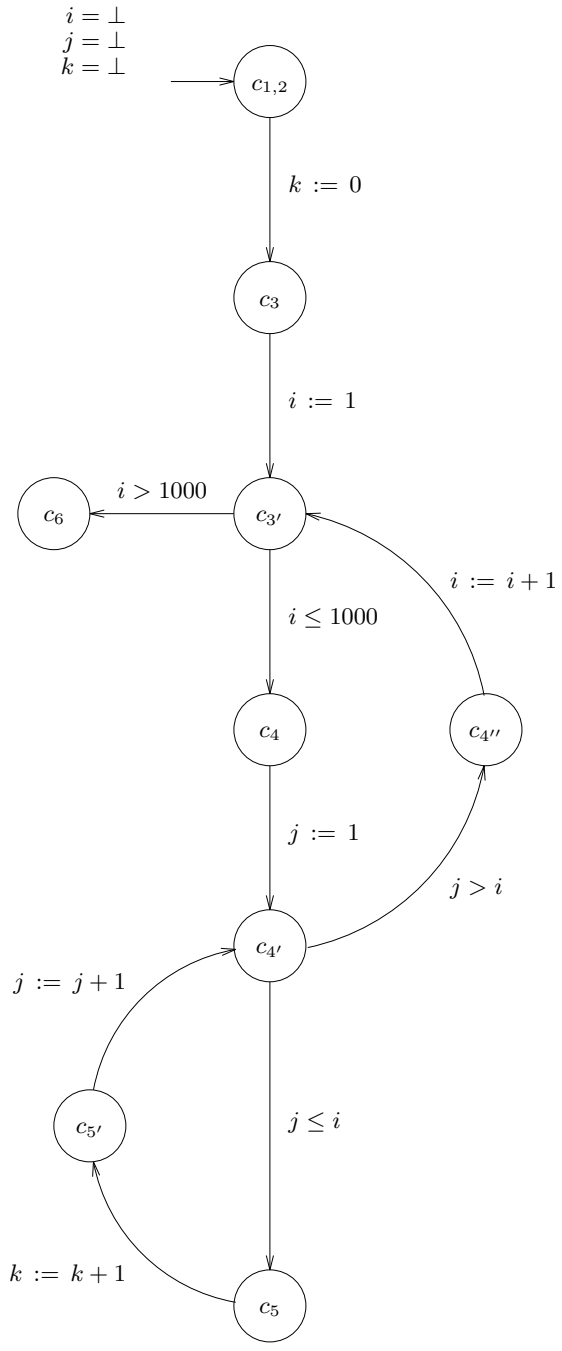

Figure 5.5: Control graph of program with nested cycles.

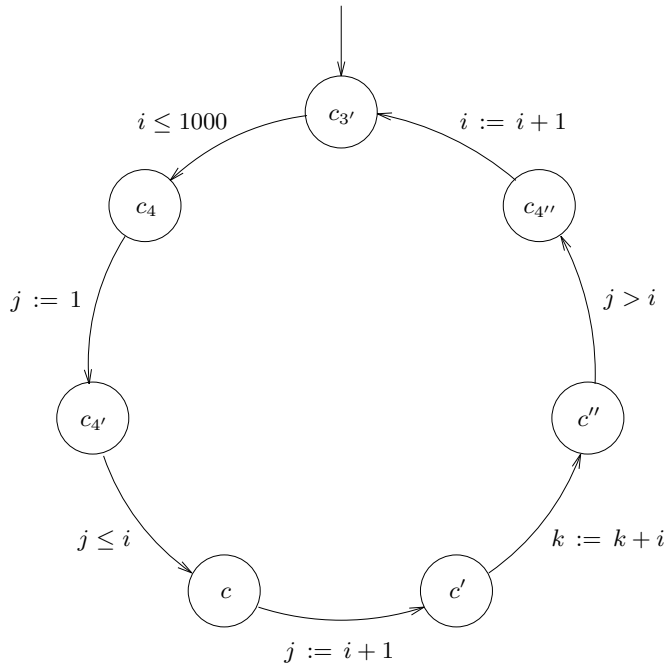

Figure 5.6: Cycle equivalent to nested loops.

Definition 5.23 *Let $\mathcal{A} = (C, c_0, M, m_0, Op, T)$ be an SMA or $\mathcal{A} = (C, c_0, M, m_0, Op, T, F)$ be an SMBA and $\mathcal{C}$ be a simple cycle in the control graph of $\mathcal{A}$. The cycle $\mathcal{C}$ is* optimizable *if the following conditions are satisfied:*

- *The cycle $\mathcal{C}$ can be associated with a meta-transition;*
- *Each control location visited by $\mathcal{C}$ except first($\mathcal{C}$) has exactly one incoming transition;*
- *The control location first($\mathcal{C}$) has exactly two outgoing transitions;*
- *There exists a function $f : M \to M$ over the memory domain such that for any $m \in M$, we have*

$$\{m' \in M \mid (\exists k \in \mathbf{N}_0)(m' = \theta(body(\mathcal{C})^k(m)))\} = \{f(m)\},$$

where θ is the memory operation labeling the outgoing transition from first($\mathcal{C}$) that does not belong to $\mathcal{C}$. The corresponding function

$$g : 2^M \to 2^M : U \mapsto \{f(m) \mid m \in U\}$$

is computable with respect to the representation system used for sets of memory contents. In addition, an algorithm for applying g can be obtained algorithmically from the specifications of $\mathcal{C}$ and θ.

function OPTIMIZE-CYCLE-SMBA(SMBA $(C, c_0, M, m_0, Op, T, F)$, cycle $\mathcal{C}$) : SMBA

1: **var** c_a, c_b, c_1, c, c' : control locations;
2: θ : memory operation;
3: **begin**
4: $C := C \cup \{c_a, c_b\}$;
5: $c_1 := \mathit{first}(\mathcal{C})$;
6: **for each** $(c, \theta, c') \in T \setminus \mathcal{C}$ **such that** $c' = c_1$ **do**
7: $T := (T \setminus \{(c, \theta, c')\}) \cup \{(c, \theta, c_a)\}$;
8: **for each** $(c, \theta, c') \in T \setminus \mathcal{C}$ **such that** $c = c_1$ **do**
9: $T := T \cup \{(c_a, (\mathit{body}(\mathcal{C})^+; \theta), c_b), (c_b, \mathrm{id}, c'), (c_a, \theta, c')\}$;
10: $T := T \cup \{(c_a, \mathrm{id}, c_1)\}$;
11: **if** $(\exists (c, \theta, c') \in \mathcal{C})(c \in F)$ **then** $F := F \cup \{c_b\}$;
12: **if** $c_0 = c_1$ **then** $c_0 := c_a$;
13: **return** $(C, c_0, M, m_0, Op, T, F)$
14: **end**.

Figure 5.7: Loop optimization for SMBAs.

Intuitively, a cycle is optimizable if it has exactly one entry and one exit, and if the overall effect of entering the cycle, following it an arbitrary number of times, and then exiting it is equivalent to a transition whose parameters can be computed.

If a cycle is optimizable, then optimizing it consists of modifying the control graph of the SMA or SMBA in such a way that repeated occurrences of this cycle in exploration paths are transformed into one occurrence of its equivalent transition. The difficulty is to ensure that the optimized state machine has the same reachable states as the original one (if the machine is an SMA) or is equivalent with respect to the emptiness of its accepted language (if the machine is an SMBA).

The optimization algorithms are given in Figures 5.7 and 5.8. Intuitively, they proceed by replacing the only transition that allows to exit the cycle (let t denote this transition) by a transition equivalent to repeated executions of the cycle followed by t. The replacement operation is illustrated in Figure 5.9. The correctness of the algorithms is established by two theorems. The first one expresses that the optimized state machine has the same reachable states as the original one, provided that the control locations added during the optimization are not taken into account.

Theorem 5.24 *Let $\mathcal{A} = (C, c_0, M, m_0, Op, T)$ be an SMA, $\mathcal{C}$ be an optimizable cycle in its control graph (C, T), and $\mathcal{A}'$ be the SMA returned by the function*

function OPTIMIZE-CYCLE-SMA(SMA $(C, c_0, M, m_0, Op, T, F)$, cycle $\mathcal{C}$) : SMA

1: **var** $(C', c'_0, M', m'_0, Op', T', F')$: SMBA;
2: **begin**
3: $(C', c'_0, M', m'_0, Op', T', F') :=$
OPTIMIZE-CYCLE-SMBA($(C, c_0, M, m_0, Op, T, \emptyset), \mathcal{C}$);
4: **return** $(C', c'_0, M', m'_0, Op', T')$
5: **end**.

Figure 5.8: Loop optimization for SMAs.

call OPTIMIZE-CYCLE-SMA$(\mathcal{A}, \mathcal{C})$*. The sets* Q_R *and* Q'_R *of reachable states of* $\mathcal{A}$ *and* $\mathcal{A}'$ *(respectively) are such that for every* $c \in C$*, we have* $values(Q_R, c) = values(Q'_R, c)$*.*

Proof The proof is in two parts. We first show that for every exploration path of $\mathcal{A}$, there exists a corresponding path of $\mathcal{A}'$. Then, we show that for every exploration path of $\mathcal{A}'$ ending in a control location that belongs to C, there exists a corresponding path of $\mathcal{A}$.

- *Let* $c \in C$ *and* $m \in M$*. If there exists a path* π *of* $\mathcal{A}$ *leading from* (c_0, m_0) *to* (c, m)*, then there exists a path* π' *of* $\mathcal{A}'$ *leading from* (c_0, m_0) *to* (c, m)*.* Let $c_1 = first(\mathcal{C})$ and $t = (c_1, \theta, c'_3)$ be the outgoing transition from c_1 that does not belong to $\mathcal{C}$. The only transitions of $\mathcal{A}$ that are modified during the optimization operation are those ending in c_1 and t. One can transform π into π' as follows (c_a and c_b denote the control locations created at Line 4 of the algorithm):

  1. For every occurrence of t in π, one performs the following operations. Let $l \geq 0$ be the number of occurrences of the sequence of transitions labeling $\mathcal{C}$ immediately left of t in π, and let t' be the transition immediately left of those occurrences of $body(\mathcal{C})$ (or of t if $l = 0$) in π, if such a transition exists. We thus have that π is of the form

  $$\pi = \cdots (c'_2, m) \xrightarrow{t'} (c_1, m') \overset{body(\mathcal{C})^l}{\rightarrow} (c_1, m'') \xrightarrow{t} (c'_3, m''') \cdots,$$

  with $c'_2, c'_3 \in C$, $m', m'', m''' \in M$, and $t' = (c'_2, \theta', c_1)$. If $l = 0$, one simply replaces the subpath between (c'_2, m) and (c'_3, m''') by

  $$(c'_2, m) \xrightarrow{\theta'} (c_a, m') \xrightarrow{\theta} (c'_3, m''').$$

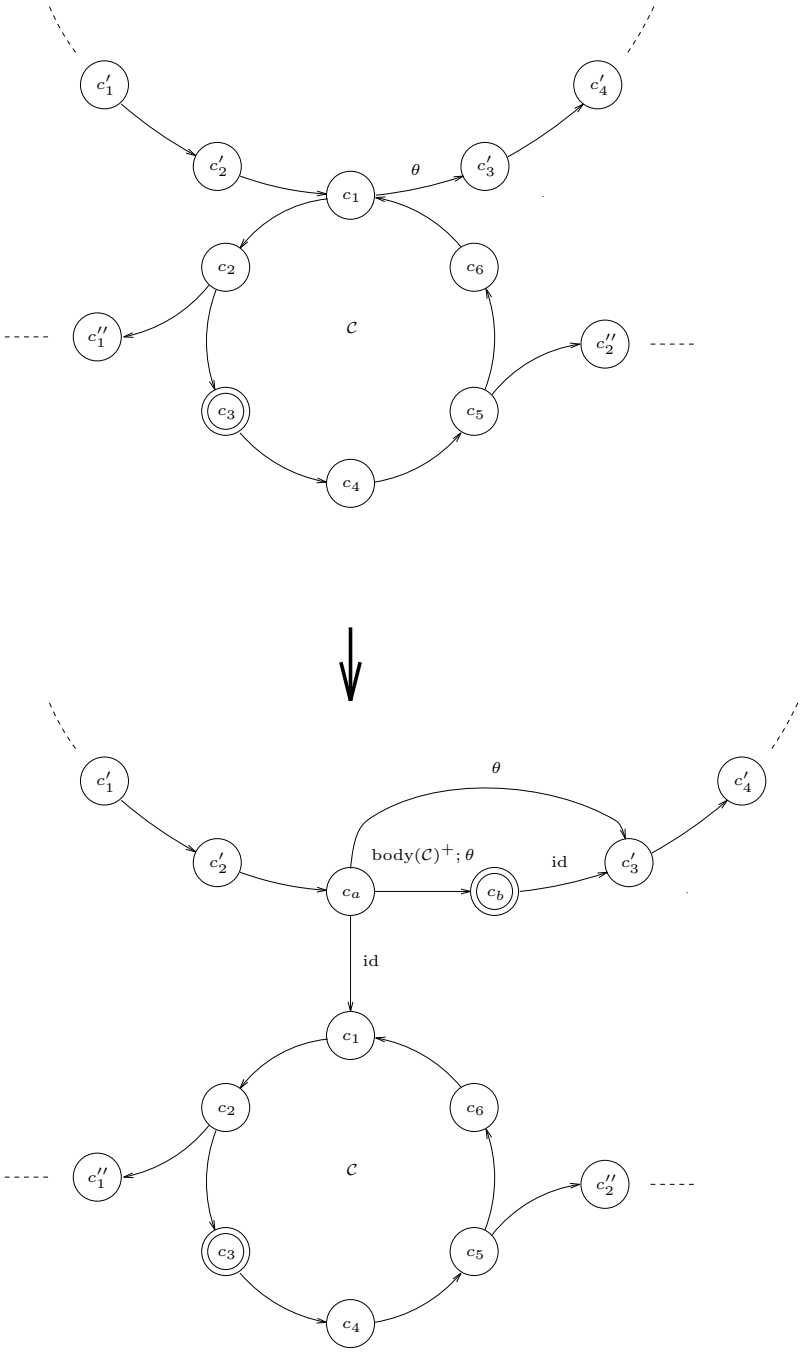

Figure 5.9: Illustration of loop optimization.

If $l > 0$, one replaces that subpath by

$$(c'_2, m) \xrightarrow{\theta'} (c_a, m') \overset{body(\mathcal{C})^+;\theta}{\Rightarrow} (c_b, m''') \xrightarrow{\mathrm{id}} (c'_3, m''').$$

If t' does not exist, i.e., if c_1 is the initial control location of $\mathcal{A}$ and there is no transition preceding the occurrence of $\mathcal{C}$ in π, then the same operations are performed without t' appearing in the original and modified subpaths.

2. For every occurrence in π of the first transition t_1 of $\mathcal{C}$ which is not contained in a sequence of complete occurrences of $\mathcal{C}$ followed by t, one performs the following operation. Let t' be the transition immediately left of t_1 in π (if this transition exists). We thus have that π is of the form

$$\pi = \cdots (c'_2, m) \xrightarrow{t'} (c_1, m') \xrightarrow{t_1} (c_2, m''),$$

with $c'_2, c_2 \in C$, $m', m'' \in M$. One replaces the subpath between (c'_2, m) and (c_1, m') by

$$(c'_2, m) \xrightarrow{\theta'} (c_a, m') \xrightarrow{\mathrm{id}} (c_1, m').$$

If t' does not exist, then the same operation is performed without t' appearing in the original and modified subpaths.

- *Let $c \in C$ and $m \in M$. If there exists a path π' of $\mathcal{A}'$ leading from (c_0, m_0) to (c, m), then there exists a path π of $\mathcal{A}$ leading from (c_0, m_0) to (c, m).* Let $c_1 = \mathit{first}(\mathcal{C})$. The only transitions of $\mathcal{A}'$ that are not transitions of $\mathcal{A}$ are those ending in c_1, $t_1 = (c_a, \mathit{body}(\mathcal{C})^+; \theta, c_b)$, $t_2 = (c_a, \theta, c'_3)$ $t_3 = (c_a, \mathrm{id}, c_1)$ and $t_4 = (c_b, \mathrm{id}, c'_3)$, with $c'_3 \in C$. One applies the following modifications to π':

  - Each occurrence of a transition ending in c_a is replaced by an identical transition ending in c_1;
  - Each occurrence of t_1 is replaced by the appropriate number of occurrences of $\mathit{body}(\mathcal{C})$, followed by one occurrence of the only transition t of $\mathcal{A}$ that goes from c_1 and does not belong to $\mathcal{C}$;
  - Each occurrence of t_2 is replaced by t;
  - All the occurrences of t_3 and t_4 are deleted.

  The result is a path π of $\mathcal{A}$ equivalent to π'.

□

The second theorem establishes that loop-optimizing an SMBA does not influence the emptiness of its accepted language.

Theorem 5.25 *Let $\mathcal{B} = (C, c_0, M, m_0, Op, T, F)$ be an SMBA, $\mathcal{C}$ be an optimizable cycle in (C, T), and $\mathcal{B}'$ be the SMBA returned by OPTIMIZE-CYCLE-SMBA$(\mathcal{B}, \mathcal{C})$. The language accepted by $\mathcal{B}$ is empty if and only if the language accepted by $\mathcal{B}'$ is empty.*

Proof Let π be an exploration path corresponding to an (infinite) run of $\mathcal{B}$. The transformation introduced in the proof of Theorem 5.24 allows to transform this path into a path π' corresponding to an infinite run of $\mathcal{B}'$. The reciprocal transformation is possible as well. It remains to show that π visits infinitely many accepting control locations if and only if π' visits infinitely many accepting control locations. Let $c_1 = \mathit{first}(\mathcal{C})$, $t = (c_1, \theta, c_3')$ be the only outgoing transition from c_1 that does not belong to $\mathcal{C}$, and c_a, c_b be the control locations created by the optimization algorithm at Line 4 of OPTIMIZE-CYCLE-SMA.

- *If π visits infinitely many accepting control locations, then π' visits infinitely many accepting control locations.* The only control locations that do not necessarily appear in π' each time they are visited in π are those visited by $\mathcal{C}$. Indeed, repeated occurrences of $\mathcal{C}$ in π may correspond to a single occurrence of $(c_a, \mathit{body}(\mathcal{C})^+; \theta, c_b)$ in π'. However, since c_b is accepting provided that $\mathcal{C}$ visits at least one accepting control location, we have that any finite number of visits of accepting control locations by π within successive occurrences of $\mathcal{C}$ corresponds to one visit of the accepting control location c_b in π'. The transformation thus preserves the accepting nature of the path.

- *If π' visits infinitely many accepting control locations, then π visits infinitely many accepting control locations.* The only accepting control location that does not appear in π each time it appears in π' is c_b. Since each occurrence of c_b in π' corresponds to at least one complete occurrence of $\mathcal{C}$ in π, and since $\mathcal{C}$ visits at least one accepting control location provided that c_b is accepting, we have that the transformation of π' into π preserves the accepting nature of the path.

□

5.6.3 Implementation

Here, we study how one can use the loop optimization algorithms proposed in Section 5.6.2 in an actual implementation.

The first problem concerns an operation performed by the algorithms. Given a sequence of memory operations $\sigma \in Op^*$ and a memory operation $\theta \in Op$, one should be able to decide whether a computable operation equivalent to $(\sigma^+; \theta)$ can

function OPTIMIZABLE?(SMA (C, c_0, M, m_0, Op, T) or SMBA $(C, c_0, M, m_0, Op, T, F)$, simple cycle $\mathcal{C}$) : $\{\mathbf{T}, \mathbf{F}\}$

```
1:  var c, c', c'', c''' : control locations;
2:      θ, θ' : memory operations;
3:  begin
4:      if ¬(META?(body(C))) then
5:          return F;
6:      c_1 := first(C);
7:      if (∃(c, θ, c') ∈ C, (c'', θ', c''') ∈ T)((c, θ, c') ≠ (c'', θ', c''') ∧ c' ≠ c_1
                                                          ∧ c' = c''') then
8:          return F;
9:      if (∃(c, θ, c'), (c'', θ', c''') ∈ T \ C)((c, θ, c') ≠ (c'', θ', c''') ∧ c = c'' = c_1) then
10:         return F;
11:     for each (c, θ, c') ∈ T \ C such that c = c_1 do
12:         return EXISTS-LOOP-EQUIV?(body(C), θ);
13:     return F
14: end.
```

Figure 5.10: Decision procedure for optimizability of a simple cycle.

be obtained. We therefore require the existence of a computable predicate EXISTS-LOOP-EQUIV? : $Op^* \times Op \to \{\mathbf{T}, \mathbf{F}\}$ and a computable function LOOP-EQUIV-OP : $Op^* \times Op \to Op$ such that for any $\sigma \in Op^*$ and $\theta \in Op$ such that EXISTS-LOOP-EQUIV?$(\sigma, \theta) = \mathbf{T}$, LOOP-EQUIV-OP$(\sigma, \theta)$ returns a memory operation $\theta' \in Op$ equivalent to $(\sigma^+; \theta)$. In practice, those predicate and function strongly depend on the representation system used for sets of memory contents. Implementations of EXISTS-LOOP-EQUIV? and LOOP-EQUIV-OP for two important memory domains will be given in Chapters 7 and 8. An algorithm for deciding whether a simple cycle is optimizable based on the predicate EXISTS-LOOP-EQUIV? is given in Figure 5.10. The correctness of this algorithm is a direct consequence of Definition 5.23.

The second problem is to integrate loop optimization together with cycle detection. Indeed, since loop optimization modifies the control graph and therefore its set of cycles, performing the optimization operation more than once requires to recompute the set of cycles of the control graph after each optimization. An easy way of carrying out this recomputation consists of updating after each optimization only the cycles that have been influenced by this optimization. The advantage is that

those cycles are usually in very small number and are quite easy to distinguish from the others, since each loop optimization only modifies a very small number of transitions in the control graph. Algorithms implementing repeated loop optimizations for SMBAs and SMAs are given in Figures 5.11, 5.12 and 5.13. Their correctness is established by the following results.

Theorem 5.26 *Let $\mathcal{B} = (C, c_0, M, m_0, Op, T, F)$ be a SMBA, S be a finite set of simple cycles in (C, T), and $\mathcal{B}'$ be the SMBA returned by OPTIMIZE-SET-SMBA($\mathcal{B}$, S). The language accepted by $\mathcal{B}$ is empty if and only if the language accepted by $\mathcal{B}'$ is empty.*

Theorem 5.27 *Let $\mathcal{A} = (C, c_0, M, m_0, Op, T)$ be an SMA, S be a finite set of simple cycles in its control graph (C, T), and $\mathcal{A}'$ be the SMA returned by the function call OPTIMIZE-SET-SMA($\mathcal{A}, S$). The sets Q_R and Q'_R of reachable states of $\mathcal{A}$ and $\mathcal{A}'$ (respectively) are such that for every $c \in C$, we have values(Q_R, c) = values(Q'_R, c).*

Proofs The SMA or SMBA $\mathcal{A}'$ is obtained from $\mathcal{A}$ by applying OPTIMIZE-CYCLE-SMBA (at Line 11 of OPTIMIZE-SET-SMBA) as many times as there are optimizable cycles in S. The purpose of the subsequent part of the main loop (Lines 12–29) is to update S so as to satisfy the invariant "every element of S is a simple cycle of (C, T)" at Line 11. We simply show that this modification of S is performed correctly. When a loop optimization is performed on the current state machine with the optimizable simple cycle $\mathcal{C} \in S$, the only elements of S that are not necessarily cycles in the control graph of the optimized state machine are those visiting $c_1 = \mathit{first}(\mathcal{C})$. This is a consequence of the way loop optimization is performed. The purpose of Line 13 is precisely to enumerate each cycle $\mathcal{C}'$ in S that needs to be updated. There are two possibilities:

- *After visiting c_1, $\mathcal{C}'$ follows the first transition of $\mathcal{C}$.* In this case, the update consists of transforming the subsequence of transitions

  $$(c, \theta, c_1), (c_1, \theta_1, c_2)$$

  in $\mathcal{C}'$, where $c, c_2 \in C$ and $\theta, \theta_1 \in Op$, into

  $$(c, \theta, c_a), (c_a, \mathrm{id}, c_1), (c_1, \theta_1, c_2),$$

  where c_a denotes the control location of the same name created during the loop optimization.

- *After visiting c_1, $\mathcal{C}'$ follows a transition that does not belong to $\mathcal{C}$.* Here, the update simply consists of transforming the control location c_1 into c_a in every transition of $\mathcal{C}'$ beginning or ending at that location.

In both cases, the updated cycle does correspond to $\mathcal{C}'$ in the control graph of the optimized machine. □

function OPTIMIZE-SET-SMBA(SMBA $(C, c_0, M, m_0, Op, T, F)$, set of simple cycles S) : SMBA

1: **var** $\mathcal{C}, \mathcal{C}'$: cycles;
2: $c_1, c, c', c'', c''', c_a, c'_a, c_1, c_2, c_3$: control locations;
3: $\theta, \theta', \theta'', \theta_a$: memory operations;
4: $(C', c'_0, M', m'_0, Op', T', F')$: SMBA;
5: σ, σ' : **sequences of** memory operations;
6: **begin**
7: **for each** $\mathcal{C} \in S$ **such that** OPTIMIZABLE?$((C, c_0, M, m_0, Op, T, F), \mathcal{C})$ **do**
8: **begin**
9: $c_1 := \mathit{first}(\mathcal{C})$;
10: **let** $(c, \theta, c') \in T \setminus \mathcal{C}$ **such that** $c = c_1$;
11: $(C', c'_0, M', m'_0, Op', T', F') :=$
OPTIMIZE-CYCLE-SMBA$((C, c_0, M, m_0, Op, T, F), \mathcal{C})$;
12: **let** $(c_a, \theta_a, c'_a) \in T' \setminus \mathcal{C}$ **such that** $c'_a = c_1$;
13: **for each** $\mathcal{C}' \in S$ **such that** $(\exists (c_2, \theta', c_3) \in \mathcal{C}')(c_2 = c_1)$ **do**
14: **if** $(c, \theta, c') \in \mathcal{C}'$ **then**
15: **begin**
16: $S := S \setminus \{\mathcal{C}'\}$;
17: **for each** $\sigma, \sigma' \in (T')^*, (c'', \theta'', c''') \in T'$ **such that**
$c''' = c_1 \wedge \mathcal{C}' = \sigma, (c'', \theta'', c'''), \sigma'$ **do**
18: $\mathcal{C}' := \sigma, (c'', \theta'', c_a), \sigma'$;
19: **let** $\sigma, \sigma' \in (T')^*$ **such that** $\mathcal{C}' = \sigma, (c, \theta, c'), \sigma'$;
20: $\mathcal{C}' := \sigma, (c_a, \theta, c'), \sigma'$;
21: $S := S \cup \{\mathcal{C}'\}$
22: **end**

(. . .)

Figure 5.11: Repeated loop optimizations for SMBAs.

```
    (...)
23:                          else
24:                              begin
25:                                  S := S \ {C'};
26:                                  for each σ,σ' ∈ (T')*, (c'',θ'',c''') ∈ T' such that
                                             c''' = c_1 ∧ C' = σ,(c'',θ'',c'''),σ' do
27:                                      C' := σ,(c'',θ'',c_a),(c_a,id,c_1),σ';
28:                                  S := S ∪ {C'}
29:                              end;
30:                  (C, c_0, M, m_0, Op, T, F) := (C', c'_0, M', m'_0, Op', T', F')
31:              end;
32:          return (C, c_0, M, m_0, Op, T, F)
33:      end.
```

Figure 5.12: Repeated loop optimizations for SMBAs (continued).

```
function OPTIMIZE-SET-SMA(SMA (C, c_0, M, m_0, Op, T), set of simple cycles S) : SMA
1:    var (C', c'_0, M', m'_0, Op', T', F') : SMBA;
2:    begin
3:        (C', c'_0, M', m'_0, Op', T', F') :=
              OPTIMIZE-SET-SMBA((C, c_0, M, m_0, Op, T, ∅), C);
4:        return (C', c'_0, M', m'_0, Op', T')
5:    end.
```

Figure 5.13: Repeated loop optimizations for SMAs.

Chapter 6

Finite-State Representation Systems

This chapter introduces a general technique for designing representation systems for sets of memory contents. The idea consists of encoding memory contents as words over some finite alphabet, and then representing a set as a finite automaton accepting the encodings of the contents of the set. This approach can easily be followed for a large number of domains, since memory contents admit most of the time natural encodings consisting of a string of bits. Moreover, we will show that the operations that one must be able to perform on representable sets of memory contents translate naturally into operations on automata.

The chapter is organized as follows. First, we introduce finite-state automata as well as a few associated notions. Then, we give algorithms for performing elementary operations on automata. Next, after defining the notion of encoding, we show that automata can be used as representations of sets of memory contents. Finally, we show that elementary operations on representable sets can be carried out by simply performing the corresponding operations on their representations.

6.1 Finite-State Automata

Intuitively, a finite-state automaton is a state machine recognizing a set of words by using only a finite amount of memory. Formally, we have the following definition.

Definition 6.1 *A* finite-state automaton *is a tuple* $(\Sigma, S, \Delta, I, F)$*, where*

- Σ *is a finite* alphabet*;*
- S *is a finite set of* states*;*
- $\Delta \subseteq S \times \Sigma^* \times S$ *is a* transition relation*. For each transition* $(s, w, s') \in \Delta$*,* s *is the* origin*,* s' *is the* end*, and* w *is the* label *of the transition;*

- $I \subseteq S$ *is a set of* initial states*;*
- $F \subseteq S$ *is a set of* accepting states*.*

Let $\mathcal{A} = (\Sigma, S, \Delta, I, F)$ be an automaton. Given two states $s, s' \in S$ and a word $w \in \Sigma^*$, we write $(s, w, s') \in \Delta^*$ if there exist $k \in \mathbf{N}$, $s_0, s_1, \ldots, s_k \in S$ and $w_0, w_1, \ldots, w_{k-1} \in \Sigma^*$ such that $s_0 = s$, $s_k = s'$, $w = w_0 w_1 \cdots w_{k-1}$, and $(s_i, w_i, s_{i+1}) \in \Delta$ for every $i \in \{0, 1, \ldots, k-1\}$. The sequence of transitions $(s_0, w_0, s_1), (s_1, w_1, s_2), \ldots, (s_{k-1}, w_{k-1}, s_k)$ is a *path* leading from s to s', labeled by w. The word w is *accepted* by $\mathcal{A}$ if there exists a path labeled by w leading from a state of I to a state of F. The set of all the words accepted by $\mathcal{A}$ is the *language* accepted by $\mathcal{A}$ and is denoted $L(\mathcal{A})$. If the label of every transition of $\mathcal{A}$ has a length less or equal to one, then $\mathcal{A}$ is said to be in *normal form*. If the label of every transition of $\mathcal{A}$ has a length exactly equal to one, then $\mathcal{A}$ is said to be in *strong normal form*. If $\mathcal{A}$ has at most one initial state, and does not have two transitions $(s_1, w_1, s_1'), (s_2, w_2, s_2')$ such that $s_1 = s_2$ and either $w_1 \in \mathrm{pre}(w_2)$ or $w_2 \in \mathrm{pre}(w_1)$, where $\mathrm{pre}(w)$ denotes the set of the prefixes of the word $w \in \Sigma^*$, then $\mathcal{A}$ is said to be *deterministic*. Intuitively, $\mathcal{A}$ is deterministic if for every state $s \in S$ and word $w \in \Sigma^*$, there is at most one transition in Δ labeled by a prefix of w that has the origin s. The following theorem is a well-known result of automata theory:

Theorem 6.2 *For every automaton $\mathcal{A}$, there exists a deterministic automaton $\mathcal{A}'$ in strong normal form such that $L(\mathcal{A}) = L(\mathcal{A}')$.*

Proof A constructive proof can be found in [HU79] or [Per90]. An algorithm for computing $\mathcal{A}'$ given $\mathcal{A}$ will be presented in Section 6.2. □

Theorem 6.2 implies that the languages that can be accepted by finite-state automata do not depend on the deterministic nature of these automata. Such languages can easily be described by simple formulas called *regular expressions*. We have the following definition, inspired by [HU79]:

Definition 6.3 *Let Σ be a finite alphabet. The regular expressions over Σ and the subsets of Σ^* that they denote are defined recursively as follows:*

- *$\emptyset$ is a regular expression and denotes the empty language $\emptyset$;*
- *ε is a regular expression and denotes the language $\{\varepsilon\}$ whose only element is the empty word ε;*
- *For each $a \in \Sigma$, a is a regular expression and denotes the language $\{a\}$;*
- *If e_1 is a regular expression denoting the language $L_1 \in \Sigma^*$, then (e_1^*) is a regular expression and denotes the language*

$$L_1^* = \{w \in \Sigma^* \mid (\exists k \in \mathbf{N}, w_1, w_2, \ldots, w_k \in L_1)(w = w_1 w_2 \cdots w_k)\}.$$

This language is called the Kleene closure *of L_1;*

- *If e_1 and e_2 are regular expressions denoting respectively the languages $L_1, L_2 \in \Sigma^*$, then*
  - *$(e_1 + e_2)$ is a regular expression and denotes the language $L_1 \cup L_2$;*
  - *$(e_1 \cdot e_2)$ is a regular expression and denotes the language*

$$L_1 \cdot L_2 = \{w_1 w_2 \mid w_1 \in L_1 \wedge w_2 \in L_2\}.$$

This language is called the concatenation *of L_1 and L_2.*

In writing regular expressions, one may delete many pairs of parentheses by assuming that "$*$" has an higher precedence than "$\cdot$" and "$+$", and that "$\cdot$" has an higher precedence than "$+$". If e is a regular expression, then e^+ is a shorthand for $e \cdot e^*$.

Any language that can be denoted by a regular expression is said to be *regular*. The following theorem states that regular languages are exactly those than can be accepted by finite-state automata.

Theorem 6.4 *Let Σ be a finite alphabet, and $L \subseteq \Sigma^*$ be a language. There exists a finite-state automaton accepting L if and only if there exists a regular expression over Σ denoting L.*

Proof A constructive proof can be found in [HU79] or [Per90]. □

Because of the equivalence between regular languages and regular expressions, we will often use the same notation for denoting both. For instance, if L_1 and L_2 are languages, then $(L_1 \cdot L_2)^*$ will be used as a shorthand for "the language denoted by the regular expression $(E_1 \cdot E_2)^*$, where E_1 and E_2 are regular expressions denoting respectively L_1 and L_2".

6.2 Operations on Automata

An advantage of automata over other representations of regular languages is that it is easy to write algorithms for manipulating automata. In this section, we define some operations that can be performed on automata, and give algorithms for carrying out these operations. The algorithms are not fully described in this thesis. They are only included for completeness, since they will be used extensively in the sequel. Detailed descriptions of those algorithms can be found in [HU79], [Per90] and [Hop71].

6.2.1 Determinization

The goal of the determinization operation is to compute, given an automaton $\mathcal{A}$, an automaton DETERMINIZE($\mathcal{A}$) which is complete, in strong normal form, and

```
function NORMALIZE(automaton (Σ, S, Δ, I, F)) : automaton
 1:  var (s, w, s') : transition;
 2:      s_1, s_2, ... : states;
 3:  begin
 4:      for each (s, w, s') ∈ Δ such that |w| > 1 do
 5:          begin
 6:              let s_1, s_2, ..., s_{|w|-1} ∉ S;
 7:              S := S ∪ {s_1, s_2, ..., s_{|w|-1}};
 8:              Δ := (Δ \ {(s, w, s')}) ∪ {(s, w[1], s_1), (s_{|w|-1}, w[|w|], s')}
                                          ∪ {(s_{i-1}, w[i], s_i) | 1 < i < |w|}
 9:          end;
10:      return (Σ, S, Δ, I, F)
11:  end.
```

Figure 6.1: Normalization of an automaton.

accepts exactly the same language as $\mathcal{A}$. The determinization operation proceeds by first normalizing $\mathcal{A}$, i.e., converting it into an automaton NORMALIZE($\mathcal{A}$) that accepts the same language, but is in normal form. Normalizing an automaton can be done by replacing each transition whose label has a length greater than one by a succession of transitions labeled by a single symbol, creating as many intermediate new states as necessary. An algorithm[1] implementing this operation is given in Figure 6.1.

After normalizing the input automaton $\mathcal{A}$, obtaining an automaton $\mathcal{A}'$, the determinization procedure creates an automaton $\mathcal{A}''$ whose states correspond to subsets of states of $\mathcal{A}'$. The set of initial states of $\mathcal{A}''$ contains only one element, corresponding to the set of all the states of $\mathcal{A}'$ that can be reached without reading any symbol (in other words, all the states that can be reached by reading the empty word). Whenever it is possible to go from a set Q of states of $\mathcal{A}'$ to another such set Q' by reading a symbol a, one creates a transition of $\mathcal{A}''$ labeled by a with its origin and end corresponding respectively to Q and to Q'. The creation of transitions can be seen as the exploration of a finite graph and can thus be performed according to a depth-first strategy. Finally, a state of $\mathcal{A}''$ is accepting if and only if it corresponds to a set of states of $\mathcal{A}'$ containing at least one accepting state. Following those rules, one eventually obtains an automaton $\mathcal{A}''$ which is deterministic and in strong nor-

[1] In this algorithm, the notations $|w|$ and $w[i]$ denote respectively the length of the word w, i.e., the number of symbols composing w, and the i-th symbol of w ($1 \leq i \leq |w|$).

mal form (by construction), and such that $L(\mathcal{A}'') = L(\mathcal{A})$. An algorithm formalizing this determinization procedure is given in Figure 6.2. The time and space cost of determinizing an automaton can be as high as $O(2^{|\Delta|})$, where $|\Delta|$ denotes the sum of the lengths of the labels of all the transitions.

6.2.2 Minimization

The purpose of the minimization operation is to compute, given an automaton $\mathcal{A}$ that is deterministic and in strong normal form, the smallest automaton (with respect to the number of states) that accepts the same language as $\mathcal{A}$ and is deterministic and in strong normal form as well (this automaton is denoted MINIMIZE($\mathcal{A}$)). According to a well-known result [Har65, McC65, HU79], MINIMIZE($\mathcal{A}$) always exists and is unique up to isomorphism.

An efficient algorithm for computing MINIMIZE($\mathcal{A}$) has been given by Hopcroft [Hop71]. This algorithm can be found in Figures 6.3 and 6.4. It proceeds by first partitioning the states of $\mathcal{A}$ according to the coarsest equivalence relation such that for every equivalent states s, s' of $\mathcal{A}$ and symbol a, if s and s' both have outgoing transitions labeled by a, then the ends of those transitions belong to the same equivalence class. The automaton MINIMIZE($\mathcal{A}$) is then obtained by creating one state for each equivalence class, and linking those states by the same transitions as those linking the elements of the equivalence classes to each other. A complete description of the algorithm as well as a proof of its correctness can be found in [Hop71]. The time and space cost of the algorithm is $O(|S| \log |S|)$, where $|S|$ denotes the number of states of $\mathcal{A}$.

Algorithms have also been developed for minimizing transition systems with respect to finer equivalence relations than language equivalence [PT87, BFH91]. We do not describe them in this thesis.

6.2.3 Closure and Concatenation

The goal of the closure operation is to compute an automaton $\mathcal{A}'$ accepting the language L^*, given an automaton $\mathcal{A}$ accepting the language L. This is done by building $\mathcal{A}'$ in such a way that each of its accepting paths corresponds to a succession of accepting paths of $\mathcal{A}$. The construction is illustrated in Figure 6.5. An algorithm implementing the closure operation is given in Figure 6.6. The time and space cost of this operation is $O(|I| + |F|)$, where $|I|$ and $|F|$ denote respectively the number of initial and of accepting states of $\mathcal{A}$.

The concatenation operation aims at computing, given two automata $\mathcal{A}_1$ and $\mathcal{A}_2$, an automaton $\mathcal{A}'$ accepting the language $L(\mathcal{A}_1) \cdot L(\mathcal{A}_2)$. It is performed by constructing $\mathcal{A}'$ in such a way that each of its accepting paths corresponds to an accepting path of $\mathcal{A}_1$ followed by an accepting path of $\mathcal{A}_2$. The construction is

```
function DETERMINIZE(automaton 𝒜) : automaton
 1:  var (Σ, S, Δ, I, F) (Σ′, S′, Δ′, I′, F′) : automata;
 2:      Q_0 : set of states;
 3:  procedure generate(set of states Q)
 4:      var Q′ : set of states;
 5:          a : symbol;
 6:      begin
 7:          S′ := S′ ∪ {Q};
 8:          for each a ∈ Σ such that (∃(s, a′, s′) ∈ Δ*)(s ∈ Q ∧ a′ = a) do
 9:              begin
10:                  Q′ := {s′ ∈ S | (∃s ∈ Q)((s, a, s′) ∈ Δ*)};
11:                  Δ′ := Δ′ ∪ {(Q, a, Q′)};
12:                  if Q′ ∉ S′ then generate(Q′)
13:              end
14:      end;
15:  begin
16:      (Σ, S, Δ, I, F) := NORMALIZE(𝒜);
17:      Σ′ := Σ;
18:      S′ := ∅;
19:      Δ′ := ∅;
20:      Q_0 := {s ∈ S | (∃s_0 ∈ I)((s_0, ε, s) ∈ Δ*)};
21:      generate(Q_0);
22:      I′ := {Q_0};
23:      F′ := {Q ∈ S′ | Q ∩ F ≠ ∅};
24:      return (Σ′, S′, Δ′, I′, F′)
25:  end.
```

Figure 6.2: Determinization of an automaton.

```
function MINIMIZE(deterministic automaton (Σ, S, Δ, I, F)) : deterministic automaton
 1:   var (Σ', S', Δ', I', F') : automaton;
 2:       s'' : state;
 3:   function partition(automaton (Σ, S, Δ, I, F)) : set of sets of states
 4:       var class : array[1, 2, ...] of sets of states;
 5:           image-class : array[1, 2, ...; Σ] of sets of states;
 6:           part1, part2 : sets of states;
 7:           split-list : array[Σ] of sets of integers;
 8:           i, j, k, l : integers;
 9:           a : symbol;
10:       begin
11:           class[1] := F;
12:           class[2] := S \ F;
13:           for each a ∈ Σ, i ∈ {1, 2} do
14:               image-class[i, a] := {s ∈ class[i] | (∃(s1, a1, s2) ∈ Δ)
                                                    (s2 = s ∧ a1 = a)};
15:           for each a ∈ Σ do
16:               if |image-class[1, a]| ≤ |image-class[2, a]| then split-list[a] := {1}
17:               else split-list[a] := {2};
18:           k := 3;
19:           while (∃a ∈ Σ)(split-list[a] ≠ ∅) do
20:               begin
21:                   let a ∈ Σ such that split-list[a] ≠ ∅;
22:                   let i ∈ split-list[a];
23:                   split-list[a] := split-list[a] \ {i};
24:                   for each j such that 1 ≤ j < k ∧ (∃s ∈ class[j],
                       (s1, a', s2) ∈ Δ)(s1 = s ∧ a' = a ∧ s2 ∈ image-class[i, a]) do
25:                       begin
26:                           part1 := {s ∈ S | (∃(s1, a', s2) ∈ Δ)(s1 = s
                                            ∧ a' = a ∧ s2 ∈ image-class[i, a])};

    (...)
```

Figure 6.3: Minimization of a deterministic automaton.

```
    (...)
27:                                    part2 := class[j] \ part1;
28:                                    class[j] := part1;
29:                                    class[k] := part2;
30:                                    for each a ∈ Σ, l ∈ {j, k} do
31:                                        image-class[l, a] := {s ∈ class[l] |
                                               (∃(s1, a1, s2) ∈ Δ)(s2 = s ∧ a1 = a)};
32:                                    for each a ∈ Σ do
33:                                        if j ∉ split-list[a] ∧ |image-class[j, a]|
                                                   ≤ |image-class[k, a]| then
34:                                            split-list[a] := split-list[a] ∪ {j}
35:                                        else split-list[a] := split-list[a] ∪ {k};
36:                                    k := k + 1
37:                                end
38:                        end;
39:                    return {class[l] | 1 ≤ l < k ∧ class[l] ≠ ∅}
40:            end;
41:    begin
42:        S := {s ∈ S | (∃s0 ∈ I, w ∈ Σ*)((s0, w, s) ∈ Δ*)};
43:        let s'' ∉ S;
44:        S := S ∪ {s''};
45:        for each s1 ∈ S, a ∈ Σ such that (∄(s', a', s''') ∈ Δ)(s' = s1 ∧ a' = a) do
46:            Δ := Δ ∪ {(s1, a, s'')};
47:        Σ' := Σ;
48:        S' := partition((Σ, S, Δ, I, F));
49:        Δ' := {(Q, a, Q') ∈ S' × Σ' × S' | (∃s ∈ Q, s ∈ Q', a ∈ Σ)((s, a, s') ∈ Δ)};
50:        I' := {Q ∈ S' | I ⊆ Q};
51:        F' := {Q ∈ S' | Q ∩ F ≠ ∅};
52:        return (Σ', S', Δ', I', F')
53:    end.
```

Figure 6.4: Minimization of a deterministic automaton (continued).

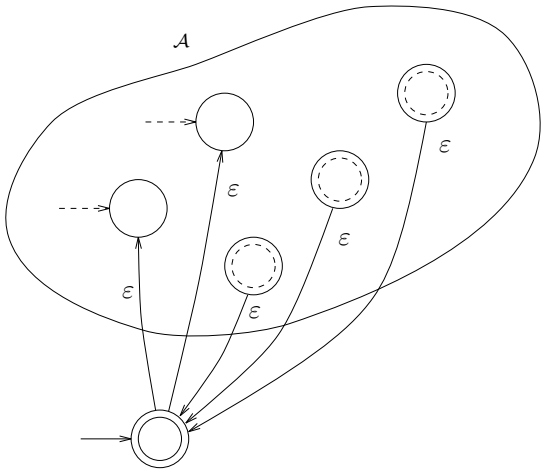

Figure 6.5: Computing the closure of an automaton.

---

function CLOSURE(automaton $(\Sigma, S, \Delta, I, F)$) : automaton

```
1:  var s : state;
2:  begin
3:      let s ∉ S;
4:      S := S ∪ {s};
5:      Δ := Δ ∪ {(s, ε, s') | s' ∈ I} ∪ {(s', ε, s) | s' ∈ F};
6:      return (Σ, S, Δ, {s}, {s})
7:  end.
```

Figure 6.6: Closure of an automaton.

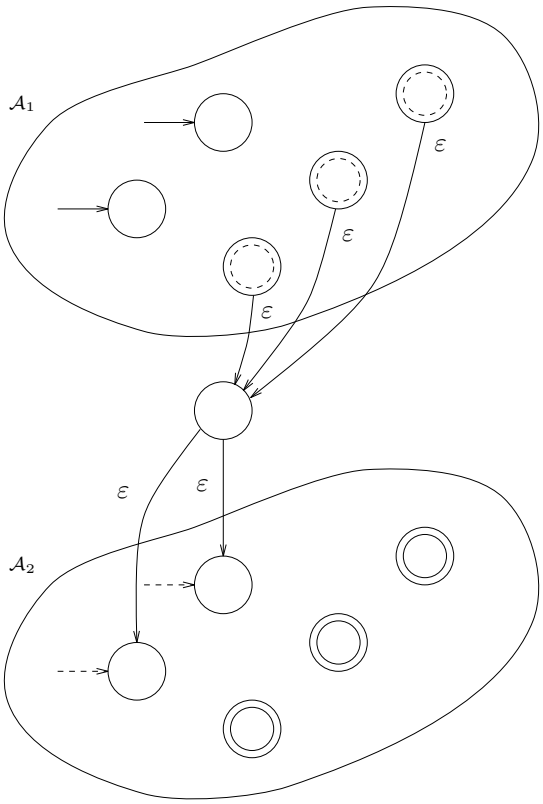

Figure 6.7: Concatenating two automata.

illustrated in Figure 6.7. An algorithm implementing the concatenation operation is given in Figure 6.8. Its time and space cost of is $O(|F_1| + |I_2|)$, where $|F_1|$ and $|I_2|$ denote respectively the number of accepting states of $\mathcal{A}_1$ and the number of initial states of $\mathcal{A}_2$.

6.2.4 Set-Theory Operators

The *synchronous product* operation takes as arguments two automata $\mathcal{A}_1$ and $\mathcal{A}_2$ of respective alphabets Σ_1 and Σ_2, in normal form, and computes an automaton $\mathcal{A}'$ accepting the language

$$\{(a_1, a'_1)(a_2, a'_2) \cdots (a_k, a'_k) \mid k \in \mathbf{N} \wedge a_1 a_2 \cdots a_k \in L(\mathcal{A}_1) \wedge a'_1 a'_2 \cdots a'_k \in L(\mathcal{A}_2)\}.$$

It is performed by constructing an automaton whose set of states is the Cartesian product of the sets of states of $\mathcal{A}_1$ and of $\mathcal{A}_2$. Each transition of $\mathcal{A}'$ labeled by a pair of symbols $(a, a') \in \Sigma_1 \times \Sigma_2$ corresponds to a transition of $\mathcal{A}_1$ labeled by a and a transition of $\mathcal{A}_2$ labeled by a' followed simultaneously. Each transition of $\mathcal{A}'$ labeled by ε corresponds either to a transition of $\mathcal{A}_1$ labeled by ε, or to a transition of $\mathcal{A}_2$ labeled by ε. An algorithm implementing this operation is given in Figure 6.9. The time and space cost of this algorithm is $O(|\Delta_1||\Delta_2|)$.

function CONCATENATE(automata $(\Sigma_1, S_1, \Delta_1, I_1, F_1)$, $(\Sigma_2, S_2, \Delta_2, I_2, F_2)$
such that $S_1 \cap S_2 = \emptyset$) : automaton

1: **var** $(\Sigma, S, \Delta, I, F)$: automaton;
2: s : state;
3: **begin**
4: **let** $s \notin S_1 \cup S_2$;
5: $\Sigma := \Sigma_1 \cup \Sigma_2$;
6: $S := S_1 \cup S_2 \cup \{s\}$;
7: $\Delta := \Delta \cup \{(s', \varepsilon, s) \mid s' \in F_1\} \cup \{(s, \varepsilon, s') \mid s' \in I_2\}$;
8: $I := I_1$;
9: $F := F_2$;
10: **return** $(\Sigma, S, \Delta, I, F)$
11: **end**.

Figure 6.8: Concatenation of two automata.

function PRODUCT(automata $(\Sigma_1, S_1, \Delta_1, I_1, F_1)$, $(\Sigma_2, S_2, \Delta_2, I_2, F_2)$) : automaton

1: **var** $(\Sigma, S, \Delta, I, F)$: automaton;
2: **begin**
3: $(\Sigma_1, S_1, \Delta_1, I_1, F_1) :=$ NORMALIZE$((\Sigma_1, S_1, \Delta_1, I_1, F_1))$;
4: $(\Sigma_2, S_2, \Delta_2, I_2, F_2) :=$ NORMALIZE$((\Sigma_2, S_2, \Delta_2, I_2, F_2))$;
5: $\Sigma := \Sigma_1 \times \Sigma_2$;
6: $S := S_1 \times S_2$;
7: $\Delta := \{((s_1, s_2), (a_1, a_2), (s_1', s_2')) \in S \times \Sigma \times S \mid (s_1, a_1, s_1') \in \Delta_1$
$\wedge\, (s_2, a_2, s_2') \in \Delta_2\}$
$\cup \{((s_1, s_2), \varepsilon, (s_1', s_2')) \in S \times \{\varepsilon\} \times S \mid (s_1 = s_1' \wedge (s_2, \varepsilon, s_2') \in \Delta_2)$
$\vee\, (s_2 = s_2' \wedge (s_1, \varepsilon, s_1') \in \Delta_1)\}$;
8: $I := I_1 \times I_2$;
9: $F := F_1 \times F_2$;
10: **return** $(\Sigma, S, \Delta, I, F)$
11: **end**.

Figure 6.9: Synchronous product of two automata.

```
function INTERSECTION(automata (Σ1, S1, Δ1, I1, F1), (Σ2, S2, Δ2, I2, F2)) : automaton
 1:  var (Σ, S, Δ, I, F) : automaton;
 2:      s, s' : states;
 3:      a1, a2 : symbols;
 4:  begin
 5:      (Σ, S, Δ, I, F) := PRODUCT((Σ1, S1, Δ1, I1, F1), (Σ2, S2, Δ2, I2, F2));
 6:      for each (s, (a1, a2), s') ∈ Δ do
 7:          if a1 = a2 then
 8:              Δ := (Δ \ {(s, (a1, a2), s')}) ∪ {(s, a1, s')}
 9:          else
10:              Δ := Δ \ {(s, (a1, a2), s')};
11:      Σ := Σ1 ∩ Σ2;
12:      return (Σ, S, Δ, I, F)
13:  end.
```

Figure 6.10: Intersection of two automata.

The goal of the *intersection* operation is to compute, given two automata $\mathcal{A}_1$ and $\mathcal{A}_2$ of respective alphabets Σ_1 and Σ_2, an automaton $\mathcal{A}'$ accepting the language $L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$. This operation is performed by first computing the synchronous product $\mathcal{A}$ of $\mathcal{A}_1$ and $\mathcal{A}_2$, and then deleting from the set of transitions of $\mathcal{A}$ all the transitions labeled by a pair $(a, a') \in \Sigma_1 \times \Sigma_2$ such that $a \neq a'$. Each accepting path of the resulting automaton thus corresponds to an accepting path of $\mathcal{A}_1$ and an accepting path of $\mathcal{A}_2$ that both read the same word. An algorithm implementing this operation is given in Figure 6.10. The time and space cost of this algorithm is $O(|\Delta_1||\Delta_2|)$.

The *union* operation consists of computing, given two automata $\mathcal{A}_1$ and $\mathcal{A}_2$, an automaton $\mathcal{A}'$ accepting the language $L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$. It is performed by constructing $\mathcal{A}'$ in such a way that each of its accepting paths corresponds either to an accepting path of $\mathcal{A}_1$ or to an accepting path of $\mathcal{A}_2$. The construction is illustrated in Figure 6.11. An algorithm implementing the union operation is given in Figure 6.12. The time and space cost of this operation is $O(|I_1| + |I_2|)$.

The *complement* operation consists of computing an automaton $\mathcal{A}'$ accepting the complement $\overline{L(\mathcal{A})}$ of the language accepted by an automaton $\mathcal{A}$ over its alphabet. This operation is performed by first determinizing $\mathcal{A}$ and then completing its set of transitions so as to have at each state and for every symbol of the alphabet an outgoing transition labeled by that symbol. The complemented automaton is

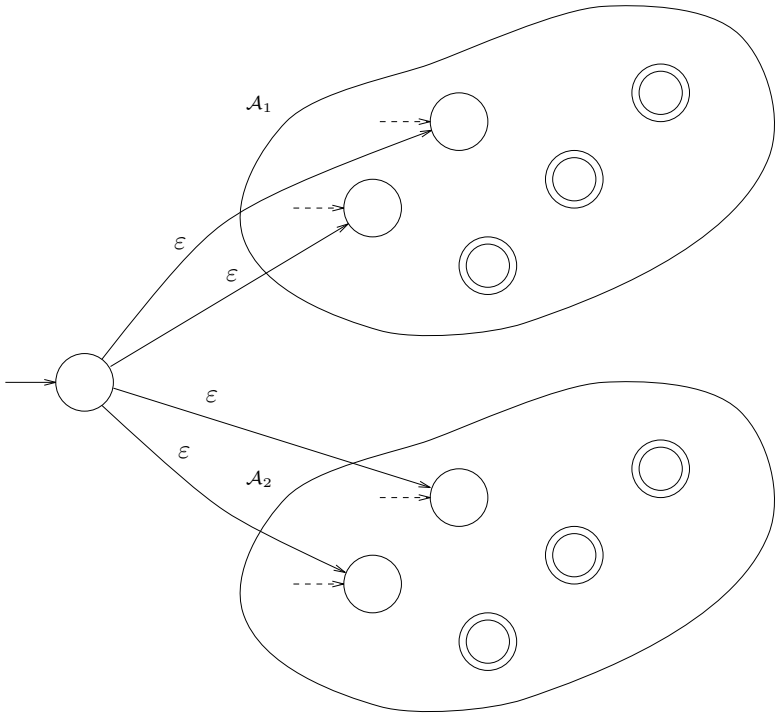

Figure 6.11: Computing the union of two automata.

---

function UNION(automata $(\Sigma_1, S_1, \Delta_1, I_1, F_1)$, $(\Sigma_2, S_2, \Delta_2, I_2, F_2)$
such that $S_1 \cap S_2 = \emptyset$) : automaton

1: **var** $(\Sigma, S, \Delta, I, F)$: automaton;
2: s : state;
3: **begin**
4: **let** $s \notin S_1 \cup S_2$;
5: $\Sigma := \Sigma_1 \cup \Sigma_2$;
6: $S := S_1 \cup S_2 \cup \{s\}$;
7: $\Delta := \Delta \cup \{(s, \varepsilon, s') \mid s' \in I_1 \cup I_2\}$;
8: $I := \{s\}$;
9: $F := F_1 \cup F_2$;
10: **return** $(\Sigma, S, \Delta, I, F)$
11: **end**.

Figure 6.12: Union of two automata.

```
function COMPLEMENT(automaton A) : automaton
 1:  var (Σ, S, Δ, I, F) : automaton;
 2:      s, s_1 : states;
 3:      a : symbol;
 4:  begin
 5:      (Σ, S, Δ, I, F) := DETERMINIZE(A);
 6:      let s ∉ S;
 7:      S := S ∪ {s};
 8:      for each s_1 ∈ S, a ∈ Σ such that (∄(s', a', s'') ∈ Δ)(s' = s_1 ∧ a' = a) do
 9:          Δ := Δ ∪ {(s_1, a, s)};
10:      F := S \ F;
11:      return (Σ, S, Δ, I, F)
12:  end.
```

Figure 6.13: Complement of an automaton.

then obtained by exchanging the accepting and non-accepting states. An algorithm implementing this operation is given in Figure 6.13. Its time and space cost is $O(2^{|\Delta|})$ if the automaton is non-deterministic, and $O(|S|)$ if it is deterministic.

The *difference* operation consists of computing, given two automata $\mathcal{A}_1$ and $\mathcal{A}_2$, an automaton accepting the language $L(\mathcal{A}_1) \setminus L(\mathcal{A}_2)$. This operation can simply be performed by computing the intersection of $\mathcal{A}_1$ and of the complement of $\mathcal{A}_2$. An algorithm implementing this operation is given in Figure 6.14. Its time and space cost is $O(|\Delta_1|2^{|\Delta_2|})$ if $\mathcal{A}_2$ is non-deterministic, and $O(|\Delta_1||\Delta_2|)$ is $\mathcal{A}_2$ is deterministic.

The next operation is to test whether the language accepted by an automaton is empty or not. This is equivalent to checking whether the automaton admits at

```
function DIFFERENCE(automata A_1, A_2) : automaton
1:  begin
2:      return INTERSECTION(A_1, COMPLEMENT(A_2))
3:  end.
```

Figure 6.14: Difference between two automata.

```
function EMPTY?(automaton (Σ, S, Δ, I, F)) : {T, F}
 1:  var states : set of states;
 2:      s : state;
 3:  function accepting-path(state s') : {T, F}
 4:      var (s1, a, s2) : transition;
 5:      begin
 6:          if s' ∈ F do return T;
 7:          states := S ∪ {s'};
 8:          for each (s1, a, s2) ∈ Δ such that s1 = s' ∧ s2 ∉ states do
 9:              if accepting-path(s2) then return T;
10:          return F
11:      end;
12:  begin
13:      states := ∅;
14:      for each s ∈ I do
15:          if accepting-path(s) then return F;
16:      return T
17:  end.
```

Figure 6.15: Test of emptiness of the language accepted by an automaton.

least one accepting path, which can simply be done by performing a depth-first search for an accepting state. An algorithm implementing this operation is given in Figure 6.15. Its time and space cost is $O(|\Delta|)$.

Let us now address the problem that consists of testing whether the language accepted by an automaton $\mathcal{A}_1$ is included in the language accepted by an automaton $\mathcal{A}_2$. This test can be carried out by simply checking the emptiness of the difference between $\mathcal{A}_1$ and $\mathcal{A}_2$. An algorithm implementing this operation is given in Figure 6.16. Its time and space cost is $O(|\Delta_1|2^{|\Delta_2|})$ if $\mathcal{A}_2$ is non-deterministic, and $O(|\Delta_1||\Delta_2|)$ is $\mathcal{A}_2$ is deterministic.

The last operation studied in this section is the application of an homomorphism. Let Σ be an alphabet. An *homomorphism* over Σ is a function $f : \Sigma^* \to \Sigma^*$ such that for any two words $w_1, w_2 \in \Sigma^*$, $f(w_1 \cdot w_2) = f(w_1) \cdot f(w_2)$ (in other words, an homomorphism is a function that is distributive over the concatenation of words). It follows from this definition that an homomorphism f can simply be defined by the mapping $\{(a, f(a)) \mid a \in \Sigma\}$ of the symbols of the alphabet. Applying

```
function INCLUDED?(automata A_1, A_2) : {T, F}
    1:  begin
    2:      return EMPTY?(DIFFERENCE(A_1, A_2))
    3:  end.
```

Figure 6.16: Test of inclusion between two languages accepted by automata.

an homomorphism f to an automaton $\mathcal{A}$ consists of computing an automaton $\mathcal{A}'$ accepting $\{f(w) \mid w \in L(\mathcal{A})\}$. This can be done by simply applying the homomorphism to each transition label of $\mathcal{A}$. An algorithm implementing this operation is given in Figure 6.17. In particular, this algorithm allows to compute an automaton accepting the *projection* $L(\mathcal{A})|_{\Sigma'}$ of the language accepted by a given automaton $\mathcal{A} = (\Sigma, S, \Delta, I, F)$ over a subset Σ' of its alphabet by applying to $\mathcal{A}$ the homomorphism $f_{\Sigma'}$ such that for every $a \in \Sigma$, $f_{\Sigma'}(a) = a$ if $a \in \Sigma'$ and $f_{\Sigma'}(a) = \varepsilon$ if $a \notin \Sigma'$. The time and space cost of applying an homomorphism to an automaton is $O(C|\Delta|)$, where C is the cost of computing the image of a transition label by the homomorphism.

6.3 Automata as Representations of Sets

Let M be a memory domain and Σ be a finite alphabet. An *encoding scheme* for the elements of M over Σ is a relation that associates to each memory content $m \in M$ one or several words over Σ such that each of them describes unambiguously m. Formally, we have the following definition.

Definition 6.5 *An* encoding scheme *for the elements of M over Σ is a relation $E \subseteq M \times V$, where $V \subseteq \Sigma^*$ is a set of* valid encodings, *such that*

- *For each $m \in M$, there exists $(m_1, w_1) \in E$ such that $m_1 = m$ (the relation is complete over M), and*
- *For each $w \in V$, there exists exactly one $(m_1, w_1) \in E$ such that $w_1 = w$ (the relation is complete and unambiguous over V).*

Since encoding schemes associate words to memory contents, they can be used for transforming a set of memory contents into a language. Given a memory domain M, an alphabet Σ and an encoding scheme E, we define the *encoding* $E(U)$ of a set

```
function APPLY-HOMOMORPHISM(automaton (Σ, S, Δ, I, F), function f) : automaton
1:  begin
2:      Δ := {(s1, f(w), s2) | (s1, w, s2) ∈ Δ};
3:      return (Σ, S, Δ, I, F)
4:  end.
```

Figure 6.17: Application of an homomorphism to an automaton.

of memory contents $U \subseteq M$ as the language[2]

$$E(U) = \{w \in \Sigma^* \mid (\exists m \in U, (m_1, w_1) \in E)(m_1 = m \wedge w_1 = w)\}.$$

Reciprocally, given a language $L \subseteq \Sigma^*$, we define the *decoding* $D(L)$ of L as the set of memory contents

$$D(L) = \{m \in M \mid (\exists w \in L, (m_1, w_1) \in E)(m_1 = m \wedge w_1 = w)\}.$$

As a consequence of the requirements of Definition 6.5, we have $D(E(U)) = U$ for every set of memory contents $U \subseteq M$. This shows that the encoding of a set of memory contents describes unambiguously this set.

The next step is to represent the encoding of a set of memory contents, i.e., to associate to such a set a finite object containing sufficient information for describing this set unambiguously. We have the following definition.

Definition 6.6 *Let M be a memory domain, Σ be a finite alphabet, and E be an encoding scheme for the elements of M over Σ. A* finite-state representation *of a set $U \subseteq M$ of memory contents with respect to E is a finite-state automaton $\mathcal{A}$ of alphabet Σ, such that $L(\mathcal{A}) = E(U)$.*

If M is infinite, then it is not possible to represent each of its subsets. The reason is that there are uncountably many subsets of memory contents but only countably many finite-state representations. Following Theorem 6.4, the next definition characterizes the sets of memory contents that have a finite-state representation.

Definition 6.7 *A set of memory contents $U \subseteq M$ is* recognizable *with respect to an encoding scheme E if the language $E(U)$ is regular.*

The advantages of finite-state representations over other representation systems are that they are quite expressive, i.e., they allow to represent a large class of sets,

[2]By extension, if $m \in M$ and $w \in E(\{m\})$, then w is called an *encoding* of m.

and that there are a lot of useful operations that can be performed on finite-state automata. In the next section, we study how to reduce operations on representable sets of memory contents to the operations on finite-state automata presented in Section 6.2. The problem of characterizing precisely the expressiveness of finite-state representations will be addressed for two particular memory domains in Chapters 7 and 8.

6.4 Operations on Representable Sets

Thanks to the following theorem, applying set-theory operators over recognizable sets of memory contents can simply be done by performing the corresponding operations over their finite-state representation.

Theorem 6.8 *Let M be a memory domain, E be an encoding scheme and $\theta \in \{\cup, \cap, \setminus\}$ be a set-theory operation. For every recognizable sets $U_1, U_2 \subseteq M$ of memory contents, we have $E(U_1 \,\theta\, U_1) = E(U_1) \,\theta\, E(U_2)$.*

Proof

- *We have* $E(U_1 \cup U_2) \subseteq E(U_1) \cup E(U_2)$. If $w \in E(U_1 \cup U_2)$, then there exist $m \in U_1 \cup U_2$ and $(m_1, w_1) \in E$ such that $m_1 = m \wedge w_1 = w$. If $m \in U_1$, then $w \in E(U_1)$. If $m \in U_2$, then $w \in E(U_2)$.

- *We have* $E(U_1) \cup E(U_2) \subseteq E(U_1 \cup U_2)$. If $w \in E(U_1) \cup E(U_2)$, there there exists $(m_1, w_1) \in E$ such that $w_1 = w$ and m_1 belongs to $U_1 \cup U_2$. Thus, $w \in E(U_1 \cup U_2)$.

- *We have* $E(U_1 \cap U_2) \subseteq E(U_1) \cap E(U_2)$. If $w \in E(U_1 \cap U_2)$, then there exist $m \in U_1 \cap U_2$ and $(m_1, w_1) \in E$ such that $m_1 = m \wedge w_1 = w$. Since $m \in U_1$ and $m \in U_2$, it follows that $w \in E(U_1) \cap E(U_2)$.

- *We have* $E(U_1) \cap E(U_2) \subseteq E(U_1 \cap U_2)$. If $w \in E(U_1) \cap E(U_2)$, there there exist $(m_1, w_1), (m_2, w_2) \in E$ such that $w_1 = w_2 = w$, $m_1 \in U_1$, and $m_2 \in U_2$. Since $w \in E(U_1)$, we have $E(\{m_2\}) \subseteq E(U_1)$, from which we deduce $m_2 \in U_1$. Since (m_2, w_2) belongs to E and is such that $m_2 \in U_1 \cap U_2$ and $w_2 = w$, it follows that $w \in E(U_1 \cap U_2)$.

- *We have* $E(U_1 \setminus U_2) \subseteq E(U_1) \setminus E(U_2)$. If $w \in E(U_1 \setminus U_2)$, then there exists $(m_1, w_1) \in E$ such that $w_1 = w$, $m_1 \in U_1$, and $m_1 \notin U_2$. It follows that $E(\{m_1\}) \subseteq E(U_1)$ and $E(\{m_2\}) \cap E(U_2) = \emptyset$. Since $w \in E(\{m_1\})$, this implies $w \in E(U_1) \setminus E(U_2)$.

- *We have* $E(U_1) \setminus E(U_2) \subseteq E(U_1 \setminus U_2)$. If $w \in E(U_1) \setminus E(U_2)$, then there exists $(m_1, w_1) \in E$ such that $w_1 = w$ and $m_1 \in U_1$, and there does not exist $(m_2, w_2) \in E$ such that $w_2 = w$ and $m_2 \in U_2$. It follows that $m_1 \notin U_2$, hence that $m_1 \in U_1 \setminus U_2$, which implies $w \in E(U_1 \setminus U_2)$.

□

Similarly, testing the emptiness of a set or the inclusion of a set into another can also be done by performing the corresponding operation on the finite-state representations of the sets. This is expressed by the following theorem.

Theorem 6.9 *Let M be a memory domain and E be an encoding scheme whose set of valid encodings V is not empty. For every recognizable set $U_1 \subseteq M$, U_1 is empty if and only if $E(U_1)$ is empty. Moreover, for every recognizable sets $U_1, U_2 \subseteq M$, we have $U_1 \subseteq U_2$ if and only if $E(U_1) \subseteq E(U_2)$.*

Proof The first result is a direct consequence of the fact that E is complete over M and over V. The second result can be reduced to the first by remarking that we have $U_1 \subseteq U_2$ if and only if $U_1 \setminus U_2$ is empty. Applying Theorem 6.8 twice, we obtain that $U_1 \setminus U_2$ is empty if and only if $E(U_1) \setminus E(U_2)$ is empty, hence the result. □

Chapter 7

Systems Using FIFO Channels

Chapters 3–6 introduced a general technique for analyzing properties of infinite-state systems, as well as a general method for designing a representation system for possibly infinite sets of memory contents. In this chapter, we particularize these results to an important class of infinite-state systems: those whose memory is composed of a finite set of unbounded FIFO channels on which send and receive operations are performed [BZ83, MF85, Pac86]. Such machines are a popular model for representing and reasoning about communication protocols, and are also used for defining the semantics of standardized protocol specification languages such as SDL [CCI88] and Estelle [DAAC89]. Indeed, unbounded FIFO channels provide a useful abstraction that simplifies the semantics of specification languages and frees the protocol designer from implementation details related to buffering policies and limitations.

The chapter is organized as follows. First, it introduces systems using unbounded FIFO channels (also called *queues*) and defines their syntax, semantics, and elementary memory operations. After showing that such systems are Turing-expressive, it then proposes an encoding scheme for queue-set contents which leads to a powerful finite-state representation system for sets of queue-set contents, the *Queue Decision Diagram (QDD)*. Then, it gives algorithms implementing with QDDs all the predicates and functions required by Chapters 3–6.

7.1 Basic Notions

7.1.1 Queue SMAs

Let Σ be a finite alphabet. A *FIFO channel*, or *queue*, is an object whose value is a finite word over Σ and on which two *elementary operations* are defined:

- The *send* operation consists of appending a specified word to the queue content. Formally, the send operation is defined by the function

$$q!u : \Sigma^* \to \Sigma^* : w \mapsto w \cdot u,$$

where q denotes the queue undergoing the operation, $u \in \Sigma^*$ is the word being sent, and "$\cdot$" denotes the concatenation of words;

- The *receive* operation consists of removing a specified word from the beginning of the queue content. Formally, the receive operation is defined by the function

$$q?u \ : \ \Sigma^* \to \Sigma^* \ : \ u \cdot w \mapsto w,$$

where q denotes the queue undergoing the operation and $u \in \Sigma^*$ is the word being received.

The domains of those functions are extended to sets of queue contents in the usual way, i.e., we define

$$\begin{array}{lcl} q!u & : & 2^{\Sigma^*} \to 2^{\Sigma^*} \ : \ U \mapsto \{q!u(w) \mid w \in U\}; \\ q?u & : & 2^{\Sigma^*} \to 2^{\Sigma^*} \ : \ U \mapsto \{q?u(w) \mid w \in U\}. \end{array}$$

We are now ready to define the class of SMAs that will be studied in this chapter.

Definition 7.1 *A* Queue SMA (QSMA) *is an SMA* (C, c_0, M, m_0, Op, T) *such that*

- *Its memory domain* M *is of the form* $\Sigma_1^* \times \Sigma_2^* \times \cdots \times \Sigma_n^*$*, where* $n \geq 0$ *is the* number of queues *of the QSMA, and each* Σ_i $(1 \leq i \leq n)$ *is the* queue alphabet *of the* i*-th queue* q_i *of the QSMA. Each memory content* $m = (m_1, m_2, \ldots, m_n) \in M$ *is called a* queue-set content*, and associates a* queue content m_i *to each* q_i*;*

- *Its set of memory operations Op contains only send and receive operations. Formally, we have*

$$Op = \{q_i!u \mid 1 \leq i \leq n \wedge u \in \Sigma_i^*\} \cup \{q_i?u \mid 1 \leq i \leq n \wedge u \in \Sigma_i^*\}.$$

The notion of *Extended QSMA (EQSMA)* is defined similarly.

Definition 7.2 *An* Extended QSMA (EQSMA) *is an ESMA* $(C, c_0, M, m_0, Op, T, \bar{T})$ *such that its underlying SMA* (C, c_0, M, m_0, Op, T) *is a QSMA.*

7.1.2 Turing Expressiveness

Brand and Zafiropoulo have shown that state machines with unbounded FIFO channels can simulate arbitrary Turing machines. The following theorem is inspired by [BZ83].

Theorem 7.3 *Let* $n > 0$*, and* $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ *be finite queue alphabets such that* $|\Sigma_i| > 1$ *for at least one* $i \in \{1, 2, \ldots, n\}$*. The class of all the QSMAs that have the memory domain* $M = \Sigma_1^* \times \Sigma_2^* \times \cdots \times \Sigma_n^*$ *is Turing-expressive.*

Proof The idea is to show that QSMAs can simulate arbitrary two-counter machines, which are SMAs that have two positive integer variables, and whose memory operations can increment or decrement the value of a variable as well as test whether the value of a variable is equal to zero. It is indeed well known [HU79] that two-counter machines can simulate arbitrary Turing machines.

Let us show that an arbitrary two-counter machine $\mathcal{M}$ can be simulated by a QSMA $\mathcal{A}$ that has the memory domain M. Let a and b be two different symbols in Σ_i. We build $\mathcal{A}$ in such a way that the content of the queue q_i is always composed of a concatenation of words belonging to $\{aa, ab, bb\}$. The other queues are not used and their contents can be left empty. The idea is to make the number of occurrences of aa in the content of q_i correspond to the value of the first counter x_1 of $\mathcal{M}$. Similarly, the number of occurrences of ab will correspond to the value of the second counter x_2. The special word bb is used as a delimiter and will always appear once in the content of q_i. The SMA $\mathcal{A}$ is constructed according to the following rules:

- The initial content of q_i is $(aa)^{n_1}(ab)^{n_2}(bb)$, where n_1 and n_2 denote respectively the initial values of x_1 and x_2;
- Every increment operation of $\mathcal{M}$ is simulated by a send operation $q_i!w$, where w is either aa or ab depending on the counter involved in the increment operation.
- Every decrement operation of $\mathcal{M}$ is simulated by a loop in which the following operations are performed. First, a non-deterministic choice is made between the three operations $q_i?aa$, $q_i?ab$ and $q_i?bb$. According to our rules, exactly one of them can succeed. If the operation $q_i?u$ corresponding to the counter concerned by the decrement (i.e., $u = aa$ for x_1 and $u = ab$ for x_2), then the decrement operation is complete and the loop exits. If the operation $q_i?u$ corresponding to the other counter succeeds, then the loop resumes its execution after performing the operation $q_i!u$. Finally, if $q_i?bb$ succeeds, then the operation $q_i!bb$ is first performed. Then, the loop resumes its execution if it was the first time that $q_i?bb$ succeeded during the simulation of the current decrement operation, and blocks otherwise (since that would mean that the value of the counter involved in the decrement operation is equal to zero).
- Every test operation of $\mathcal{M}$ is simulated by a loop in which the following operations are performed. First, a non-deterministic choice is made between the three operations $q_i?u$, with $u \in \{aa, ab, bb\}$, each of these operations being followed by the corresponding send operation $q_i!u$. If the operation that succeeds corresponds to the counter involved in the test, then the loop exits and the test concludes that the value of the counter is different from zero. Otherwise, the second time that the successful operation receives bb, the loop exits and the test concludes that the value of the counter is equal to zero.

□

As it has been shown in Chapter 4, a consequence of this theorem is that the emptiness problem is undecidable for QSMAs associated with a set of accepting control locations.

7.1.3 Queue Decision Diagrams

According to the results of Chapter 6, the first step towards obtaining a representation system for sets of queue-set contents that is well suited for QSMAs is to define an encoding scheme for queue-set contents. We use the following scheme.

Definition 7.4 *Let $n \geq 0$, and $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ be finite queue alphabets such that $\Sigma_i \cap \Sigma_j = \emptyset$ for every $i, j \in \{1, 2, \ldots, n\}$ such that $i \neq j$ (the fact that the alphabets are disjoint can always be assumed without loss of generality). The* sequential encoding scheme *E_S is the relation that associates to a queue-set content the concatenation of each individual queue content. Formally, we have*

$$E_S \subseteq M \times V_S = \{((w_1, w_2, \ldots, w_n), w_1 \cdot w_2 \cdots w_n) \mid w_1 \in \Sigma_1^*, w_2 \in \Sigma_2^*, \ldots, w_n \in \Sigma_n^*\},$$

where $M = \Sigma_1^* \times \Sigma_2^* \times \cdots \times \Sigma_n^*$ and $V_S = \Sigma_1^* \cdot \Sigma_2^* \cdots \Sigma_n^*$.

The sequential encoding scheme satisfies the requirements of Definition 6.5. Indeed, by definition, E_S is complete over M, and is complete and unambiguous over V_S. The corresponding decoding function D_S is given by the formula

$$D_S : 2^{V_S} \to 2^M : L \mapsto \{(w|_1, w|_2, \ldots, w|_n) \mid w \in L\},$$

where for each $i \in \{1, 2, \ldots, n\}$, $w|_i$ is the *projection* of the word w over the alphabet Σ_i, i.e., the word obtained from w by deleting all the symbols that do not belong to Σ_i.

We are now ready to define the representation system for sets of queue-set contents.

Definition 7.5 *A* Queue Decision Diagram (QDD) *is a finite-state representation of a set $U \subseteq M$ of queue-set contents based on the sequential encoding E_S.*

In other words, a QDD representing a set $U \subseteq M$ of queue-set contents is simply a finite-state automaton accepting the sequential encodings of the elements of U.

QDDs were originally introduced in [BG96b]. In [BGWW97], it is shown that the class of sets of queue-set contents that can be represented as QDDs contains exactly all the sets that can be expressed as a finite union of Cartesian products of regular languages over the queue alphabets. This property is expressed by the following theorem.

Theorem 7.6 *Let $n \geq 0$, and $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ be finite disjoint queue alphabets. A set $U \subseteq \Sigma_1^* \times \Sigma_2^* \times \cdots \times \Sigma_n^*$ of queue-set contents is recognizable with respect to the sequential encoding E_S over the alphabets $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ if and only if there exist $k \in N$ and a regular set $U_{ij} \subseteq \Sigma_i^*$ for every $i \in \{1, 2, \ldots, n\}$ and $j \in \{1, 2, \ldots, k\}$ such that*

$$U = \bigcup_{1 \leq j \leq k} \prod_{1 \leq i \leq n} U_{ij}.$$

Proof Let $\mathcal{A} = (\Sigma, S, \Delta, I, F)$ be a QDD in strong normal form, with $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \cdots \cup \Sigma_n$. For each $i \in \{1, 2, \ldots, n\}$, we define the automaton $\mathcal{A}_i = (\Sigma_i, S_i, \Delta_i, I_i, F_i)$ as follows:

- The alphabet Σ_i is the queue alphabet of q_i;
- The set of states S_i is the set of states S of $\mathcal{A}$;
- The set of transitions Δ_i contains all the transitions of $\mathcal{A}$ labeled by words over the queue alphabet of q_i. Formally, we have $\Delta_i = \Delta \cap (S \times \Sigma_i^* \times S)$;
- The accepting states of $\mathcal{A}_i$ are all the states of $\mathcal{A}$ that are reachable from an initial state by reading only symbols in $\Sigma_1 \cup \Sigma_2 \cup \cdots \cup \Sigma_i$. Formally, we have

$$F_i = \{s \in S \mid (\exists s_0 \in I, w \in (\bigcup_{1 \leq j \leq i} \Sigma_j)^*)((s_0, w, s) \in \Delta^*)\};$$

- The initial states of $\mathcal{A}_i$ are the accepting states of $\mathcal{A}_{i-1}$ if $i > 0$, or the initial states of $\mathcal{A}$ if $i = 0$. Formally,

$$I_i = \begin{cases} I & \text{if } i = 0, \\ F_{i-1} & \text{if } i > 0. \end{cases}$$

Given $n+1$ states $s_1 \in I_1$, $s_2 \in I_2$, ..., $s_n \in I_n$, $s_{n+1} \in F_n \cap F$, the language $L_{s_1,\ldots,s_{n+1}}$ is defined as the Cartesian product $L(\mathcal{A}'_1) \times \cdots \times L(\mathcal{A}'_n)$, where each $\mathcal{A}'_i$ is a copy of the automaton $\mathcal{A}_i$ with only s_i as initial state and s_{i+1} as accepting state. By definition of the QDDs, each word w accepted by $L(\mathcal{A})$ is of the form $w = w|_1 \cdot w|_2 \cdots w|_n$. For every $i \in \{1, 2, \ldots, n\}$, each path of $\mathcal{A}$ accepting w visits a state s_i in I_i and a state s_{i+1} in F_i between which the word $w|_i$ is read. It follows that $(w|_1, w|_2, \ldots, w|_n) \in L_{s_1,\ldots,s_{n+1}}$, which implies that $L(\mathcal{A})$ is the (finite) union of all the possible $L_{s_1,\ldots,s_{n+1}}$. Since each $L_{s_1,\ldots,s_{n+1}}$ is a product of regular languages over the queue alphabets $\Sigma_1, \ldots, \Sigma_n$, we have that $L(\mathcal{A})$ is a finite union of Cartesian products of regular languages over these alphabets.

The other direction of the theorem is immediate since regular languages are closed under concatenation and finite union. □

Let us now show that the QDD is a representation system well suited for QSMAs. According to Definition 3.3, the first requirement is the ability to represent the sets

$\emptyset$, M and $\{(w_1, w_2, \ldots, w_n)\}$ for each $(w_1, w_2, \ldots, w_n) \in M$, where $M = \Sigma_1^* \times \Sigma_2^* \times \cdots \times \Sigma_n^*$. Since the sequential encodings of these sets are respectively denoted by the regular expressions $\emptyset$, $\Sigma_1^* \cdot \Sigma_2^* \cdots \Sigma_n^*$ and $w_1 \cdot w_2 \cdots w_n$, they are regular and are therefore representable by QDDs. The next requirements are that representations of the sets $U_1 \cup U_2$, $U_1 \cap U_2$, $U_1 \setminus U_2$ are computable and that the inclusion $U_1 \subseteq U_2$ and the test of emptiness of U_1 are decidable given the representations of U_1 and U_2. This is a direct consequence of the results of Section 6.4. Finally, it is required that one can compute the image of sets represented as QDDs by the memory function labeling any transition or meta-transition. The case of transitions will be addressed in Section 7.2. The requirement on the meta-transitions will be enforced by restricting the set of potential memory functions to those that can be applied algorithmically to sets of memory contents represented as QDDs. The appropriate definitions of the predicate META? and of the function MULTI-META-SET expressing this restriction will be given in Sections 7.3 and 7.4.

QDDs are defined with respect to the sequential encoding scheme E_S. One should be aware that there are other ways of encoding queue-set contents. For instance, the *interleaved encoding scheme* is defined as follows.

Definition 7.7 *Let $n \geq 0$, and $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ be disjoint finite queue alphabets. The* interleaved encoding scheme *E_I is the relation that associates to a queue-set content the word obtained by first concatenating the symbols located at the same position in each individual queue content, and then concatenating the resulting words together. Formally, we have*

$$E_I \subseteq M \times V_I = \{((w_1, w_2, \ldots, w_n), w_1[1] \cdot w_2[1] \cdots w_n[1] \cdot w_1[2] \cdot w_2[2] \cdots w_n[2] \cdot \cdots w_1[l] \cdot w_2[l] \cdots w_n[l]) \mid w_1 \in \Sigma_1^*, w_2 \in \Sigma_2^*, \ldots, w_n \in \Sigma_n^*\},$$

where

- $M = \Sigma_1^* \times \Sigma_2^* \times \cdots \times \Sigma_n^*$;
- $V_I = ((\Sigma_1 \cup \{\beta\}) \cdot (\Sigma_2 \cup \{\beta\}) \cdots (\Sigma_n \cup \{\beta\}) \setminus \{\beta^n\})^*$;
- $\beta \notin \Sigma_1 \cup \Sigma_2 \cup \cdots \cup \Sigma_n$ *is a* blank symbol;
- *For any word w and $i \in \mathbf{N}_0$, $w[i]$ is equal to the i-th symbol of w if $i \leq |w|$, and to β if $i > |w|$;*
- $l \in \mathbf{N}$ *is the length of the longest w_i, for $i \in \{1, 2, \ldots, n\}$.*

The sequential and interleaved encoding schemes have different expressivenesses, i.e., the classes of sets of queue-set contents that they allow to represent do not coincide. Intuitively, the sequential encoding scheme allows to represent sets of queue-set contents in which the contents of individual queues are loosely coupled,

while the interleaved encoding scheme imposes a tight correlation between those contents.

Let us give an example. Consider a QSMA with two queues q_1 and q_2 such that $\Sigma_1 = \{a\}$ and $\Sigma_2 = \{b\}$. The set of queue-set contents

$$U = \{(a^n, b^n) \mid n \in \mathbf{N}\},$$

in which the contents of q_1 and q_2 are tightly correlated, cannot be represented by a QDD. Indeed, we have

$$E_S(U) = \{a^n b^n \mid n \in \mathbf{N}\},$$

which is not regular (since any finite-state automaton with p states accepting a word of the form $a^n b^n$ with $n > p$ would also have to accept the word $a^{n+k} b^n$ for some $k > 0$). On the other hand, the language

$$E_I(U) = \{(ab)^n \mid n \in \mathbf{N}\}$$

can be denoted by the regular expression $(a \cdot b)^*$ and is hence regular.

The choice of the sequential rather than interleaved encoding scheme in the definition of QDDs is motivated by the nature of the meta-transitions that the two schemes allow to consider. Inspired by [BG96b], we qualify as acceptable an encoding scheme according to which the transitive closure of a single elementary operation can be computed over representable sets. This is equivalent to stating that it must be possible to associate a cycle meta-transition to each cycle whose body is labeled by a single elementary operation. It will be shown in Section 7.3 that this requirement is met by the sequential encoding scheme (and, in addition, that a class of cycle meta-transitions much broader than those corresponding to single memory operations can be considered in this case). Contrariwise, the transitive closure of a single receive operation is not always computable over sets represented according to the interleaved encoding scheme. Indeed, let us consider a QSMA with two queues q_1 and q_2 such that $\Sigma_1 = \{a, b\}$ and $\Sigma_2 = \{c, d\}$, and the set of queue-set contents

$$U = \{(a^n b^m, c^n d^m) \mid n, m \in \mathbf{N}\}.$$

Since the language

$$E_I(U) = \{(ac)^n (bd)^m \mid n, m \in \mathbf{N}\}$$

is denoted by the regular expression $(a \cdot c)^* \cdot (b \cdot d)^*$, we have that U is representable with respect to the interleaved encoding scheme. Applying to U the transitive closure of the receive operation $q_1?a$ yields the set

$$U' = (q_1?a)^*(U) = \{(a^n b^m, c^{n'} d^m) \mid n, n', m \in \mathbf{N} \wedge n' \geq n\},$$

for which we have

$$E_I(U') = \{(ac)^n(bc)^m(\beta c)^{n'-n-m}(\beta d)^m \mid n, n', m \in \mathbf{N} \wedge n' \geq n+m\} \cup \{(ac)^n(bc)^{n'-n}(bd)^{n+m-n'}(\beta d)^{n'-n} \mid n, n', m \in \mathbf{N} \wedge n' \leq n+m\}.$$

The language $E_I(U')$ is not regular. Indeed, any finite-state automaton with p states accepting a word of the form $(bc)^m(\beta c)^{m'}(\beta d)^m$ with $m > p$ and $m' \in \mathbf{N}$ would also accept the word $(bc)^{m+k}(\beta c)^{m'}(\beta d)^m$ for some $k > 0$, despite the fact that this word does not belong to the language. It follows that U' cannot be represented according to the interleaved encoding scheme.

7.1.4 Notations

Let us recall some notations introduced in Chapter 3 and present some new definitions that will be used throughout this chapter. If $\theta_1, \theta_2, \ldots, \theta_p$ are elementary queue operations (of the form $q_i!u$ or $q_i?u$), then $\sigma = \theta_1; \theta_2; \ldots; \theta_p$ is a *sequence* of operations. The effect of a sequence of operations $\sigma = \theta_1; \theta_2; \ldots; \theta_p$ on a queue-set content u is $\sigma(u) = \theta_p(\cdots\theta_2(\theta_1(u))\cdots)$. The effect of a sequence of operations σ on a set U of queue-set contents is $\sigma(U) = \{\sigma(u) \mid u \in U\}$. The sum of the lengths of the words involved in the elementary queue operations composing σ is denoted $|\sigma|$. We denote by $\sigma_!$ (resp. $\sigma_?$) the subsequence of σ consisting of all the send (resp. receive) operations. The *projection* of σ over the queue q_i, which is denoted $\sigma|_i$, is the subsequence of σ consisting of all the operations involving q_i. We write $\mu(\sigma)$ to represent the word obtained from σ by extracting the message symbols from the queue operations, i.e., by replacing each $q_i!u$ and $q_i?u$ by u. If $p \in \mathbf{N}$, then σ^p denotes the sequence obtained by repeating σ p times. Let $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ be finite queue alphabets. The *projection* of a word $w \in (\Sigma_1 \cup \Sigma_2 \cup \cdots \cup \Sigma_n)^*$ over the alphabet Σ_i is denoted $w|_i$ and is defined as the word obtained from w by deleting all the symbols that do not belong to Σ_i. We define $w|_{>i}$ as the word obtained from w by deleting all the symbols that do not belong to $\Sigma_{i+1} \cup \Sigma_{i+2} \cup \cdots \cup \Sigma_n$. The notations $w|_{<i}$, $w|_{\geq i}$, $w|_{\leq i}$, $w|_{\neq i}$ are defined similarly. Projections are extended naturally to languages by defining $L|_i$, with $L \subseteq (\Sigma_1 \cup \Sigma_2 \cup \cdots \cup \Sigma_n)^*$, as $\{w|_i \mid w \in L\}$. The notations $L|_{>i}$, $L|_{<i}$, $L|_{\geq i}$, $L|_{\leq i}$, $L|_{\neq i}$ are defined similarly.

7.2 Elementary Queue Operations

The problem addressed here consists of computing the image of a set of queue-set contents represented as a QDD by the function labeling a transition of a QSMA. Since such a function is equivalent to a finite sequence of send and receive operations involving a single symbol, it is sufficient to obtain algorithms for computing the effect of these two operations on sets represented as QDDs. We first consider the case of a QSMA with only one queue, and then reduce the general problem to that case.

function APPLY-RECEIVE-ONE(QDD $(\Sigma, S, \Delta, I, F)$, symbol a) : QDD

1: **begin**
2: $(\Sigma, S, \Delta, I, F) :=$ NORMALIZE$((\Sigma, S, \Delta, I, F))$;
3: $I := \{s' \in S \mid (\exists s \in I)(s, a, s') \in \Delta^*\}$;
4: **return** $(\Sigma, S, \Delta, I, F)$
5: **end**.

Figure 7.1: Receive operation for a single-queue QDD.

7.2.1 Systems with One Queue

Let q be a queue of alphabet Σ. If q is the only queue of the system, then sets of queue-set contents (actually, sets of queue contents) coincide with the languages of the encodings of their elements, i.e., we have $U = E_S(U)$ for every $U \subseteq \Sigma^*$.

The first problem consists of computing a QDD representing $(q?a)(U)$ given a QDD representing a set $U \subseteq \Sigma^*$ and a symbol $a \in \Sigma$. We present here the solution that is given in [BG96b]. Let $\mathcal{A}$ be a finite-state automaton accepting U. We can assume that $\mathcal{A}$ is in normal form. In order to compute the set $(q?a)(U)$, one has simply to remove from U all the words that do not begin with a, and then delete the initial symbol from all the remaining words. An automaton accepting $(q?a)(U)$ can thus be obtained by moving all the initial states of $\mathcal{A}$ along transitions labeled by a. An algorithm implementing this operation is given in Figure 7.1.

Theorem 7.8 *Let q be a queue of alphabet Σ, $a \in \Sigma$ be a symbol, and $\mathcal{A}$ be a QDD representing the set $U \subseteq \Sigma^*$. APPLY-RECEIVE-ONE$(\mathcal{A}, a)$ is a QDD representing the set $(q?a)(U)$.*

Proof Let $\mathcal{A}'$ = APPLY-RECEIVE-ONE$(\mathcal{A}, a)$. The automaton $\mathcal{A}'$ has the accepting path π if and only if the automaton $\mathcal{A}$ has an accepting path of the form $\pi'\pi$, where the only symbol read by π' is a. □

The second problem consists of computing a QDD representing $(q!a)(U)$ given a QDD representing a set $U \subseteq \Sigma^*$ and a symbol $a \in \Sigma$. Once again, we present here the solution given in [BG96b]. Let $\mathcal{A}$ be a finite-state automaton accepting U. In order to compute the set $(q!a)(U)$, one simply has to append the symbol a to each word belonging to U. An automaton accepting $(q!a)(U)$ can thus be obtained by creating a new state s, adding transitions labeled by a leading from all the accepting states of $\mathcal{A}$ to s, and finally keeping s as the only accepting state. An algorithm implementing this operation is given in Figure 7.2.

function APPLY-SEND-ONE(QDD $(\Sigma, S, \Delta, I, F)$, symbol a) : QDD

```
1:  begin
2:      let s ∉ S;
3:      S := S ∪ {s};
4:      Δ := Δ ∪ {(s', a, s) | s' ∈ F};
5:      F := {s};
6:      return (Σ, S, Δ, I, F)
7:  end.
```

Figure 7.2: Send operation for a single-queue QDD.

Theorem 7.9 *Let q be a queue of alphabet Σ, $a \in \Sigma$ be a symbol, and $\mathcal{A}$ be a QDD representing the set $U \subseteq \Sigma^*$. APPLY-SEND-ONE($\mathcal{A}, a$) is a QDD representing the set $(q!a)(U)$.*

Proof Let $\mathcal{A}'$ = APPLY-SEND-ONE($\mathcal{A}, a$). The automaton $\mathcal{A}'$ has the accepting path π if and only if the automaton $\mathcal{A}$ has an accepting path π' such that $\pi = \pi'\pi''$, where the only symbol read by π'' is a. □

The algorithms proposed in this section can easily be extended to sequences of queue operations by simply performing one by one the operations composing the sequence. Send and receive operations involving more than one symbol can also be performed in this way since for any word w, the operations $q!w$ and $q?w$ are respectively equivalent to the sequences $q!(w[1]); q!(w[2]); \ldots; q!(w[|w|])$ and $q?(w[1]); q?(w[2]); \ldots; q?(w[|w|])$. An algorithm for computing the image by an arbitrary sequence of queue operations of a set of queue contents represented as a QDD is given in Figure 7.3.

Theorem 7.10 *Let q be a queue of alphabet Σ, σ be a sequence of elementary operations on q, and $\mathcal{A}$ be a QDD representing the set $U \subseteq \Sigma^*$. APPLY-ONE($\mathcal{A}, \sigma$) is a QDD representing the set $\sigma(U)$.*

Proof Immediate. □

7.2.2 Systems with Any Number of Queues

The problem addressed here is to compute the image by a send or a receive operation of a set of queue-set contents involving an arbitrary number $n \geq 0$ of queues. We present here the solution given in [BGWW97], which consists of reducing the problem to the case of a system with only one queue. The idea is to show that

```
function APPLY-ONE(QDD 𝒜, sequence of queue operations θ_1; θ_2; ...; θ_m) : QDD
 1:   var i, j : integer;
 2:   var f : function;
 3:   begin
 4:       for i := 1 to m do
 5:           begin
 6:               if θ_i is a send operation then
 7:                   f := APPLY-SEND-ONE
 8:               else
 9:                   f := APPLY-RECEIVE-ONE;
10:               for j := 1 to |μ(θ_i)| do
11:                   𝒜 := f(𝒜, μ(θ_i)[j])
12:           end;
13:       return 𝒜
14:   end.
```

Figure 7.3: Image of a single-queue QDD by a sequence of queue operations.

any algorithm applying an operation to the set of contents of a single queue can be turned into an algorithm performing the same operation on a specified queue of a set of queue-set contents with an arbitrary number of queues.

Formally, let $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ be the alphabets of the queues $q_1, q_2, \ldots, q_n$ and let $M = \Sigma_1^* \times \Sigma_2^* \times \cdots \times \Sigma_n^*$. We consider a particular queue q_i $(1 \leq i \leq n)$, and a function $f_i : \Sigma_i^* \to 2^{\Sigma_i^*}$ transforming the content of q_i into a set of image contents. The function f_i can be extended into a function $\bar{f}_i$ defined over the sets of contents of q_i:

$$\bar{f}_i : 2^{\Sigma_i^*} \to 2^{\Sigma_i^*} : U \mapsto \bigcup_{w \in U} f_i(w).$$

Given an algorithm for computing the image by $\bar{f}_i$ of a representable set of queue contents, the problem consists of deriving an algorithm for applying the function

$$\bar{f}'_i : 2^M \to 2^M : U \mapsto \{(w_1, \ldots, w_{i-1}, w'_i, w_{i+1}, \ldots, w_n) \mid (\exists (w_1, \ldots, w_n) \in U)(w'_i \in f_i(w_i))\}$$

to representable sets of queue-set contents.

Intuitively, turning an algorithm for performing $\bar{f}_i$ into one for performing $\bar{f}'_i$ can be done in the following way. Let $\mathcal{A}$ be a QDD representing the set of queue-set contents $U \subseteq M$. Since queue-set contents are encoded sequentially, each path of $\mathcal{A}$ reading the queue-set content $(w_1, w_2, \ldots, w_n)$ is composed of three successive subpaths reading respectively the queue contents w_1 to w_{i-1}, w_i, and w_{i+1} to w_n. Since $\mathcal{A}$ has a finite number of states, there are only a finite number of possible starting and ending states for each of those subpaths. Taking into account all the possibilities, we obtain that the language accepted by $\mathcal{A}$ can be expressed as a finite union of languages of the form $L_{<i} \cdot L_{=i} \cdot L_{>i}$, where $L_{<i}$, $L_{=i}$ and $L_{>i}$ are regular languages defined respectively over the alphabets $\Sigma_1 \cup \cdots \cup \Sigma_{i-1}$, Σ_i, and $\Sigma_{i+1} \cup \cdots \cup \Sigma_n$. One can apply $\bar{f}'_i$ to the set represented by $\mathcal{A}$ by first computing automata accepting the $L_{<i}$, $L_{=i}$ and $L_{>i}$ involved in the expression of $L(\mathcal{A})$, and then applying the algorithm implementing $\bar{f}_i$ (which is available by hypothesis) to each automaton accepting an $L_{=i}$. An algorithm[1] implementing this method is given in Figure 7.4.

Theorem 7.11 *Let $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ be finite disjoint queue alphabets, $M = \Sigma_1^* \times \Sigma_2^* \times \cdots \times \Sigma_n^*$, $i \in \{1, 2, \ldots, n\}$ and $f_i : \Sigma_i^* \to 2^{\Sigma_i^*}$. Let $\bar{f}_i$ and $\bar{f}'_i$ be functions derived from f_i as previously explained in this section. Let g be a computable function*

[1]The argument p of the function PERFORM-FUNCTION implemented by this algorithm is used as a convenience parameter. It is not used by the function, but simply transmitted to the algorithm implementing $\bar{f}_i$. For instance, if $\bar{f}_i$ is implemented by the algorithm APPLY-SEND-ONE introduced in Section 7.2.1, then the value of p represents the symbol being sent. This parameter is introduced here for compatibility with future applications of PERFORM-FUNCTION, and may be ignored in this section.

function PERFORM-FUNCTION(QDD $(\Sigma, S, \Delta, I, F)$, function f, integer i, parameter p) : QDD

1: **var** $\mathcal{A}, \mathcal{A}', \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$: QDDs;
2: s, s' : states;
3: **begin**
4: $\mathcal{A} := (\Sigma, S, \Delta, I, F) :=$ NORMALIZE$((\Sigma, S, \Delta, I, F))$;
5: $\mathcal{A}' := (\Sigma, \emptyset, \emptyset, \emptyset, \emptyset)$;
6: **for each** $(s, s') \in S^2$ **such that**
$(\exists s_0 \in I, w \in L(\mathcal{A}))((s_0, w|_{<i}, s) \in \Delta^* \wedge (s, w|_i, s') \in \Delta^*)$ **do**
7: **begin**
8: $\mathcal{A}_1 := (\Sigma|_{<i}, S, \Delta \cap (S \times (\Sigma|_{<i})^* \times S), I, \{s\})$;
9: $\mathcal{A}_2 := f((\Sigma|_i, S, \Delta \cap (S \times (\Sigma|_i)^* \times S), \{s\}, \{s'\}), p)$;
10: $\mathcal{A}_3 := (\Sigma|_{>i}, S, \Delta \cap (S \times (\Sigma|_{>i})^* \times S), \{s'\}, F)$;
11: $\mathcal{A}' :=$ UNION$(\mathcal{A}',$ CONCATENATE(
CONCATENATE$(\mathcal{A}_1, \mathcal{A}_2), \mathcal{A}_3))$
12: **end**;
13: **return** $\mathcal{A}'$
14: **end**.

Figure 7.4: Application of a QDD operation to a specified queue.

and p be a value such that for every set $U_i \subseteq \Sigma_i^*$ and QDD $\mathcal{A}_i$ representing this set, $g(\mathcal{A}_i, p)$ is a QDD representing the set $\bar{f}_i(U_i)$. For every set $U \subseteq M$ and QDD $\mathcal{A} = (\Sigma, S, \Delta, I, F)$ representing this set, PERFORM-FUNCTION$(\mathcal{A}, g, i, p)$ returns a QDD representing the set $\bar{f}'_i(U)$.

Proof Assume that $\mathcal{A}$ is in normal form. Let $\{(s_1, s'_1), (s_2, s'_2), \ldots, (s_m, s'_m)\}$ $(m \geq 0)$ be the set of all the pairs (s, s') of states of $\mathcal{A}$ satisfying the condition at Line 6 of the algorithm. For each pair (s_k, s'_k), we define the automata $\mathcal{A}_{k,<} = (\Sigma|_{<i}, S, \Delta \cap (S \times (\Sigma|_{<i})^* \times S), I, \{s_k\})$, $\mathcal{A}_{k,=} = (\Sigma|_i, S, \Delta \cap (S \times (\Sigma|_i)^* \times S), \{s_k\}, \{s'_k\})$ and $\mathcal{A}_{k,>} = (\Sigma|_{>i}, S, \Delta \cap (S \times (\Sigma|_{>i})^* \times S), \{s'_k\}, F)$. We have

$$L(\mathcal{A}) = \bigcup_{1 \leq j \leq m} L(\mathcal{A}_{j,<}) \cdot L(\mathcal{A}_{j,=}) \cdot L(\mathcal{A}_{j,>}),$$

with $L(\mathcal{A}_{k,<}) \subseteq (\Sigma_1 \cup \cdots \cup \Sigma_{i-1})^*$, $L(\mathcal{A}_{k,=}) \subseteq \Sigma_i^*$ and $L(\mathcal{A}_{k,>}) \subseteq (\Sigma_{i+1} \cup \cdots \cup \Sigma_n)^*$ for every $k \in \{1, 2, \ldots, m\}$. Let $\mathcal{A}'$ be the QDD returned at Line 11. We have

$$\begin{aligned} L(\mathcal{A}') &= \bigcup_{1 \leq j \leq m} L(\mathcal{A}_{j,<}) \cdot \bar{f}_i(L(\mathcal{A}_{j,=})) \cdot L(\mathcal{A}_{j,>}) \\ &= \bigcup_{w \in L(\mathcal{A})} \{w|_{<i}\} \cdot f_i(w|_i) \cdot \{w|_{>i}\}. \end{aligned}$$

The QDD $\mathcal{A}'$ thus represents the set $\bar{f}'_i(U)$. □

The first application of the algorithm of Figure 7.4 is to compute the effect of an elementary operation on a set of queue-set contents represented as a QDD. Algorithms for performing the send and the receive operation are respectively given in Figures 7.5 and 7.6.

Theorem 7.12 *Let $q_1, q_2, \ldots, q_n$ be queues of alphabets $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$, $M = \Sigma_1^* \times \Sigma_2^* \times \cdots \times \Sigma_n^*$, $i \in \{1, 2, \ldots, n\}$, $a \in \Sigma_i$ and $\mathcal{A}$ be a QDD representing the set $U \subseteq M$. APPLY-SEND$(\mathcal{A}, i, a)$ is a QDD representing the set $(q_i!a)(U)$.*

Proof The result is a direct consequence of Theorems 7.9 and 7.11. □

Theorem 7.13 *Let $q_1, q_2, \ldots, q_n$ be queues of alphabets $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$, $M = \Sigma_1^* \times \Sigma_2^* \times \cdots \times \Sigma_n^*$, $i \in \{1, 2, \ldots, n\}$, $a \in \Sigma_i$ and $\mathcal{A}$ be a QDD representing the set $U \subseteq M$. APPLY-RECEIVE$(\mathcal{A}, i, a)$ is a QDD representing the set $(q_i?a)(U)$.*

Proof The result is a direct consequence of Theorems 7.8 and 7.11. □

The property expressed by Theorem 7.11 is very general and is not limited to elementary queue operations. It will be used several times in the rest of this chapter in order to obtain algorithms for operations involving an arbitrary number of queues from algorithms involving only one queue.

```
function APPLY-SEND(QDD A, integer i, symbol a) : QDD
1:  begin
2:      return PERFORM-FUNCTION(A, APPLY-SEND-ONE, i, a)
3:  end.
```

Figure 7.5: Send operation for an arbitrary QDD.

```
function APPLY-RECEIVE(QDD A, integer i, symbol a) : QDD
1:  begin
2:      return PERFORM-FUNCTION(A, APPLY-RECEIVE-ONE, i, a)
3:  end.
```

Figure 7.6: Receive operation for an arbitrary QDD.

```
function APPLY(QDD A, sequence of queue operations σ,
                                  alphabets Σ_1, Σ_2, ..., Σ_n) : QDD
1:  var i : integer;
2:  begin
3:      for i := 1 to n do
4:          A := PERFORM-FUNCTION(A, APPLY-ONE, i, σ|_i);
5:      return A
6:  end.
```

Figure 7.7: Image of an arbitrary QDD by a sequence of queue operations.

7.2.3 Sequence of Elementary Operations

As in the case of single-queue QDDs, the algorithms for performing elementary queue operations on arbitrary QDDs can easily be generalized to sequences of queue operations. A simple way of performing this generalization consists of remarking that two operations involving different queues are independent, i.e., that the result of applying those operations to a queue-set content does not depend on the order in which they are applied. It follows that for every sequence σ of operations involving the queues $q_1, q_2, \ldots, q_n$ and set of queue-set contents U, we have $\sigma(U) = (\sigma|_1; \sigma|_2; \ldots; \sigma|_n)(U)$. Applying σ to U can thus be done by applying successively to U the projections of σ onto the different queues of the system. An algorithm formalizing this method is given in Figure 7.7.

Theorem 7.14 *Let $q_1, q_2, \ldots, q_n$ be queues of alphabets $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$, σ be a sequence of elementary operations on these queues, and $\mathcal{A}$ be a QDD representing the set $U \subseteq \Sigma_1^* \times \Sigma_2^* \times \cdots \times \Sigma_n^*$. APPLY($\mathcal{A}, \sigma, \Sigma_1, \Sigma_2, \ldots, \Sigma_n$) is a QDD representing the set $\sigma(U)$.*

Proof Immediate, as a consequence of Theorems 7.10 and 7.11. □

7.3 Creation of Cycle Meta-Transitions

As it has been shown in Section 3.4.1, the creation of cycle meta-transitions is controlled by:

- A computable predicate META? defined over the set of potential sequences of operations, whose purpose is to decide whether the meta-transition corresponding to a given sequence can be created, i.e., whether the closure of the sequence can always be applied to arbitrary sets of memory contents;

- An algorithm for computing the image of any representable set of memory contents by the closure of a sequence of operations satisfying META?.

This section aims at providing algorithms for computing a suitable predicate META? over sequences of queue operations, and for applying closures of such sequences to sets of queue-set contents represented by QDDs. We will give here the most general solution to this problem, in the sense that it will always be possible to compute the closure σ^* of a sequence σ provided that the image by σ^* of any representable set is representable. Computing the truth value of META? for a particular sequence σ will thus amount to deciding whether the image by σ^* of any representable set of queue-set contents is representable.

7.3.1 Systems with One Queue

In the case of systems with only one queue, a somewhat surprising result is that for any sequence σ of queue operations and recognizable set U of queue contents, $\sigma^*(U)$ is recognizable. Moreover, a finite-state representation of $\sigma^*(U)$ is computable given σ and a representation of U. We establish this result constructively, i.e., in the form of an algorithm for computing the image by σ^* of an arbitrary set represented as a QDD.

Let q be a queue of alphabet Σ, $U \subseteq \Sigma^*$ be a recognizable set of queue contents, $\mathcal{A}$ be a QDD representing U, and σ be a finite sequence of operations on q. We assume that $\mathcal{A}$ is in normal form. We have $\sigma^*(U) = L(\mathcal{A}_0) \cup L(\mathcal{A}_1) \cup L(\mathcal{A}_2) \cup \cdots$, where $\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2, \ldots$ are QDDs such that:

- $\mathcal{A}_0 = \mathcal{A}$;
- $\mathcal{A}_{i+1} = \text{APPLY-ONE}(\mathcal{A}_i, \sigma)$ for every $i \in \mathbf{N}$.

For each $i \in \mathbf{N}$, we denote $(\Sigma_i, S_i, \Delta_i, I_i, F_i)$ the components of $\mathcal{A}_i$.

The goal of the algorithm we are about to develop is to construct a finite automaton accepting exactly all the words accepted by any of the $\mathcal{A}_i$. The idea is that there is some redundancy among the transitions of the different $\mathcal{A}_i$, and that this redundancy can be captured within a finite structure. The first step is to characterize the relationship between the sets of states and of transitions of the different $\mathcal{A}_i$.

Let $i \in \mathbf{N}$. The computation of APPLY-ONE$(\mathcal{A}_i, \sigma)$ proceeds by applying successively to $\mathcal{A}_i$ each elementary queue operation composing σ. The effect of a receive operation is to modify the set of initial states of the automaton. The effect of a send operation is to create a new state which becomes the only accepting state, as well as to add some transitions ending in that new accepting state. Thus, the sets of states and of transitions of $\mathcal{A}_{i+1}$ are identical to those of $\mathcal{A}_i$, except for a series of new states and transitions depending on $\sigma_!$. The situation is depicted in Figure 7.8.

The set of all the states that have been created during the first i applications of APPLY-ONE is called the *tail* of $\mathcal{A}_i$, and is denoted $\mathit{tail}(i)$. If $i > 0$, we thus have $S_i = S_{i-1} \cup \mathit{tail}(i)$. The set $\mathit{tail}(i)$ contains exactly $i|\sigma_!|$ states which are denoted $\mathit{tail}(i,1), \mathit{tail}(i,2), \ldots, \mathit{tail}(i,i|\sigma_!|)$, in the order of their creation. For every $s \in \mathit{tail}(i)$, the integer j such that $s = \mathit{tail}(i,j)$ is called the *rank* of s and is denoted $\mathit{rank}(s)$.

During the applications of APPLY-ONE, the effect of each receive operation is to move the initial states along transitions labeled by the symbol being received. Since $S_i \subseteq S_{i+1}$ and $\Delta_i \subseteq \Delta_{i+1}$, we have that for every $s' \in I_{i+1}$ there exists $s \in I_i$ such that $(s, \mu(\sigma_?), s') \in \Delta_{i+1}^*$. We say that the initial state s' is a *shift* of s. The set I_{i+1} is thus the set of all the shifts of the elements of I_i. If $s' \in I_{i+1} \cap \mathit{tail}(i+1)$ is a shift of $s \in I_i \cap \mathit{tail}(i)$, then $\mathit{rank}(s') = \mathit{rank}(s) + |\sigma_?|$.

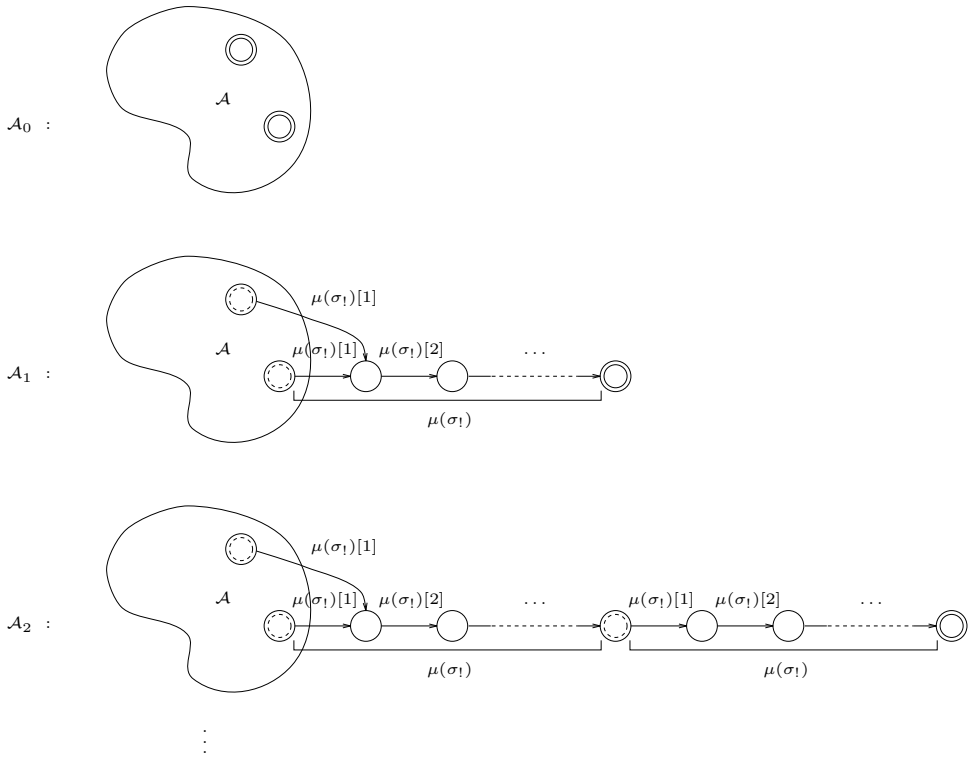

Figure 7.8: Effect of repeated applications of APPLY-ONE.

Let us divide I_i into two subsets, containing respectively the states that belong to *tail*(i) and those that do not. We first consider the latter set, which we denote I_i^α. We have $I_i^\alpha = I_i \cap S_0$. By construction, there is no path of transitions of $\mathcal{A}_i$ leading from a state in *tail*(i) to a state in S_0. It follows that I_i^α contains only shifts of states belonging to I_{i-1}^α (if $i > 0$), which implies that the value of I_i^α depends only on the value of I_{i-1}^α. Since this holds for every $i \in \mathbf{N}_0$ and since the size of each I_i^α is bounded by the number of states in S_0, we have that the sequence $I_0^\alpha, I_1^\alpha, I_2^\alpha, \ldots$ is *ultimately periodic*, i.e., that there exist $b \in \mathbf{N}$ and $p \in \mathbf{N}_0$ such that for every $i \geq b$, $I_i^\alpha = I_{i+p}^\alpha$. The numbers b and p are respectively called the *base* and the *period* of the sequence of the I_i^α. Without loss of generality, we can choose p and b among the multiples of $|\sigma_!|$ (the purpose of this requirement is to simplify future computations).

Computing the values of b and of p is the first step towards capturing the regularity of the infinite sequence of the $\mathcal{A}_i$. Indeed, considering only one $\mathcal{A}_i$ out of p eliminates the need to take into account the changes occurring among the initial states that do not belong to the tail. We define the QDDs $\mathcal{A}'_0, \mathcal{A}'_1, \mathcal{A}'_2, \ldots$ such that for every $i \in \mathbf{N}$, $\mathcal{A}'_i = \mathcal{A}_{b+ip}$. The components of each $\mathcal{A}'_i$ are denoted $(\Sigma'_i, S'_i, \Delta'_i, I'_i, F'_i)$. The problem which consists of computing the infinite union of the $L(\mathcal{A}_i)$ is easily reduced to the computation of the infinite union of the $L(\mathcal{A}'_i)$, thanks to the following result.

Theorem 7.15 *The language $\bigcup_{i\in\mathbf{N}} L(\mathcal{A}_i)$ can be expressed in terms of $\bigcup_{i\in\mathbf{N}} L(\mathcal{A}'_i)$ and of a finite number of individual $L(\mathcal{A}_i)$ using finite union and applying σ a finite number of times.*

Proof

$$\begin{aligned}
\bigcup_{i\in\mathbf{N}} L(\mathcal{A}_i) &= \bigcup_{i<b} L(\mathcal{A}_i) \cup \bigcup_{i\geq b} L(\mathcal{A}_i) \\
&= \bigcup_{i<b} L(\mathcal{A}_i) \cup \bigcup_{0\leq j<p} \bigcup_{i\in N} L(\mathcal{A}_{b+j+ip}) \\
&= \bigcup_{i<b} \sigma^i(L(\mathcal{A}_0)) \cup \bigcup_{0\leq j<p} \sigma^j \left(\bigcup_{i\in\mathbf{N}} L(\mathcal{A}_{b+ip}) \right) \\
&= \bigcup_{i<b} \sigma^i(L(\mathcal{A}_0)) \cup \bigcup_{0\leq j<p} \sigma^j \left(\bigcup_{i\in\mathbf{N}} L(\mathcal{A}'_i) \right).
\end{aligned}$$

□

If the sequence σ is such that $|\sigma_!| = 0$, then the tail of each $\mathcal{A}'_i$ is empty, and thus we have $\mathcal{A}'_i = \mathcal{A}'_j$ for every $i, j \in \mathbf{N}$. In this case, we have

$$\bigcup_{i\in N} L(\mathcal{A}'_i) = L(\mathcal{A}'_0).$$

From now on, we assume that $|\sigma_!| > 0$. For each $i \in \mathbf{N}$, the tail of $\mathcal{A}'_i$ is denoted $tail'(i)$ (we thus have $tail'(i) = tail(b + ip)$). The tail of $\mathcal{A}'_i$ contains exactly $|tail'(i)| = (b + ip)|\sigma_!|$ states which are denoted $tail'(i, 1)$, $tail'(i, 2), \ldots,$ $tail'(i, |tail'(i)|)$, in the order of their creation. If $i > 0$, then the initial states of $\mathcal{A}'_i$ are obtained by shifting p times those of $\mathcal{A}'_{i-1}$. We have the following definition.

Definition 7.16 *Let $i \in \mathbf{N}_0$. An initial state $s' \in I'_i$ of $\mathcal{A}'_i$ is a* p-shift *of an initial state $s \in I'_{i-1}$ of $\mathcal{A}'_{i-1}$ if we have $(s, \mu(\sigma_?)^p, s') \in (\Delta'_i)^*$.*

The initial states of $\mathcal{A}'_i$ are p-shifts of initial states of $\mathcal{A}'_{i-1}$. A crucial point is that this property does not imply that for every initial state s of $\mathcal{A}'_{i-1}$, there exists an initial state s' of $\mathcal{A}'_i$ that is a p-shift of s. Indeed, during the p applications of σ that allow to obtain $\mathcal{A}'_i$ from $\mathcal{A}'_{i-1}$, initial states are shifted (by receive operations) and transitions are created (by send operations) in an order depending on the place of the different operations in σ. If the transitions that are needed in order to shift a state are not yet created when the receive operation is performed, then the state cannot be shifted. On the other hand, if the necessary transitions are available prior to executing the receive operation, then the state can be shifted. Those observations allow us to write a sufficient condition on initial states that can be p-shifted.

Definition 7.17 *Let $i \in \mathbf{N}_0$. An initial state $s \in I'_i$ of $\mathcal{A}'_i$ is* robust *if either:*

- *There exists* $s' \in S'_i$ *such that* $(s, \mu(\sigma_?)^p, s') \in (\Delta'_i)^*$, *or*
- $|\sigma_!| \geq |\sigma_?|$ *and there exists* $s'' \in tail'(i-1)$ *such that* $(s'', \mu(\sigma_?)^p, s) \in (\Delta'_i)^*$.

The sufficient condition is expressed by the following theorem.

Theorem 7.18 *Let* $i \in \mathbf{N}_0$ *and* $s \in I'_i$. *If* s *is robust, then there exists an initial state* $s' \in I'_{i+1}$ *such that* s' *is a* p*-shift of* s.

Proof

- If there exists $s' \in S'_i$ such that $(s, \mu(\sigma_?)^p, s') \in (\Delta'_i)^*$, then $s' \in I'_{i+1}$ and the result is immediate.
- If $|\sigma_!| \geq |\sigma_?|$ and there exists $s'' \in tail'(i-1)$ such that $(s'', \mu(\sigma_?)^p, s) \in (\Delta'_i)^*$, then $s'' \in I'_{i-1}$ and s is a p-shift of s''. During the p executions of APPLY-ONE that compute $\mathcal{A}'_i$ from $\mathcal{A}'_{i-1}$, the state s of $\mathcal{A}'_i$ is obtained from the state s'' of $\mathcal{A}'_{i-1}$ by following a sequence of transitions labeled by $\mu(\sigma_?)^p$. Since the tail of $\mathcal{A}'_i$ has a greater length than the one of $\mathcal{A}'_{i-1}$, and since for any subpath outgoing from s'' in $\mathcal{A}'_{i-1}$ labeled by a word $w \in \Sigma^*$, there exists a subpath outgoing from s in $\mathcal{A}'_i$ labeled by w, following an identical sequence of transitions is possible and allows to go from the state s of $\mathcal{A}'_i$ to some state s' of $\mathcal{A}'_{i+1}$.

□

Applying this sufficient condition to the initial states present on the tail of each $\mathcal{A}_i$ yields the following result.

Theorem 7.19 *Let* $i \geq 0$ *and* $s \in tail'(i) \cap I'_i$. *If* $rank(s) > p|\sigma_?|$, *and either* $|\sigma_!| \geq |\sigma_?|$ *or* $rank(s) \leq |tail'(i)| - p|\sigma_?|$, *then* s *is a robust state of* $\mathcal{A}'_i$.

Proof Since $rank(s) > p|\sigma_?|$, we have that $i > 0$ and that s is a p-shift of a state s' of $\mathcal{A}'_{i-1}$ such that $rank(s') = rank(s) - p|\sigma_?|$. Therefore, there exists in the tail of $\mathcal{A}'_i$ a path π composed of $p|\sigma_?|$ transitions leading from s' to s and labeled by $\mu(\sigma_?)^p$. If $|\sigma_!| \geq |\sigma_?|$, then s is robust by definition since $s' \in tail'(i-1)$. If $|\sigma_!| < |\sigma_?|$, then $rank(s) \leq |tail'(i)| - p|\sigma_?|$. There exists in the tail of $\mathcal{A}'_i$ a path π' composed of $p|\sigma_?|$ transitions leading from s to a state s'' such that $rank(s'') = rank(s) + p|\sigma_?|$. By construction of the tail of $\mathcal{A}'_i$, the transitions composing π' are labeled by the same word as those composing π. Thus, we have $(s, \mu(\sigma_?)^p, s'') \in \Delta'_i$. Hence, s is robust. □

The effect of Theorem 7.19 is illustrated in Figures 7.9 and 7.10, in which the grey area contains the states that satisfy the hypotheses of the theorem. In these figures, dashed diagonal arrows represent p-shifts of initial states, and dashed ovals are used to group initial states that are shifted together.

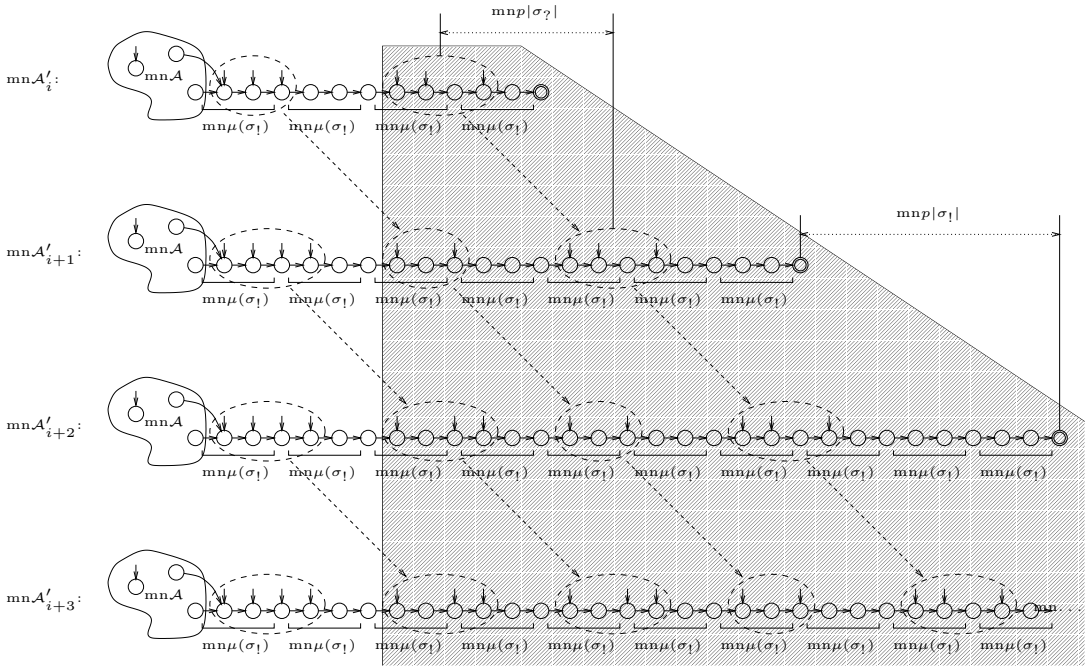

Figure 7.9: Initial states that are provably robust ($|\sigma_!| \geq |\sigma_?|$).

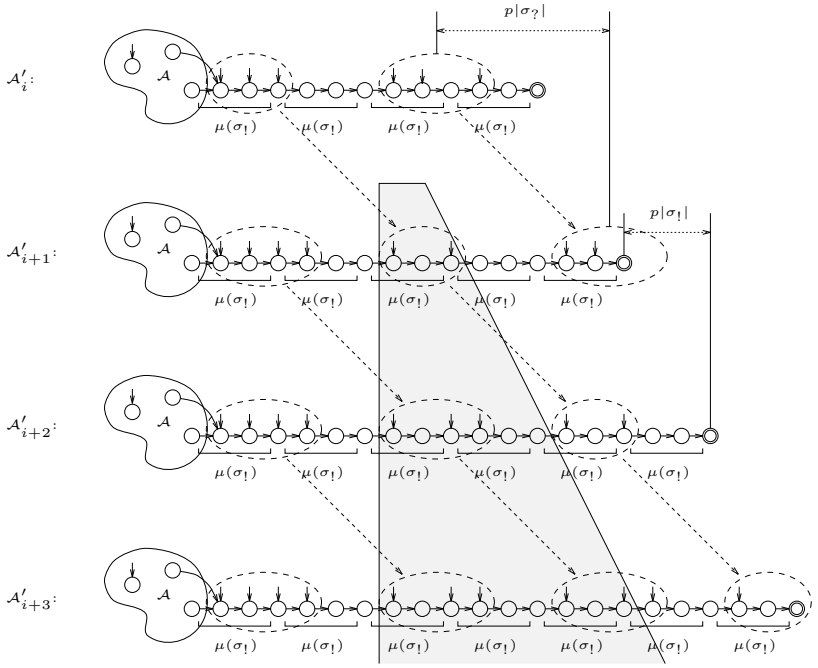

Figure 7.10: Initial states that are provably robust ($|\sigma_!| < |\sigma_?|$).

The notion of robustness allows to capture a part of the redundancy between the initial states that are present on the tails of the different $\mathcal{A}'_i$. Indeed, since for every $i \in \mathbf{N}_0$, all the robust initial states of $\mathcal{A}'_i$ can be p-shifted in order to become initial states of $\mathcal{A}'_{i+1}$, there will be similar patterns of initial states repeated on the tails of the $\mathcal{A}'_i$. In order to analyze precisely the nature of those patterns, it is useful to define a lower bound on the indices of the $\mathcal{A}'_i$ that will be considered: the *initial index* i_0 is defined as the lowest nonzero integer such that $|tail'(i_0 - 1)| \geq p|\sigma_?|$, $|tail'(i_0)| \geq 2p|\sigma_?|$, and $|tail'(i_0 + 1)| \geq 3p|\sigma_?|$.

The set of initial states I'_i of each $\mathcal{A}'_i$ such that $i \geq i_0$ can be partitioned into five subsets:

- $I_i^{\alpha'}$: The initial states that belong to S_0;
- $I_i^{\beta'}$: The initial states that belong to $tail'(i)$ and whose rank is less or equal to $p|\sigma_?|$;
- $I_i^{\gamma'}$: The initial states that belong to $tail'(i)$ and whose rank is greater than $p(i - i_0 + 1)|\sigma_?|$;
- $I_i^{\delta'}$: The initial states that do not belong to $I_i^{\alpha'}$, $I_i^{\beta'}$ or $I_i^{\gamma'}$ and that satisfy the hypotheses of Theorem 7.19;
- $I_i^{\epsilon'}$: The remaining initial states.

This partitioning is illustrated in Figures 7.11 and 7.12.

For each $i \geq i_0$ and $\xi \in \{\alpha, \beta\,\gamma, \delta, \epsilon\}$, we define $\mathcal{A}_i^{\xi'}$ as an automaton identical to $\mathcal{A}'_i$, except for its set of initial states which is made equal to $I_i^{\xi'}$ (formally, we have $\mathcal{A}_i^{\xi'} = (\Sigma'_i, S'_i, \Delta'_i, I_i^{\xi'}, F_i)$). Computing the infinite union of the $L(\mathcal{A}'_i)$ can easily be reduced to computing the infinite union of the $L(\mathcal{A}_i^{\xi'})$ for each $\xi \in \{\alpha, \beta\,\gamma, \delta, \epsilon\}$, as a consequence of the following theorem.

Theorem 7.20 *The language $\bigcup_{i\in\mathbf{N}} L(\mathcal{A}'_i)$ can be expressed in terms of the languages $\bigcup_{i\geq i_0} L(\mathcal{A}_i^{\xi'})$, where $\xi \in \{\alpha, \beta\,\gamma, \delta, \epsilon\}$, and of a finite number of individual $L(\mathcal{A}'_i)$, using finite union.*

Proof

$$\begin{aligned}\bigcup_{i\in\mathbf{N}} L(\mathcal{A}'_i) &= \bigcup_{i<i_0} L(\mathcal{A}'_i) \cup \bigcup_{i\geq i_0} L(\mathcal{A}'_i) \\ &= \bigcup_{i<i_0} L(\mathcal{A}'_i) \cup \bigcup_{\xi\in\{\alpha,\beta\,\gamma,\delta,\epsilon\}} \bigcup_{i\geq i_0} L(\mathcal{A}_i^{\xi'}).\end{aligned}$$

□

It remains to show how to compute finite-state representations of the different $\bigcup_{i\geq i_0} L(\mathcal{A}_i^{\xi'})$. We address each case separately.

The definition of the $\mathcal{A}'_i$ implies

$$I_{i_0-1}^{\alpha'} = I_{i_0}^{\alpha'} = I_{i_0+1}^{\alpha'} = I_{i_0+2}^{\alpha'} = \cdots.$$

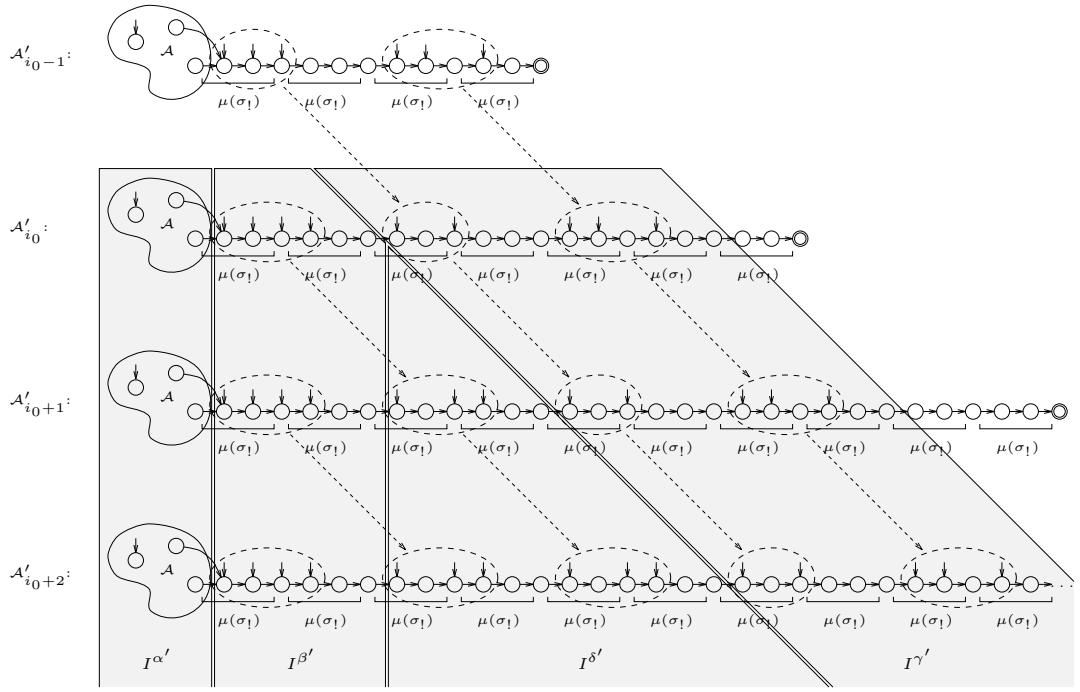

Figure 7.11: Initial states partitioning ($|\sigma_!| \geq |\sigma_?|$).

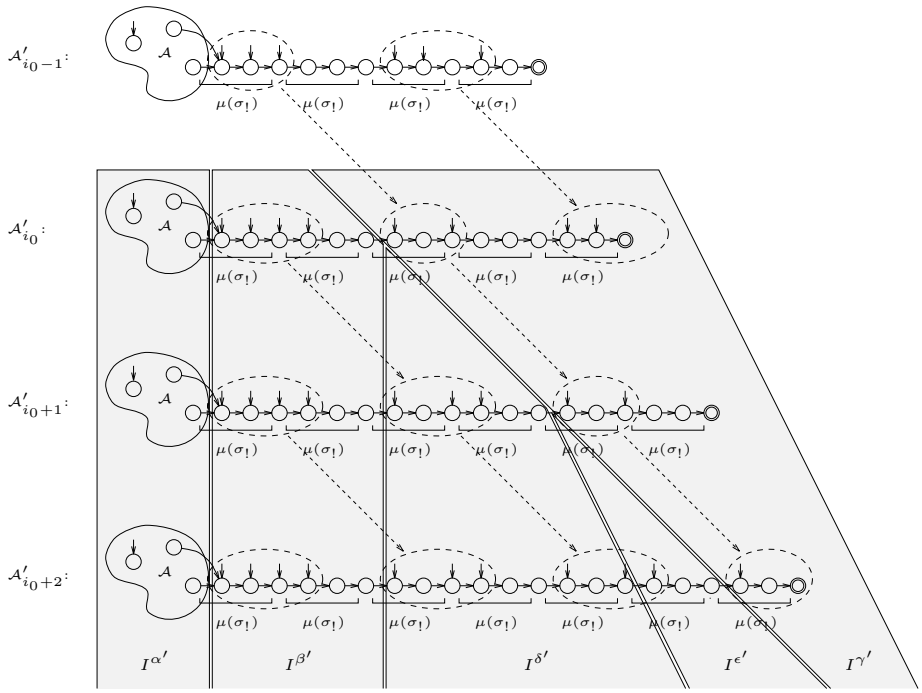

Figure 7.12: Initial states partitioning ($|\sigma_!| < |\sigma_?|$).

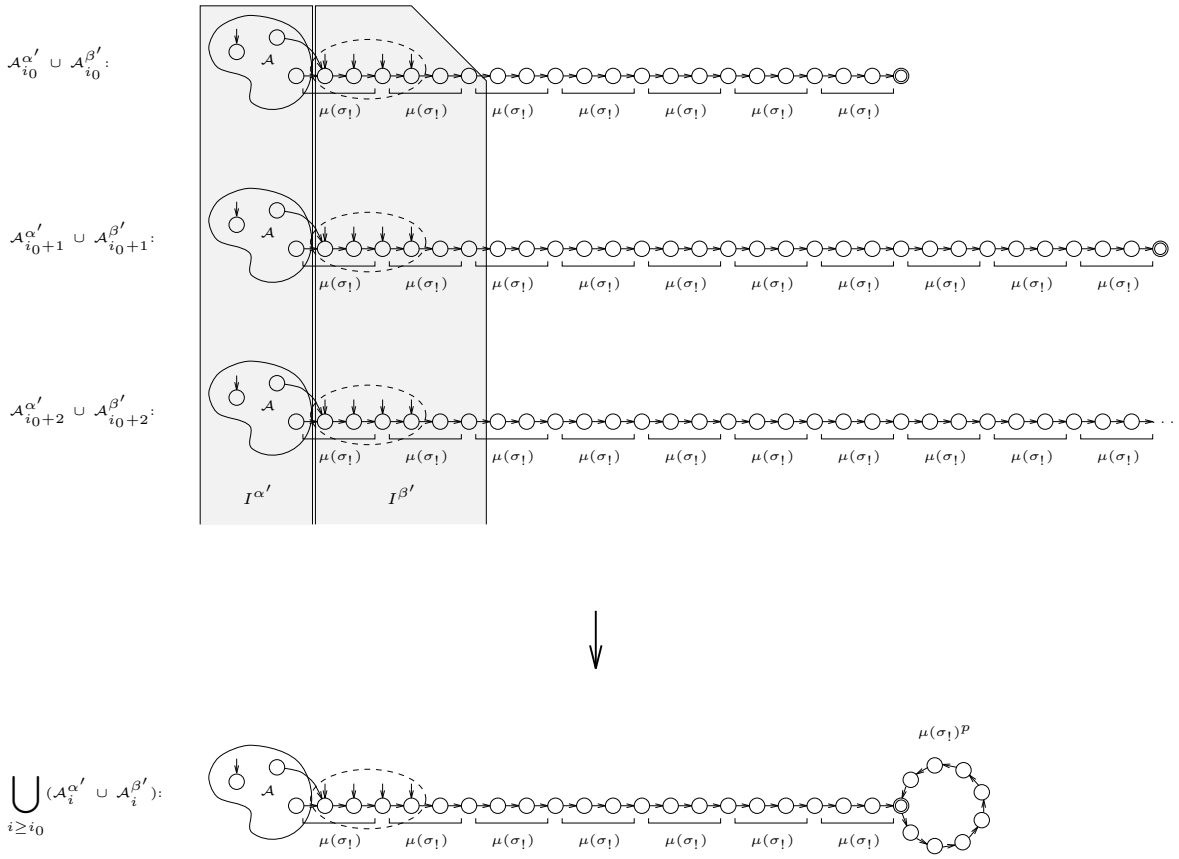

Figure 7.13: Automaton accepting $\bigcup_{i \geq i_0}(L(\mathcal{A}^{\alpha'}_i) \cup L(\mathcal{A}^{\beta'}_i))$.

Since for each $i \geq i_0$, the set $I^{\beta'}_i$ is the set of all the p-shifts that do not already belong to $I^{\alpha'}_i$ of the elements of $I^{\alpha'}_{i-1}$ that can be p-shifted, this result implies

$$I^{\beta'}_{i_0} = I^{\beta'}_{i_0+1} = I^{\beta'}_{i_0+2} = I^{\beta'}_{i_0+3} = \cdots.$$

Hence, for $\xi \in \{\alpha, \beta\}$, the only difference between the $\mathcal{A}^{\xi'}_i$ is the length of their tail. For each $i \in \mathbf{N}$, the tail of $\mathcal{A}'_{i+1}$ contains $p|\sigma_!|$ more states than the one of $\mathcal{A}'_i$, and the sequence of transitions visiting those additional states is labeled by $\mu(\sigma_!)^p$. It follows that we have

$$\bigcup_{i \geq i_0} \Big(L(\mathcal{A}^{\alpha'}_i) \cup L(\mathcal{A}^{\beta'}_i)\Big) = \Big(L(\mathcal{A}^{\alpha'}_{i_0}) \cup L(\mathcal{A}^{\beta'}_{i_0})\Big) \cdot (\mu(\sigma_!)^p)^*.$$

An automaton accepting $\bigcup_{i \geq i_0}(L(\mathcal{A}^{\alpha'}_i) \cup L(\mathcal{A}^{\beta'}_i))$ can easily be constructed by appending a cycle labeled by $\mu(\sigma_!)^p$ to an automaton accepting $L(\mathcal{A}^{\alpha'}_{i_0}) \cup L(\mathcal{A}^{\beta'}_{i_0})$. A possible construction is depicted in Figure 7.13.

In order to compute the infinite union of the $\mathcal{A}^{\gamma'}_i$, we distinguish two situations. First, if $|\sigma_!| \geq |\sigma_?|$, then for every $i \geq i_0$, all the elements of $I^{\gamma'}_i$ are robust as a

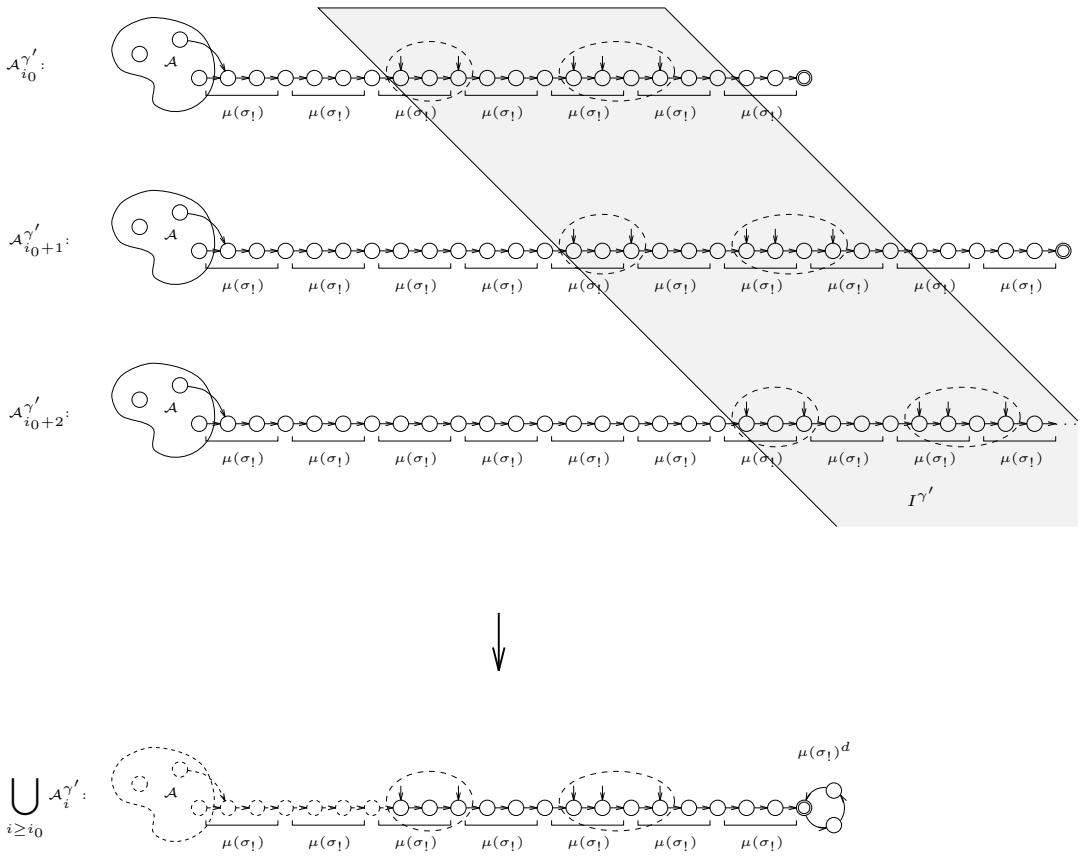

Figure 7.14: Automaton accepting $\bigcup_{i\geq i_0} L(\mathcal{A}^{\gamma'}_i)$ (with $|\sigma_!| \geq |\sigma_?|$).

consequence of Theorem 7.19. This means that $I^{\gamma'}_{i+1}$ is the set of the p-shifts of the elements of $I^{\gamma'}_i$. It follows that $L(\mathcal{A}^{\gamma'}_{i+1}) = L(\mathcal{A}^{\gamma'}_i)\cdot\mu(\sigma_!)^d$, with $d = (|\sigma_!|-|\sigma_?|)(p/|\sigma_!|)$. Therefore, we have

$$\bigcup_{i\geq i_0} L(\mathcal{A}^{\gamma'}_i) = L(\mathcal{A}^{\gamma'}_{i_0}) \cdot (\mu(\sigma_!)^d)^*.$$

An automaton accepting $\bigcup_{i\geq i_0} L(\mathcal{A}^{\gamma'}_i)$ can be constructed by appending a cycle labeled by $\mu(\sigma_!)^d$ to an automaton accepting $L(\mathcal{A}^{\gamma'}_{i_0})$. A possible construction is depicted in Figure 7.14.

If $|\sigma_!| < |\sigma_?|$, then there exists $i_1 \geq i_0$ such that for every $j \geq i_1$, $I^{\gamma'}_j = \emptyset$. Indeed, for every $i \geq i_0$, computing $\mathcal{A}^{\gamma'}_{i+1}$ from $\mathcal{A}^{\gamma'}_i$ increases the rank of each initial state of the tail by $p|\sigma_?|$ while the length of the tail is only increased by $p|\sigma_!|$. An automaton accepting $\bigcup_{i\geq i_0} L(\mathcal{A}^{\gamma'}_i)$ can be constructed by first determining the value of i_1, and then building an automaton accepting the language

$$\bigcup_{i\geq i_0} L(\mathcal{A}^{\gamma'}_i) = \bigcup_{i_0\leq i<i_1} L(\mathcal{A}^{\gamma'}_i).$$

Let us now discuss the computation of the union of the $L(\mathcal{A}^{\delta'}_i)$. We have $I^{\delta'}_{i_0} = \emptyset$.

For every $i > i_0$, all the initial states that belong to $I_i^{\delta'}$ are robust by definition. Moreover, each of these initial states is a p-shift of an initial state belonging either to $I_{i-1}^{\beta'}$ or to $I_{i-1}^{\delta'}$. Since all the $I_i^{\beta'}$ such that $i > i_0$ are equal to each other and contain only robust states, we have that for every $i > i_0$, $I_i^{\delta'} \subseteq I_{i+1}^{\delta'}$. Let us distinguish two cases. If $|\sigma_!| \geq |\sigma_?|$, then all the initial states in $I_{i+1}^{\delta'}$ that do not belong to $I_i^{\delta'}$ are exactly the states obtained as a result of p-shifting $i - i_0 + 1$ times the states in $I_{i_0+1}^{\delta'}$. It follows that

$$L(\mathcal{A}_{i+1}^{\delta'}) = L(\mathcal{A}_i^{\delta'}) \cdot \mu(\sigma_!)^d \cup L(\mathcal{A}_i^{\delta'}) \cdot \mu(\sigma_!)^p,$$

with $d = (|\sigma_!| - |\sigma_?|)(p/|\sigma_!|)$. Therefore, we have

$$\bigcup_{i \geq i_0} L(\mathcal{A}_i^{\delta'}) = L(\mathcal{A}_{i_0+1}^{\delta'}) \cdot (\mu(\sigma_!)^d)^* \cdot (\mu(\sigma_!)^p)^*.$$

An automaton accepting $\bigcup_{i \geq i_0} L(\mathcal{A}_i^{\delta'})$ can be constructed by appending successively cycles labeled by $\mu(\sigma_!)^d$ and then by $\mu(\sigma_!)^p$ to an automaton accepting $L(\mathcal{A}_{i_0+1}^{\delta'})$. A possible construction is depicted in Figure 7.15.

If $|\sigma_!| < |\sigma_?|$, then the situation is more tricky. Even though all the initial states that belong to $I_i^{\delta'}$ can be p-shifted for $i > i_0$, their p-shifts do not necessarily belong to $I_{i+1}^{\delta'}$ (as it is the case when $|\sigma_!| \geq |\sigma_?|$).

For each $i > i_0$, all the elements of $I_i^{\delta'}$ have a rank between $p|\sigma_?|+1$ and l_i, where $l_i = \min(|tail'(i)| - p|\sigma_?|, p(i - i_0 + 1)|\sigma_?|)$. We define the *right index* r_i of i as the greatest integer such that $r_i p|\sigma_?| \leq l_i$, and the *right block* $I_i^{\delta'_R}$ of $I_i^{\delta'}$ as the set

$$I_i^{\delta'_R} = \{s \in I_i^{\delta'} \mid (r_i - 1)p|\sigma_?| < rank(s) \leq r_i p|\sigma_?|\}.$$

Intuitively, this corresponds to slicing the tail of $\mathcal{A}'_i$ into groups of $p|\sigma_?|$ consecutive states. The right block of $\mathcal{A}_i^{\delta'}$ is the subset of initial states belonging to the rightmost group in which each state satisfies the hypotheses of Theorem 7.19. The notion of right block is illustrated in Figure 7.16. The automaton $\mathcal{A}_i^{\delta'_R}$ is defined as being identical to $\mathcal{A}_i^{\delta'}$ except for its set of initial states which is made equal to the right block of $\mathcal{A}_i^{\delta'}$. Formally, we have $\mathcal{A}_i^{\delta'_R} = (\Sigma'_i, S'_i, \Delta'_i, I_i^{\delta'_R}, F'_i)$.

The usefulness of the notion of right block is that for every $i > i_0$, each element of $I_{i+1}^{\delta'}$ either belongs to $I_i^{\delta'}$, or is a p-shift of a state belonging to $I_i^{\delta'_R}$. Moreover, for each state s in $I_{i+1}^{\epsilon'}$, there exists $j \in \{1, 2\}$ such that s can be obtained by p-shifting j times an element of $I_{i-j+1}^{\delta'_R}$. Therefore, we have

$$L(\mathcal{A}_{i+1}^{\delta'}) \cup L(\mathcal{A}_{i+1}^{\epsilon'}) = L(\mathcal{A}_i^{\delta'}) \cdot \mu(\sigma_!)^p \cup \sigma^p\left(L(\mathcal{A}_i^{\delta'_R})\right) \cup \sigma^{2p}\left(L(\mathcal{A}_i^{\delta'_R})\right).$$

From this expression, we deduce

$$\bigcup_{i \geq i_0} \left(L(\mathcal{A}_i^{\delta'}) \cup L(\mathcal{A}_i^{\epsilon'})\right) = \bigcup_{i > i_0} L(\mathcal{A}_i^{\delta'_R}) \cdot (\mu(\sigma_!)^p)^* \cup \sigma^p\left(\bigcup_{i > i_0} L(\mathcal{A}_i^{\delta'_R})\right) \cup \sigma^{2p}\left(\bigcup_{i > i_0} L(\mathcal{A}_i^{\delta'_R})\right),$$

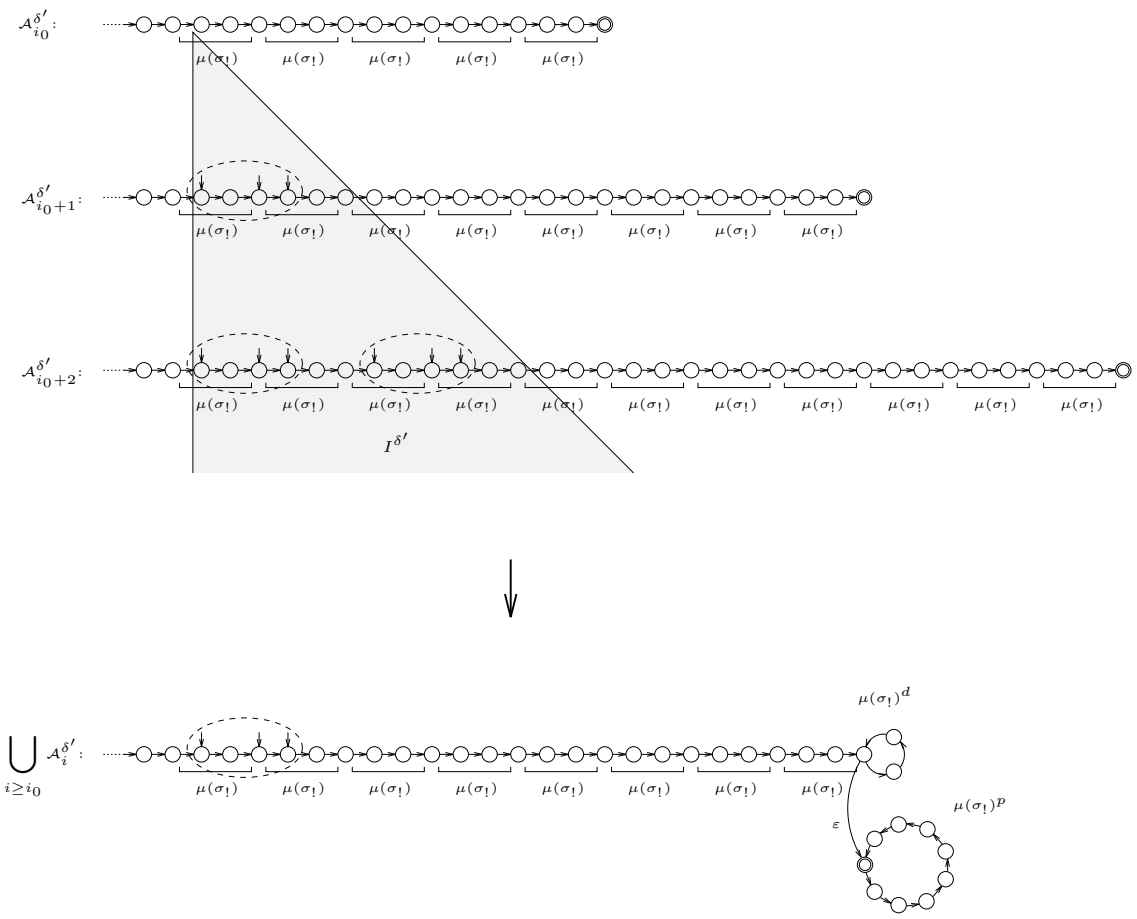

Figure 7.15: Automaton accepting $\bigcup_{i \geq i_0} L(\mathcal{A}^{\delta'}_i)$ (with $|\sigma_!| \geq |\sigma_?|$).

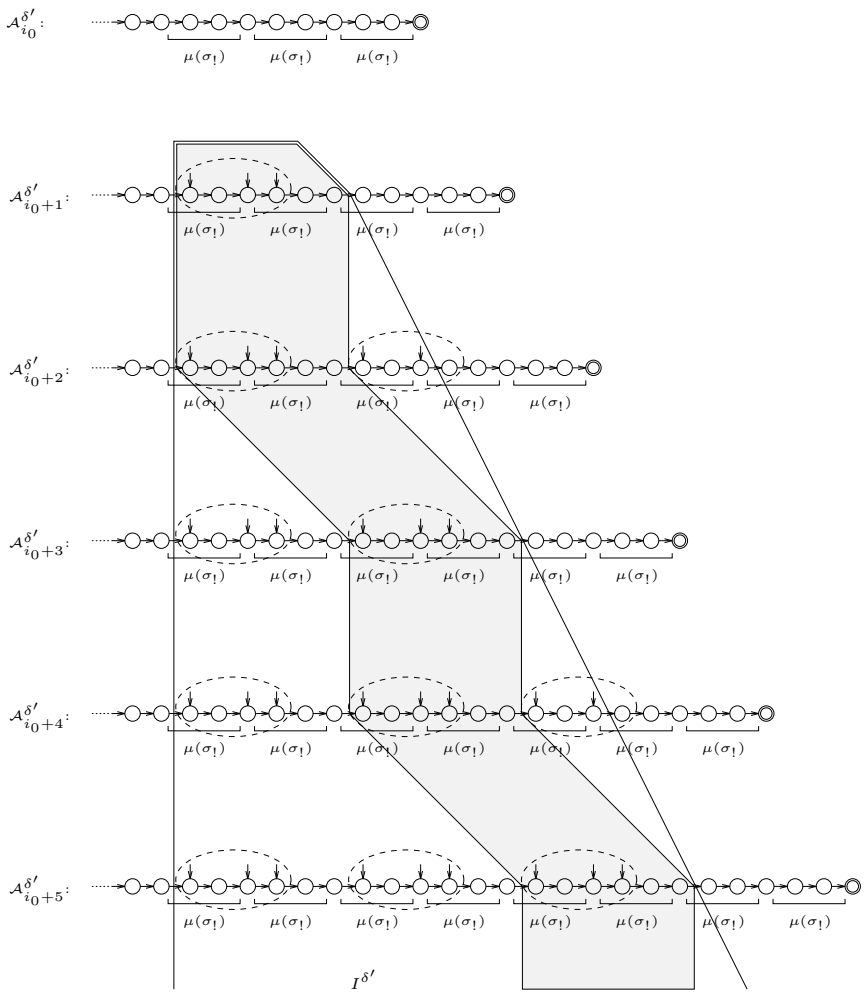

Figure 7.16: Right blocks.

which reduces the problem of computing the infinite unions of the $\mathcal{A}_i^{\delta'}$ and of the $\mathcal{A}_i^{\epsilon'}$ to the computation of the infinite union of the $\mathcal{A}_i^{\delta'_R}$.

Let us show how to solve the latter problem. For every $i > i_0$, we have

$$L(\mathcal{A}_i^{\delta'_R}) = \begin{cases} L(\mathcal{A}_i^{\delta'}) & \text{if } i = i_0 + 1; \\ \sigma^p\left(L(\mathcal{A}_{i-1}^{\delta'_R})\right) & \text{if } i_0 + 1 < i < i_2; \\ L(\mathcal{A}_{i-1}^{\delta'_R}) \cdot \mu(\sigma_!)^p & \text{if } i \geq i_2 \text{ and } \left\lfloor \frac{|tail'(i)|}{p|\sigma_?|} \right\rfloor = \left\lfloor \frac{|tail'(i-1)|}{p|\sigma_?|} \right\rfloor; \\ \sigma^p\left(L(\mathcal{A}_{i-1}^{\delta'_R})\right) & \text{if } i \geq i_2 \text{ and } \left\lfloor \frac{|tail'(i)|}{p|\sigma_?|} \right\rfloor = \left\lfloor \frac{|tail'(i-1)|}{p|\sigma_?|} \right\rfloor + 1, \end{cases} \tag{7.1}$$

where i_2 is the smallest integer such that $i_2 > i_0$ and $|tail'(i_2)| < (i_2 - i_0 + 2)p|\sigma_?|$. Intuitively, i_2 is the smallest integer such that $I_{i_2}^{\delta'_R}$ is different from the result of p-shifting $i_2 - i_0 - 1$ times the states in $I_{i_0+1}^{\delta'_R}$.

The infinite sequence of languages $L(\mathcal{A}_{i_0+1}^{\delta'_R}), L(\mathcal{A}_{i_0+2}^{\delta'_R}), L(\mathcal{A}_{i_0+3}^{\delta'_R}), \ldots$ is ultimately periodic. Indeed, for every $i \geq i_2$, applying p times Equation 7.1 yields $L(\mathcal{A}_{i+p}^{\delta'_R}) = L(\mathcal{A}_i^{\delta'_R})$. As a consequence, we have

$$\bigcup_{i>i_0} L(\mathcal{A}_i^{\delta'_R}) = \bigcup_{i_0<i<i_2+p} L(\mathcal{A}_i^{\delta'_R}).$$

An algorithm[2] summarizing the different steps of the computation of $\sigma^*(U)$ is given in Figures 7.18, 7.19 and 7.20. This algorithm relies on a subroutine which is given in Figure 7.17. The correctness of the subroutine and of the algorithm is established by the following theorems.

Theorem 7.21 *Let q be a queue of alphabet Σ, $w \in \Sigma^*$ be a word, and $\mathcal{A}$ be a QDD representing the set $U \subseteq \Sigma^*$. APPEND-LOOP($\mathcal{A}, w$) is a QDD representing the set $U \cdot w^*$.*

Proof Immediate. □

Theorem 7.22 *Let q be a queue of alphabet Σ, σ be a sequence of elementary operations involving q, and $\mathcal{A}$ be a QDD representing the set $U \subseteq \Sigma^*$. APPLY-STAR-ONE($\mathcal{A}, \sigma$) is a QDD representing the set $\sigma^*(U)$.*

Proof The algorithm of Figures 7.18, 7.19 and 7.20 is a straightforward implementation of the method described in this section. There is however a small optimization: a variable l is introduced in Lines 47–59 in order to simplify the computation of the right blocks. The principle of this optimization is to ensure that at any time, the current right block may be p-shifted inside the region containing $I^{\delta'}$ if and only if

[2]In this algorithm, the union operator "∪" applied to QDDs denotes repeated calls to the function UNION introduced in Section 6.2.4.

function APPEND-LOOP(QDD $(\Sigma, S, \Delta, I, F)$, word $a_1 a_2 \cdots a_m$) : QDD

1: **var** $s_1, s_2, \ldots, s_m$: states;
2: **begin**
3: **if** $m = 0$ **then return** $(\Sigma, S, \Delta, I, F)$;
4: **let** $s_1, s_2, \ldots, s_m \notin S$;
5: $S := S \cup \{s_1, s_2, \ldots, s_m\}$;
6: $\Delta := \Delta \cup \{(s, a_1, s_1) \mid s \in F\} \cup \{(s_{i-1}, a_i, s_i) \mid 1 < i \leq m\} \cup \{(s_m, a_1, s_1)\}$;
7: $F := F \cup \{s_m\}$;
8: **return** $(\Sigma, S, \Delta, I, F)$
9: **end**.

Figure 7.17: Subroutine APPEND-LOOP.

$l \geq 0$. Intuitively, the value of l is related to the distance (in terms of rank difference) between the current right block and the leftmost edge of that region. □

There is an important corollary to Theorem 7.22.

Corollary 7.23 *Let q be a queue of alphabet Σ, σ be a sequence of elementary operations involving q, and $U \subseteq \Sigma^*$ be a recognizable set of queue contents. The set $\sigma^*(U)$ is recognizable.*

7.3.2 Systems with Any Number of Queues

In the case of systems having more than one queue, one cannot hope to obtain a result similar to Corollary 7.23. The reason is that iterating sequences of elementary operations involving more than one queue can generate non-recognizable sets of queue-set contents. For instance, in the case of a system with two queues q_1 and q_2 whose alphabets are respectively $\{a_1\}$ and $\{a_2\}$, the closure of the sequence $q_1!a_1; q_2!a_2$ transforms the set of queue-set contents $U = \{(\varepsilon, \varepsilon)\}$ into the set $U' = \{(a_1^n, a_2^n) \mid n \in \mathbf{N}\}$, whose sequential encoding is not regular.

The first step is to characterize precisely the sequences of queue operations whose closure preserves the recognizability of sets of queue-set contents (and therefore the possibility of representing these sets by QDDs). Our characterization is based on the following notion.

Definition 7.24 *Let σ be a sequence of queue operations involving only one queue q, and let Σ be the alphabet of q. The sequence σ is* counting *if one of the following conditions is satisfied:*

function APPLY-STAR-ONE(QDD $(\Sigma, S, \Delta, I, F)$, **sequence of** queue operations σ) : QDD

var $\mathcal{A}$: **array**$[0, 1, \ldots]$ **of** QDDs;
  I^α : **array**$[0, 1, \ldots]$ **of sets of** states;
  S_0, S_1, S_2 : **sets of** states;
  $\mathcal{A}'_{i_0}, \mathcal{A}'_{i_0+1}, \mathcal{A}^{\alpha\beta'}_{i_0}, \mathcal{A}^{\alpha\beta'}, \mathcal{A}^{\gamma'}_{i_0}, \mathcal{A}^{\gamma'}, \mathcal{A}^{\gamma''}, \mathcal{A}^{\delta'}_{i_0+1}, \mathcal{A}^{\delta\epsilon'}, \mathcal{A}^{\delta'_R}, \mathcal{A}^{\delta''_R}, \mathcal{A}', \mathcal{A}''$: QDDs;
  $(\Sigma'_{i_0}, S'_{i_0}, \Delta'_{i_0}, I'_{i_0}, F'_{i_0}), (\Sigma'_{i_0+1}, S'_{i_0+1}, \Delta'_{i_0+1}, I'_{i_0+1}, F'_{i_0+1})$: QDDs;
  $n_1, n_2, p, b, d, i, i_0, i_1, i_2, l$: integers;
begin
  $\mathcal{A}[0] := (\Sigma, S, \Delta, I, F) :=$ NORMALIZE$((\Sigma, S, \Delta, I, F))$;
  if $|\sigma_?| = 0$ **then return** APPEND-LOOP$(\mathcal{A}[0], \mu(\sigma_!))$;
  $I^\alpha[0] := I$;
  $S_0 := S$;
  $n_1 := 0$;
  repeat
    $n_1 := n_1 + 1$;
    $\mathcal{A}[n_1] := (\Sigma, S, \Delta, I, F) :=$ APPLY-ONE$(\mathcal{A}[n_1 - 1], \sigma)$;
    $I^\alpha[n_1] := I \cap S_0$
  until there exists n_2 **such that** $0 \leq n_2 < n_1 \wedge I^\alpha[n_1] = I^\alpha[n_2]$;
  if $|\sigma_!| = 0$ **then return** $\bigcup_{0 \leq i \leq n_1} \mathcal{A}[i]$;
  $p := \mathrm{lcm}(n_1 - n_2, |\sigma_!|)$;
  $b := |\sigma_!|\lceil n_2/|\sigma_!|\rceil$;
  $i_0 := \left\lceil 3\dfrac{|\sigma_?|}{|\sigma_!|} - \dfrac{b}{p} \right\rceil + 1$;
  $\mathcal{A}'_{i_0} := (\Sigma'_{i_0}, S'_{i_0}, \Delta'_{i_0}, I'_{i_0}, F'_{i_0}) :=$ APPLY-ONE$(\mathcal{A}[n_1], \sigma^{b-n_1+i_0p})$;
  $\mathcal{A}'_{i_0+1} := (\Sigma'_{i_0+1}, S'_{i_0+1}, \Delta'_{i_0+1}, I'_{i_0+1}, F'_{i_0+1}) :=$ APPLY-ONE$(\mathcal{A}'_{i_0}, \sigma^p)$;
  $S_1 := \{s' \in S'_{i_0} \mid (\exists s \in S_0, w \in \Sigma^*)((s, w, s') \in \Delta'^*_{i_0} \wedge |w| \leq p|\sigma_?|)\}$;
  $S_2 := \{s' \in S'_{i_0+1} \mid (\exists s \in S_0, w \in \Sigma^*)((s, w, s') \in \Delta'^*_{i_0+1} \wedge |w| \leq 2p|\sigma_?|)\}$;
  $\mathcal{A}^{\alpha\beta'}_{i_0} := (\Sigma'_{i_0}, S'_{i_0}, \Delta'_{i_0}, I'_{i_0} \cap S_1, F'_{i_0})$;
  $\mathcal{A}^{\alpha\beta'} :=$ APPEND-LOOP$(\mathcal{A}^{\alpha\beta'}_{i_0}, \mu(\sigma_!)^p)$;

(...)

Figure 7.18: Image of a single-queue QDD by the closure of a sequence of queue operations.

(...)

28: $\mathcal{A}_{i_0}^{\gamma'} := (\Sigma'_{i_0}, S'_{i_0}, \Delta'_{i_0}, I'_{i_0} \setminus S_1, F'_{i_0})$;
29: $\mathcal{A}_{i_0+1}^{\delta'} := (\Sigma'_{i_0+1}, S'_{i_0+1}, \Delta'_{i_0+1}, I'_{i_0+1} \cap (S_2 \setminus S_1), F'_{i_0+1})$;
30: **if** $|\sigma_!| \geq |\sigma_?|$ **then**
31: **begin**
32: $d := \frac{p}{|\sigma_!|}(|\sigma_!| - |\sigma_?|)$;
33: $\mathcal{A}^{\gamma'} :=$ APPEND-LOOP$(\mathcal{A}_{i_0}^{\gamma'}, \mu(\sigma_!)^d)$;
34: $\mathcal{A}^{\delta\epsilon'} :=$ APPEND-LOOP(APPEND-LOOP$(\mathcal{A}_{i_0+1}^{\delta'}, \mu(\sigma_!)^d), \mu(\sigma_!)^p)$
35: **end**
36: **else**
37: **begin**
38: $i_1 := \left\lceil \frac{(b/p)|\sigma_!| + (i_0 - 1)|\sigma_?|}{|\sigma_?| - |\sigma_!|} \right\rceil$;
39: $\mathcal{A}^{\gamma'} := \mathcal{A}^{\gamma''} := \mathcal{A}_{i_0}^{\gamma'}$;
40: **for** $i := i_0 + 1$ **to** $i_1 - 1$ **do**
41: **begin**
42: $\mathcal{A}^{\gamma''} :=$ APPLY-ONE$(\mathcal{A}^{\gamma''}, \sigma^p)$;
43: $\mathcal{A}^{\gamma'} :=$ UNION$(\mathcal{A}^{\gamma'}, \mathcal{A}^{\gamma''})$
44: **end**;
45: $i_2 := 1 + \max(i_0, \left\lceil \frac{(b/p)|\sigma_!| + (i_0 - 2)|\sigma_?|}{|\sigma_?| - |\sigma_!|} \right\rceil)$;
46: $\mathcal{A}^{\delta'_R} := \mathcal{A}^{\delta''_R} := \mathcal{A}_{i_0+1}^{\delta'}$;
47: $l := (b + (i_0 + 2)p)|\sigma_!| - 4p|\sigma_?|$;
48: **for** $i := i_0 + 2$ **to** $i_2 + p - 1$ **do**
49: **if** $l \geq 0$ **then**
50: **begin**
51: $\mathcal{A}^{\delta''_R} :=$ APPLY-ONE$(\mathcal{A}^{\delta''_R}, \sigma^p)$;
52: $\mathcal{A}^{\delta'_R} :=$ UNION$(\mathcal{A}^{\delta'_R}, \mathcal{A}^{\delta''_R})$;
53: $l := l + p(|\sigma_!| - |\sigma_?|)$
54: **end**

(...)

Figure 7.19: Image of a single-queue QDD by the closure of a sequence of queue operations (continued).

(...)

55: **else**
56: **begin**
57: $\mathcal{A}^{\delta''_R} :=$ APPLY-ONE$(\mathcal{A}^{\delta''_R}, \sigma_!^p)$;
58: $\mathcal{A}^{\delta'_R} :=$ UNION$(\mathcal{A}^{\delta'_R}, \mathcal{A}^{\delta''_R})$;
59: $l := l + p|\sigma_!|$
60: **end**;
61: $\mathcal{A}^{\delta\epsilon'} :=$ APPEND-LOOP$(\mathcal{A}^{\delta'_R}, \mu(\sigma_!)^p)$;
62: $\mathcal{A}^{\delta\epsilon'} :=$ UNION$(\mathcal{A}^{\delta\epsilon'}$, APPLY-ONE$(\mathcal{A}^{\delta'_R}, \sigma^p))$;
63: $\mathcal{A}^{\delta\epsilon'} :=$ UNION$(\mathcal{A}^{\delta\epsilon'}$, APPLY-ONE$(\mathcal{A}^{\delta'_R}, \sigma^{2p}))$
64: **end**;
65: $\mathcal{A}' :=$ UNION$\left(\bigcup_{0\le i<i_0}\right.$ APPLY-ONE$(\mathcal{A}[n_1], \sigma^{b-n_1+pi})$, $\left.\bigcup_{\xi\in\{\alpha\beta,\gamma,\delta\epsilon\}} \mathcal{A}^{\xi'}\right)$;
66: $\mathcal{A}'' :=$ UNION$\left(\bigcup_{0\le i<b}\right.$ APPLY-ONE$(\mathcal{A}[0], \sigma^i)$, $\left.\bigcup_{0\le i<p}\right.$ APPLY-ONE$(\mathcal{A}', \sigma^i))$;
67: **return** $\mathcal{A}''$
68: **end**.

Figure 7.20: Image of a single-queue QDD by the closure of a sequence of queue operations (continued).

- $|\Sigma| > 1$ *and* $|\sigma_!| > 0$,
- $|\Sigma| = 1$ *and* $|\sigma_!| > |\sigma_?|$.

Intuitively, a sequence σ of operations that satisfies the previous definition is called counting since in that case, there are sets U of queue contents for which the number k of applications of σ to U can be determined by examining the language $\sigma^k(U)$, for every $k \geq 0$. Formally, this property is expressed by the following theorem.

Theorem 7.25 *Let σ be a sequence of queue operations involving only one queue q, and let Σ be the alphabet of q. The sequence σ is counting if and only if there exists a recognizable set $U \subseteq \Sigma^*$ of queue contents such that $\sigma^{k_1}(U) \neq \sigma^{k_2}(U)$ for all $k_1, k_2 \in \mathbf{N}$ such that $k_1 \neq k_2$.*

Proof

- *If σ is counting, then there exists $U \subseteq \Sigma^*$ such that U is recognizable and $\sigma^{k_1}(U) \neq \sigma^{k_2}(U)$ for all $k_1, k_2 \in \mathbf{N}$ such that $k_1 \neq k_2$.* If $|\Sigma| = 1$, then let a denote the only symbol in Σ. Choosing $U = \{\mu(\sigma_?)\}$ yields for every $k \in \mathbf{N}$

$$\sigma^k(U) = \{a^{(1-k)|\sigma_?|+k|\sigma_!|)}\}.$$

  If $|\Sigma| > 1$, then let $a \in \Sigma$ be a symbol different from $\mu(\sigma_?)[1]$. Choosing $U = \mu(\sigma_?)^* \cdot a$ yields for every $k \in \mathbf{N}$

$$\sigma^k(U) = \mu(\sigma_?)^* \cdot a \cdot \mu(\sigma_!)^k.$$

- *If σ is not counting, then for all the recognizable sets $U \subseteq \Sigma^*$, there exist $k_1, k_2 \in \mathbf{N}$ such that $k_1 \neq k_2$ and $\sigma^{k_1}(U) = \sigma^{k_2}(U)$.* If $|\Sigma| = 1$, then we must have $|\sigma_!| \leq |\sigma_?|$. The set U is a context-free language over a one-letter alphabet. It is well known [Mat94] that such languages can be expressed as a finite union of languages U_i of the form $a^{l_i} \cdot (a^{l'_i})^*$, where a is the only symbol of Σ, and $l_i, l'_i \in \mathbf{N}$. For every $k \in \mathbf{N}$, the language $\sigma^k(U_i)$ is either empty or of the form $a^{l''_i} \cdot (a^{l'_i})^*$, with $l''_i \leq \max(l_i, |\sigma_?| + l'_i)$ (this is easily established by induction on k). The set of possible $\sigma^k(U_i)$ is thus finite, hence such is the set of possible $\sigma^k(U)$.

  If $|\Sigma| > 1$, then we must have $\sigma = \sigma_?$. Let $\mathcal{A}_0$ be a QDD representing U. For every $k > 0$, let $\mathcal{A}_k$ be the QDD returned by APPLY-ONE$(\mathcal{A}_{k-1}, \sigma)$. For every $k \in \mathbf{N}$, the QDD $\mathcal{A}_k$ is identical to $\mathcal{A}_0$ except (possibly) for its set of initial states. As there are only a finite number of possible sets of initial states, the infinite sequence $\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2, \ldots$ is ultimately periodic, and therefore there exist $k_1, k_2 \in \mathbf{N}$ such that $L(\mathcal{A}_{k_1}) = L(\mathcal{A}_{k_2})$ and $k_1 \neq k_2$.

□

The notion of counting sequences leads to a necessary condition on sequences of operations σ (involving any number of queues) whose closure preserves the recognizable nature of sets of queue-set contents. Roughly speaking, the idea is that if σ admits two projections $\sigma|_i$ and $\sigma|_j$ $(i \neq j)$ that are both counting, then there exists a set U of queue-set contents such that for every $k \in \mathbf{N}$, the sequential encoding of $\sigma^k(U)$ represents the value of k twice (once in $\sigma^k(U)|_i$ and once in $\sigma^k(U)|_j$). Since k is unbounded, it is impossible for a finite-state machine to check whether the two represented values coincide, which implies that $\sigma^k(U)$ is not recognizable. Formally, we have the following theorem.

Theorem 7.26 *Let σ be a sequence involving the queues $q_1, q_2, \ldots, q_n$ $(n \geq 1)$ and let $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ be the alphabets of those queues. If for every recognizable set $U \subseteq \Sigma_1^* \times \Sigma_2^* \times \cdots \times \Sigma_n^*$, the set $\sigma^*(U)$ is recognizable, then there do not exist $i, j \in \mathbf{N}$ such that $1 \leq i < j \leq n$ and such that $\sigma|_i$ and $\sigma|_j$ are counting sequences.*

Proof The proof is by contradiction. Suppose that there exist $i, j \in \mathbf{N}$ such that $1 \leq i < j \leq n$ and such that $\sigma|_i$ and $\sigma|_j$ are counting. We show that there exists a recognizable set $U \subseteq \Sigma_1^* \times \Sigma_2^* \times \cdots \times \Sigma_n^*$ such that $\sigma^*(U)$ is not recognizable. For each $k \in \{1, 2, \ldots, n\}$, we define the recognizable set of queue contents $U_k \subseteq \Sigma_k^*$ as follows:

- If $k \in \{i, j\}$ and $|\Sigma_k| > 1$, then $U_k = \mu(\sigma|_{k?})^* \cdot a_k$, where $a_k \in \Sigma_k$ is a symbol different from $\mu(\sigma|_{k?})[1]$;
- If $k \in \{i, j\}$ and $|\Sigma_k| = 1$, then $U_k = \{\mu(\sigma|_{k?})\}$;
- If $k \notin \{i, j\}$, then $U_k = \Sigma_k^*$.

Let $U = U_1 \times U_2 \times \cdots \times U_n$ and $U' = \sigma^*(U)$. Let us prove by contradiction that U' is not recognizable. If U' is recognizable, then its sequential encoding $L' = E_S(U')$ is regular. Let $L'_{ij} = L'|_i \cdot L'|_j$. We have

$$L'_{ij} = \{u_i \cdot (v_i)^l \cdot u_j \cdot (v_j)^l \mid l \in \mathbf{N} \wedge u_i \in V_i \wedge u_j \in V_j\},$$

where for each $k \in \{i, j\}$, V_k and v_k are defined as follows:

- If $|\Sigma_k| > 1$, then $V_k = \mu(\sigma|_{k?})^* \cdot a_k$ and $v_k = \mu(\sigma|_{k!})$;
- If $|\Sigma_k| = 1$, then $V_k = \{\mu(\sigma|_{k?})\}$ and $v_k = (a'_k)^{m_k}$, where a'_k is the only symbol in Σ_k and $m_k = |(\sigma|_{k!})| - |(\sigma|_{k?})|$.

Since L'_{ij} is not regular, L' is not regular. Therefore, U' is not recognizable. □

The next step is to show that the condition expressed by Theorem 7.26 is not only necessary but also sufficient, i.e., that for any sequence σ satisfying the hypotheses of this theorem and recognizable set U of queue-set contents, the set $\sigma^*(U)$

is recognizable. Our proof of this result is constructive and can be translated into an algorithm for computing a QDD representing $\sigma^*(U)$, given σ and a QDD representing U. Roughly speaking, the proof is based on the fact that for every queue q_i such that $\sigma|_i$ is not counting, applying an unbounded number of times the sequence $\sigma|_i$ to an arbitrary subset of Σ_i^* is always equivalent to applying it only a bounded number of times (as a consequence of Theorem 7.25). Applying an unbounded number of times the sequence σ to U can thus be reduced to applying a bounded number of times each $\sigma|_i$ that is not counting and applying an unbounded number of times the only (if any) $\sigma|_i$ that is counting. The latter problem can then be solved as a consequence of Theorem 7.22.

Formally, the sufficient condition is stated as follows.

Theorem 7.27 *Let σ be a sequence involving the queues $q_1, q_2, \ldots, q_n$ ($n \geq 1$) and let $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ be the alphabets of those queues. If there exists at most one $i \in \mathbf{N}$ such that $1 \leq i \leq n$ and such that $\sigma|_i$ is counting, then for every recognizable set $U \subseteq \Sigma_1^* \times \Sigma_2^* \times \cdots \times \Sigma_n^*$, the set $\sigma^*(U)$ is recognizable.*

Proof If $U \subseteq \Sigma_1^* \times \Sigma_2^* \times \cdots \times \Sigma_n^*$ is recognizable, then it can be expressed as a finite union $\bigcup_{1 \leq j \leq q} U_{j0}$ where $q \geq 0$, each U_{j0} is of the form $U_{j01} \times U_{j02} \times \cdots \times U_{j0n}$, and for every $l \in \{1, 2, \ldots, n\}$, U_{j0l} is a regular subset of Σ_l^*. We define $U_0 = U$ and for every $k \in \mathbf{N}_0$, $U_k = \sigma(U_{k-1})$. For every $k \in \mathbf{N}$, we have

$$U_k = \bigcup_{1 \leq j \leq q} (\sigma|_1)^k(U_{j01}) \times (\sigma|_2)^k(U_{j02}) \times \cdots \times (\sigma|_n)^k(U_{j0n}).$$

Let i be the only integer such that $\sigma|_i$ is counting (if any), and let $i = n+1$ if there is no such integer. Applying Theorem 7.25, we obtain that for every $l \in \{1, 2, \ldots, n\}$ such that $l \neq i$ and $j \in \{1, 2, \ldots, q\}$, there exist $b_{jl} \in \mathbf{N}$ and $p_{jl} \in \mathbf{N}_0$ such that for every $k \geq b_{jl}$, $(\sigma|_l)^k(U_{j0l}) = (\sigma|_l)^{k+p_{jl}}(U_{j0l})$. Defining $b = \max_{j,l \neq i} b_{jl}$ and $p = \mathrm{lcm}_{j,l \neq i} p_{jl}$, we obtain that for every $j \in \{1, 2, \ldots, q\}$, $k \geq b$ and $l \in \{1, 2, \ldots, n\}$ such that $l \neq i$, $(\sigma|_l)^k(U_{j0l}) = (\sigma|_l)^{k+p}(U_{j0l})$. There are two possible situations.

- *If $i > n$.* Then, for every $k > b$, we have $U_{k+p} = U_k$. The set $\sigma^*(U)$ can thus be expressed as a finite union of recognizable sets:

$$\sigma^*(U) = \bigcup_{0 \leq k < b+p} U_k.$$

- *If $1 \leq i \leq n$.* Then, we have

$$\sigma^*(U) = \bigcup_{0 \leq k < b} \sigma^k(U) \cup \bigcup_{0 \leq k' < p} \sigma^{k'} \left(\bigcup_{k \in \mathbf{N}} \sigma^{b+kp}(U) \right),$$

```
function META?(sequence of queue operations σ, alphabets Σ_1, Σ_2, ..., Σ_n) : {T, F}
1:  var i, j : integers;
2:  begin
3:      j := 0;
4:      for i := 1 to n do
5:          if (|Σ_i| > 1 ∧ |(σ|_{i!})| > 0) ∨ (|Σ_i| = 1 ∧ |(σ|_{i!})| > |(σ|_{i?})|) then
6:              if j = 0 then j := 1
7:              else return F;
8:      return T
9:  end.
```

Figure 7.21: Implementation of META? for sequences of queue operations.

and

$$\bigcup_{k\in\mathbf{N}} \sigma^{b+kp}(U) = \bigcup_{1\le j\le q} (\sigma|_1)^b(U_{j01}) \times \cdots \times (\sigma|_{i-1})^b(U_{j0(i-1)}) \times ((\sigma|_i)^p)^* \left((\sigma|_i)^b(U_{j0i})\right) \times (\sigma|_{i+1})^b(U_{j0(i+1)}) \times \cdots \times (\sigma|_n)^b(U_{j0n}).$$

As a consequence of Corollary 7.23, the sets $((\sigma|_i)^p)^*((\sigma|_i)^b(U_{j0i}))$ are recognizable. Therefore, the set $\sigma^*(U)$ is also recognizable.

□

Theorems 7.26 and 7.27 can be combined into a necessary and sufficient condition:

Corollary 7.28 *Let σ be a sequence involving the queues $q_1, q_2, \ldots, q_n$ $(n \ge 1)$ and let $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ be the alphabets of those queues. The set $\sigma^*(U)$ is recognizable for every recognizable set $U \subseteq \Sigma_1^* \times \Sigma_2^* \times \cdots \times \Sigma_n^*$ if and only if there do not exist $i, j \in \mathbf{N}$ such that $1 \le i < j \le n$ and such that $\sigma|_i$ and $\sigma|_j$ are counting sequences.*

This corollary makes it possible to decide whether the closure of a sequence σ of queue operations preserves the recognizability of sets of queue-set contents, i.e., to decide whether a cycle meta-transition can be associated to σ. An algorithm implementing the decision procedure is given in Figure 7.21.

Theorem 7.29 *Let σ be a sequence involving the queues $q_1, q_2, \ldots, q_n$ $(n \ge 1)$ and $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ be the alphabets of those queues. The set $\sigma^*(U)$ is recognizable for every recognizable set $U \subseteq \Sigma_1^* \times \Sigma_2^* \times \cdots \times \Sigma_n^*$ if and only if META?$(\sigma, \Sigma_1, \Sigma_2, \ldots, \Sigma_n) =$* **T**.

Proof The result is a direct consequence of Definition 7.24 and of Corollary 7.28. □

The proof of Theorem 7.27 provides a way of constructing a QDD representing $\sigma^*(U)$ given a sequence σ of queue operations such that META?$(\sigma, \Sigma_1, \Sigma_2, \ldots, \Sigma_n) = \mathbf{T}$ and a QDD representing the set U. An algorithm[3] implementing this construction is given in Figure 7.22.

Theorem 7.30 *Let σ be a sequence involving the queues $q_1, q_2, \ldots, q_n$ ($n \geq 1$), $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ be the alphabets of those queues, and $\mathcal{A}$ be a QDD representing the set of queue set contents $U \subseteq \Sigma_1^* \times \Sigma_2^* \times \cdots \times \Sigma_n^*$. If META?$(\sigma, \Sigma_1, \ldots, \Sigma_n) = \mathbf{T}$, then APPLY-STAR($\mathcal{A}, \sigma, \Sigma_1, \ldots, \Sigma_n$) is a QDD representing the set $\sigma^*(U)$.*

Proof The algorithm of Figure 7.22 is a direct implementation of the computation method described in the proof of Theorem 7.27. □

7.4 Creation of Multicycle Meta-Transitions

This section is aimed at providing the algorithms that are needed in order to be able to associate multicycle meta-transitions to systems using FIFO queues. As it has been shown in Section 3.4.2, the creation of multicycle meta-transitions is governed by a computable function MULTI-META-SET that takes as arguments a finite number of sequences of operations, and returns a finite number of memory functions defining multicycle meta-transitions that can be associated to the cycles labeled by those sequences of operations.

Regrettably, one cannot hope here to obtain results similar to those of Section 7.3, in which it was always possible to create a meta-transition whenever the memory function of this meta-transition preserved the recognizability of sets of queue-set contents. Indeed, there are classes of systems such as *lossy systems* [AJ93, AJ94] that can be modeled as QSMAs and for which it is known [CFI96] that their set of reachable queue-set contents is always recognizable but generally not computable. Since every QSMA $\mathcal{A}$ can be simulated by a QSMA $\mathcal{A}'$ with only one control location (for instance, by creating an additional queue whose content encodes the control location of $\mathcal{A}$), the existence of an algorithm for computing the effect of every multicycle meta-transition whose memory function preserves the recognizability of sets of queue-set contents would make it possible to compute the set of reachable queue-set contents of an arbitrary lossy system.

The solution we propose is based on an algorithm for computing the image of a representable set by a multicycle meta-transition, provided that the sequences of queue operations that characterize this meta-transition satisfy some conditions.

[3] In this algorithm, the test of equality of the languages accepted by two QDDs can be implemented by two calls to the function INCLUDED? presented in Section 6.2.4.

function APPLY-STAR(QDD $(\Sigma, S, \Delta, I, F)$, **sequence of** queue operations σ, alphabets $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$) : QDD

1: **var** $\mathcal{A}$: **array**$[0, 1, \ldots]$ **of** QDDs;
2: $\mathcal{A}_0, \mathcal{A}'$: QDDs;
3: i, b, p : integers;
4: f : function;
5: **begin**
6: $i := 1$;
7: **while** $i \leq n$ **do**
8: **if** $(|\Sigma_i| > 1 \wedge |(\sigma|_{i!})| > 0) \vee (|\Sigma_i| = 1 \wedge |(\sigma|_{i!})| > |(\sigma|_{i?})|)$ **then** **goto** break
9: **else** $i := i + 1$;
10: break:
11: $f := \Sigma_1^* \cdot \Sigma_2^* \cdots \Sigma_n^* \to \Sigma_1^* \cdot \Sigma_2^* \cdots \Sigma_n^* \ : \ w \mapsto w|_{\neq i}$;
12: $j := 0$;
13: $\mathcal{A}' := \mathcal{A}_0 := (\Sigma, S, \Delta, I, F)$;
14: $\mathcal{A}[0] :=$ APPLY-HOMOMORPHISM$(\mathcal{A}', f)$;
15: **repeat**
16: $j := j + 1$;
17: $\mathcal{A}' :=$ APPLY$(\mathcal{A}', \sigma, \Sigma_1, \Sigma_2, \ldots, \Sigma_n)$;
18: $\mathcal{A}[j] :=$ APPLY-HOMOMORPHISM$(\mathcal{A}', f)$
19: **until there exists** b **such that** $0 \leq b < j \wedge L(\mathcal{A}[j]) = L(\mathcal{A}[b])$;
20: **if** $i > n$ **then return** $\bigcup_{0 \leq k < j} \mathcal{A}[k]$;
21: $p := j - b$;
22: $\mathcal{A}' :=$ PERFORM-FUNCTION$(\mathcal{A}',$ APPLY-STAR-ONE$, i, (\sigma|i)^p)$;
23: **return** $\bigcup_{0 \leq k < b}$ APPLY$(\mathcal{A}_0, \sigma^k, \Sigma_1, \Sigma_2, \ldots, \Sigma_n) \cup \bigcup_{0 \leq k < p}$ APPLY$(\mathcal{A}', \sigma^k, \Sigma_1, \Sigma_2, \ldots, \Sigma_n)$
24: **end**.

Figure 7.22: Image of a QDD by the closure of a sequence of queue operations.

These conditions will be chosen so as to allow a simple generalization of the function APPLY introduced in Section 7.3 to the case of multicycle meta-transitions. We consider successively systems with one and then with an arbitrary number of queues.

7.4.1 Systems with One Queue

The function APPLY-STAR-ONE discussed in Section 7.3.1 proceeds by constructing a finite structure equivalent to the infinite union of all the automata that can be obtained by repeated applications of a given sequence of operations. Roughly speaking, the idea behind the construction was to capture some periodicity among the sets of states of the different automata.

The same approach is followed here. Let q be a queue of alphabet Σ, and $\sigma_1, \sigma_2, \ldots, \sigma_m$ ($m \geq 0$) be sequences of elementary queue operations involving q. The set $\bar{\sigma} = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ is called the *multisequence* characterized by $\sigma_1, \sigma_2, \ldots, \sigma_m$. The image by $\bar{\sigma}$ of a set $U \subseteq \Sigma^*$ of queue contents is defined as the set

$$\bar{\sigma}(U) = \sigma_1(U) \cup \sigma_2(U) \cup \cdots \cup \sigma_m(U).$$

The goal of this section is thus to provide an algorithm for computing a QDD representing

$$\bar{\sigma}^*(U) = \bigcup_{k \in \mathbf{N}} \bar{\sigma}^k(U)$$

given a QDD representing a recognizable set $U \subseteq \Sigma^*$, under some restrictions on $\bar{\sigma}$. If $\bar{\sigma} = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ is a multisequence, then $\bar{\sigma}_!$ denotes the multisequence $\{\sigma_{1!}, \sigma_{2!}, \ldots, \sigma_{m!}\}$. If q_i is a queue, then $\bar{\sigma}|_i$ denotes the multisequence obtained by removing from the sequences composing $\bar{\sigma}$ all the operations which do not involve q_i.

The first restriction concerns the receive operations that can be performed by the sequences composing $\bar{\sigma}$. In order to be able to exploit some results already established in Section 7.3.1, we restrict the class of multisequences that are considered to those whose components all share the same sequence of receive operations. This restriction is formalized by the following definition.

Definition 7.31 *The multisequence* $\{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ *is* receive-deterministic *if it is such that* $\sigma_{1?} = \sigma_{2?} = \cdots = \sigma_{m?}$.

If $\bar{\sigma} = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ is receive-deterministic, then $\bar{\sigma}_?$ denotes the sequence $\sigma_{i?}$, where i is arbitrarily chosen in $\{1, 2, \ldots, m\}$. A positive property of receive-deterministic multisequences is that they allow a simple generalization of the function APPLY-ONE introduced in Section 7.2.1. Indeed, computing the image of a QDD $\mathcal{A}$ by a such a multisequence $\bar{\sigma}$ can be done by computing separately the image of $\mathcal{A}$ by each sequence composing $\bar{\sigma}$, and then joining together the tails of the resulting QDDs. An algorithm formalizing this construction is given in Figure 7.23.

function APPLY-MULTI-ONE(QDD $\mathcal{A}$, **multisequence of** queue operations $\{\sigma_1, \sigma_2, \ldots, \sigma_m\}$) : QDD

1: **var** $(\Sigma, S, \Delta, I, F), (\Sigma', S', \Delta', I', F')$: QDDs;
2: i : integer;
3: s : state;
4: **begin**
5: $(\Sigma, S, \Delta, I, F) := \mathcal{A}$;
6: **let** $s \notin S$;
7: $S := S \cup \{s\}$;
8: $F := \{s\}$;
9: **for** $i := 1$ **to** m **do**
10: **begin**
11: $(\Sigma', S', \Delta', I', F') :=$ APPLY-ONE$(\mathcal{A}, \sigma_i)$;
12: $S := S \cup S'$;
13: $\Delta := \Delta \cup \Delta'$;
14: $\Delta := \Delta \cup \{(s', w, s) \in S' \times \Sigma'^* \times \{s\} \mid (\exists (s_1, w_1, s_2) \in \Delta')$ $(s_1 = s' \wedge w_1 = w \wedge s_2 \in F')\}$;
15: **if** $|\sigma_{i!}| > 0$ **then** $\Delta := \Delta \setminus (S' \times \Sigma'^* \times F')$;
16: $I := I'$;
17: **if** $I' \cap F' \neq \emptyset$ **then** $I := I \cup \{s\}$
18: **end**;
19: **return** $(\Sigma, S, \Delta, I, F)$
20: **end**.

Figure 7.23: Image of a single-queue QDD by a receive-deterministic multisequence of queue operations.

Theorem 7.32 *Let q be a queue of alphabet Σ, $\bar{\sigma} = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ be a receive-deterministic multisequence of elementary operations on q, and $\mathcal{A}$ be a QDD representing the set $U \subseteq \Sigma^*$. APPLY-MULTI-ONE($\mathcal{A}, \bar{\sigma}$) is a QDD representing the set $\bar{\sigma}(U)$.*

Proof Each execution of Line 11 computes a QDD $\mathcal{A}_i$ representing $\sigma_i(U)$, with $i \in \{1, 2, \ldots, m\}$. The sets of states and of transitions of the QDD $\mathcal{A}'$ returned by the algorithm are respectively the union of the sets of states and of the sets of transitions of all the $\mathcal{A}_i$. Since $\bar{\sigma}$ is receive-deterministic, all the $\mathcal{A}_i$ are identical except for their tail. It follows that $\mathcal{A}'$ accepts a word w if and only if w is accepted by one of the $\mathcal{A}_i$ (the QDD $\mathcal{A}'$ can actually be seen as an automaton with m disjoint tails). There is one small optimization at Lines 14–15, in which the accepting states of the $\mathcal{A}_i$ are grouped into a single state. This optimization, which is introduced in order to facilitate future applications of the algorithm, does not influence the language accepted by the returned automaton. □

Let q be a queue of alphabet Σ, $U \subseteq \Sigma^*$ be a recognizable set of queue contents, $\mathcal{A}$ be a QDD representing U, and $\bar{\sigma}$ be a receive-deterministic multisequence of operations on q. We assume that $\mathcal{A}$ is in normal form. We have $\bar{\sigma}^*(U) = L(\mathcal{A}_0) \cup L(\mathcal{A}_1) \cup \cdots$, where $\mathcal{A}_0, \mathcal{A}_1, \ldots$ are QDDs such that:

- $\mathcal{A}_0 = \mathcal{A}$;
- $\mathcal{A}_{i+1} = \text{APPLY-MULTI-ONE}(\mathcal{A}_i, \bar{\sigma})$ for every $i \geq 0$.

One can apply the same reasoning as in Section 7.3.1 in order to capture the redundancy among the sets of states and of transitions of the $\mathcal{A}_i$ into a finite structure. For each $i \in \mathbf{N}$, we denote $(\Sigma_i, S_i, \Delta_i, I_i, F_i)$ the components of $\mathcal{A}_i$. The set of all the states that have been created during the first i applications of APPLY-MULTI-ONE is called the *multitail* of $\mathcal{A}_i$ and is denoted *mtail*(i). The multitail of $\mathcal{A}_i$ is actually composed of m parallel tails created by the calls to APPLY-ONE (these tails are called the *component tails* of $\mathcal{A}_i$). The situation is depicted in Figure 7.24.

In order to be able to apply straightforwardly the technique developed in Section 7.3.1 to multitails, we need to impose an additional restriction. Roughly speaking, this restriction consists of requiring that during each successive call to APPLY-MULTI-ONE, the initial states belonging to different component tails of an automaton are shifted together, i.e., that they are moved by the same amount of transitions. Formally, we have the following definition.

Definition 7.33 *Let q be a queue of alphabet Σ and $\bar{\sigma} = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ be a multisequence of elementary operations on q. The multisequence $\bar{\sigma}$ is* send-synchronized *if it is receive-deterministic, and if for every $i, j \in \{1, 2, \ldots, m\}$, $|\sigma_{i!}| = |\sigma_{j!}|$.*

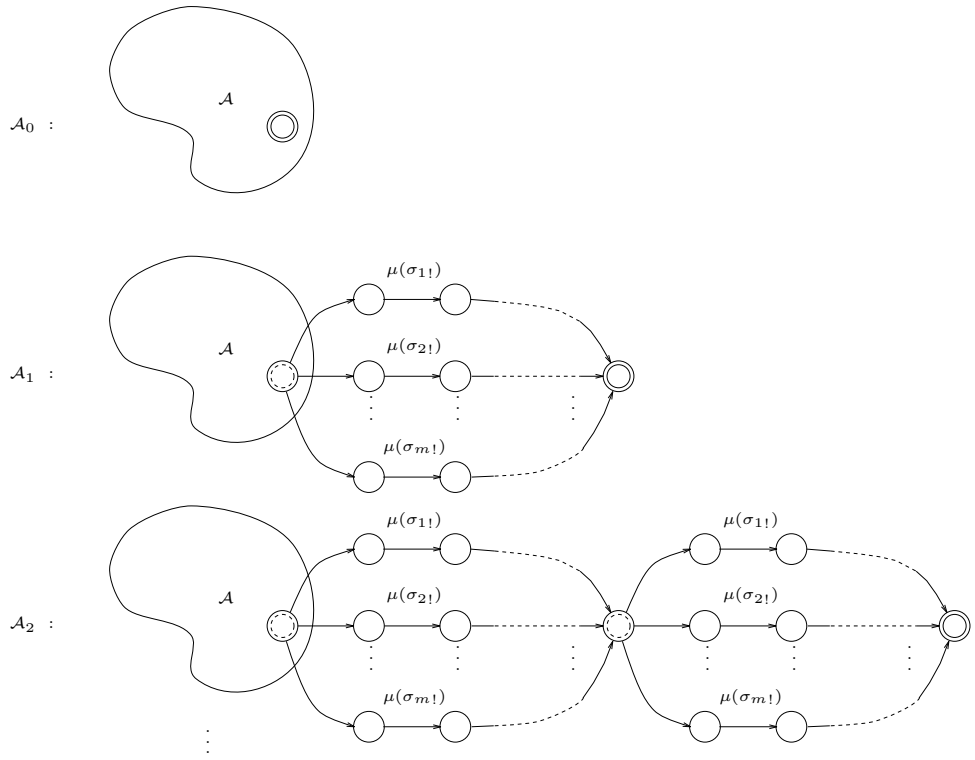

Figure 7.24: Effect of repeated applications of APPLY-MULTI-ONE.

If $\bar{\sigma} = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ is send-synchronized, then $|\bar{\sigma}_!|$ denotes the length $|\sigma_{i!}|$, where i is arbitrarily chosen in $\{1, 2, \ldots, m\}$. Computing the image of a recognizable set of queue contents by the closure of $\bar{\sigma}$ can be done by performing essentially the same operations as in Function APPLY-STAR-ONE to each component tail of the $\mathcal{A}_i$. There are however two minor differences:

- The subroutine APPEND-LOOP must be replaced by a subroutine APPEND-MULTI-LOOP, whose purpose is to apply the closure of a multisequence only composed of send operations;

- Applying k times ($k \geq 0$) the multisequence $\{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ is in general not equivalent to applying once the multisequence $\{\sigma_1^k, \sigma_2^k, \ldots, \sigma_m^k\}$. Two subroutines APPLY-N-MULTI-ONE and APPEND-N-MULTI-LOOP must be introduced in order to generalize APPLY-MULTI-ONE and APPEND-MULTI-LOOP to repetitions of multisequences of operations.

The generalization of the function APPLY-STAR-ONE to send-synchronized multisequences of queue operations is given in Figures 7.28, 7.29 and 7.30. For convenience, the function implemented by this algorithm has an additional integer parameter k that allows to apply the closure of $\bar{\sigma}^k$ (rather than the closure of $\bar{\sigma}$). The three subroutines upon which this algorithm relies are given in Figures 7.25,

```
function APPLY-N-MULTI-ONE(QDD A, multisequence of
                                queue operations σ̄, integer k) : QDD
1:    var i : integer;
2:    begin
3:        for i := 1 to k do
4:            A := APPLY-MULTI-ONE(A, σ̄);
5:        return A
6:    end;
```

Figure 7.25: Image of a single-queue QDD by repetitions of a receive-deterministic multisequence of queue operations.

7.26 and 7.27. The correctness of the subroutines and of the algorithm is established by the following theorems.

Theorem 7.34 *Let q be a queue of alphabet Σ, $\bar{\sigma}$ be a receive-deterministic multisequence of elementary operations on q, $k \in \mathbf{N}$ be an integer, and $\mathcal{A}$ be a QDD representing the set $U \subseteq \Sigma^*$. APPLY-N-MULTI-ONE($\mathcal{A}, \bar{\sigma}, k$) is a QDD representing the set $\bar{\sigma}^k(U)$.*

Proof Immediate. □

Theorem 7.35 *Let q be a queue of alphabet Σ, $w_1, w_2, \ldots, w_m \in \Sigma^*$ ($m \geq 0$) be same-length words, $k \in \mathbf{N}$ be an integer, and $\mathcal{A}$ be a QDD representing the set $U \subseteq \Sigma^*$. APPEND-MULTI-LOOP($\mathcal{A}, w_1, w_2, \ldots, w_m$) is a QDD representing the set $(\bar{\sigma})^*(U)$, where $\bar{\sigma}$ is the multisequence $\{q!w_1, q!w_2, \ldots, q!w_m\}$. APPEND-N-MULTI-LOOP($\mathcal{A}, w_1, w_2, \ldots, w_m, k$) is a QDD representing the set $(\bar{\sigma}^k)^*(U)$.*

Proof Immediate. □

Theorem 7.36 *Let q be a queue of alphabet Σ, $\bar{\sigma}'$ be a send-synchronized multisequence of elementary operations on q, $k \in \mathbf{N}$ be an integer, and $\mathcal{A}$ be a QDD representing the set $U \subseteq \Sigma^*$. APPLY-N-MULTI-STAR-ONE($\mathcal{A}, \bar{\sigma}, k$) is a QDD representing the set $\bar{\sigma}^*(U)$, where $\bar{\sigma} = (\bar{\sigma}')^k$.*

Proof The proof follows the same lines as the construction presented in Section 7.3.1. For every $i \in \mathbf{N}$, the *rank* of a state $s \in mtail(\mathcal{A}_i)$ is defined as the length of the shortest path leading from an accepting state of $\mathcal{A}_0$ to s. The base b, the period p and the automata $\mathcal{A}'_i$ are computed exactly as in Section 7.3.1, the multitail of each

function APPEND-N-MULTI-LOOP(QDD $(\Sigma, S, \Delta, I, F)$,
words $a_{1,1}a_{1,2}\cdots a_{1,l}, a_{2,1}a_{2,2}\cdots a_{2,l}, \ldots, a_{m,1}a_{m,2}\cdots a_{m,l}$, integer k) : QDD

1: **var** s : **array**$[1\ldots k, 1\ldots m, 1\ldots l]$ **of** states;
2: **begin**
3: **if** $m = 0 \vee k = 0$ **then return** $(\Sigma, S, \Delta, I, F)$;
4: **let** $\{s[i_1, i_2, i_3] \mid 1 \le i_1 \le k \wedge 1 \le i_2 \le m \wedge 1 \le i_3 \le l\} \cap S = \emptyset$;
5: $S := S \cup \{s[i_1, i_2, i_3] \mid 1 \le i_1 \le k \wedge 1 \le i_2 \le m \wedge 1 \le i_3 \le l\}$;
6: $\Delta := \Delta \cup \{(s', a_{i_2,1}, s[1, i_2, 1]) \mid s' \in F \wedge 1 \le i_2 \le m\}$
$\cup\{(s[i_1, i_2, i_3], a_{i_2,i_3+1}, s[i_1, i_2, i_3 + 1]) \mid 1 \le i_1 \le k$
$\wedge 1 \le i_2 \le m \wedge 1 \le i_3 \le l - 2\}$
$\cup\{(s[i_1, i_2, l-1], a_{i_2,l}, s[i_1, 1, l]) \mid 1 \le i_1 \le k \wedge 1 \le i_2 \le m\}$
$\cup\{(s[i_1, 1, l], a_{i_2,1}, s[i_1 + 1, i_2, 1]) \mid 1 \le i_1 \le k - 1 \wedge 1 \le i_2 \le m\}$
$\cup\{(s[k, 1, l], a_{i_2,1}, s[1, i_2, 1]) \mid 1 \le i_2 \le m\}$;
7: $F := F \cup \{s[k, 1, l]\}$;
8: **return** $(\Sigma, S, \Delta, I, F)$
9: **end**.

Figure 7.26: Subroutine APPEND-N-MULTI-LOOP.

function APPEND-MULTI-LOOP(QDD $\mathcal{A}$, words $w_1, w_2, \ldots, w_m$) : QDD

1: **begin**
2: **return** APPEND-N-MULTI-LOOP$(\mathcal{A}, w_1, w_2, \ldots, w_m, 1)$
3: **end**.

Figure 7.27: Subroutine APPEND-MULTI-LOOP.

function APPLY-N-MULTI-STAR-ONE(QDD $(\Sigma, S, \Delta, I, F)$, **multisequence of** queue operations $\bar{\sigma} = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$, integer k) : QDD

1: **var** $\mathcal{A}$: **array**$[0, 1, \ldots]$ **of** QDDs;
2: I^α : **array**$[0, 1, \ldots]$ **of sets of** states;
3: S_0, S_1, S_2 : **sets of** states;
4: $\mathcal{A}'_{i_0}, \mathcal{A}'_{i_0+1}, \mathcal{A}^{\alpha\beta'}_{i_0}, \mathcal{A}^{\alpha\beta'}, \mathcal{A}^{\gamma'}_{i_0}, \mathcal{A}^{\gamma'}, \mathcal{A}^{\gamma''}, \mathcal{A}^{\delta'}_{i_0+1}, \mathcal{A}^{\delta\epsilon'}, \mathcal{A}^{\delta'_R}, \mathcal{A}^{\delta''_R}, \mathcal{A}', \mathcal{A}''$: QDDs;
5: $(\Sigma'_{i_0}, S'_{i_0}, \Delta'_{i_0}, I'_{i_0}, F'_{i_0}), (\Sigma'_{i_0+1}, S'_{i_0+1}, \Delta'_{i_0+1}, I'_{i_0+1}, F'_{i_0+1})$: QDDs;
6: $n_1, n_2, p, b, d, i, i_0, i_1, i_2, l$: integers;
7: **begin**
8: $\mathcal{A}[0] := (\Sigma, S, \Delta, I, F) :=$ NORMALIZE$((\Sigma, S, \Delta, I, F))$;
9: **if** $|\bar{\sigma}_?| = 0 \vee k = 0$ **then return** APPEND-N-MULTI-LOOP$(\mathcal{A}[0], \mu(\sigma_{1!}), \mu(\sigma_{2!}), \ldots, \mu(\sigma_{m!}), k)$;
10: $I^\alpha[0] := I$;
11: $S_0 := S$;
12: $n_1 := 0$;
13: **repeat**
14: $n_1 := n_1 + 1$;
15: $\mathcal{A}[n_1] := (\Sigma, S, \Delta, I, F) :=$ APPLY-N-MULTI-ONE$(\mathcal{A}[n_1 - 1], \bar{\sigma}, k)$;
16: $I^\alpha[n_1] := I \cap S_0$
17: **until there exists** n_2 **such that** $0 \leq n_2 < n_1 \wedge I^\alpha[n_1] = I^\alpha[n_2]$;
18: **if** $|\bar{\sigma}_!| = 0$ **then return** $\bigcup_{0 \leq i \leq n_1} \mathcal{A}[i]$;
19: $p := \mathrm{lcm}(n_1 - n_2, k|\bar{\sigma}_!|)$;
20: $b := k|\bar{\sigma}_!|\lceil n_2/(k|\bar{\sigma}_!|)\rceil$;
21: $i_0 := \left\lceil 3\frac{|\bar{\sigma}_?|}{|\bar{\sigma}_!|} - \frac{b}{p} \right\rceil + 1$;
22: $\mathcal{A}'_{i_0} := (\Sigma'_{i_0}, S'_{i_0}, \Delta'_{i_0}, I'_{i_0}, F'_{i_0}) :=$ APPLY-N-MULTI-ONE$(\mathcal{A}[n_1], \bar{\sigma}, k(b - n_1 + i_0 p))$;
23: $\mathcal{A}'_{i_0+1} := (\Sigma'_{i_0+1}, S'_{i_0+1}, \Delta'_{i_0+1}, I'_{i_0+1}, F'_{i_0+1}) :=$ APPLY-N-ONE$(\mathcal{A}'_{i_0}, \bar{\sigma}, kp)$;
24: $S_1 := \{s' \in S'_{i_0} \mid (\exists s \in S_0, w \in \Sigma^*)((s, w, s') \in \Delta'^{*}_{i_0} \wedge |w| \leq kp|\bar{\sigma}_?|)\}$;
25: $S_2 := \{s' \in S'_{i_0+1} \mid (\exists s \in S_0, w \in \Sigma^*)((s, w, s') \in \Delta'^{*}_{i_0+1} \wedge |w| \leq 2kp|\bar{\sigma}_?|)\}$;

(...)

Figure 7.28: Image of a single-queue QDD by the closure of a send-synchronized multisequence of queue operations.

(. . .)

$\mathcal{A}_{i_0}^{\alpha\beta'} := (\Sigma'_{i_0}, S'_{i_0}, \Delta'_{i_0}, I'_{i_0} \cap S_1, F'_{i_0})$;

$\mathcal{A}^{\alpha\beta'} :=$ APPEND-N-MULTI-LOOP$(\mathcal{A}_{i_0}^{\alpha\beta'}, \mu(\sigma_{1!}), \ldots, \mu(\sigma_{m!}), kp)$;

$\mathcal{A}_{i_0}^{\gamma'} := (\Sigma'_{i_0}, S'_{i_0}, \Delta'_{i_0}, I'_{i_0} \setminus S_1, F'_{i_0})$;

$\mathcal{A}_{i_0+1}^{\delta'} := (\Sigma'_{i_0+1}, S'_{i_0+1}, \Delta'_{i_0+1}, I'_{i_0+1} \cap (S_2 \setminus S_1), F'_{i_0+1})$;

if $|\bar{\sigma}_!| \geq |\bar{\sigma}_?|$ **then**

begin

$d := \frac{p}{|\bar{\sigma}_!|}(|\bar{\sigma}_!| - |\bar{\sigma}_?|)$;

$\mathcal{A}^{\gamma'} :=$ APPEND-N-MULTI-LOOP$(\mathcal{A}_{i_0}^{\gamma'}, \mu(\sigma_{1!}), \ldots, \mu(\sigma_{m!}), kd)$;

$\mathcal{A}^{\delta\epsilon'} :=$ APPEND-N-MULTI-LOOP(
APPEND-N-MULTI-LOOP$(\mathcal{A}_{i_0+1}^{\delta'}, \mu(\sigma_{1!}), \ldots,$
$\mu(\sigma_{m!}), kd), \mu(\sigma_{1!}), \ldots, \mu(\sigma_{m!}), kp)$

end

else

begin

$i_1 := \left\lceil \frac{(b/p)|\bar{\sigma}_!| + (i_0 - 1)|\bar{\sigma}_?|}{|\bar{\sigma}_?| - |\bar{\sigma}_!|} \right\rceil$;

$\mathcal{A}^{\gamma'} := \mathcal{A}^{\gamma''} := \mathcal{A}_{i_0}^{\gamma'}$;

for $i := i_0 + 1$ **to** $i_1 - 1$ **do**

begin

$\mathcal{A}^{\gamma''} :=$ APPLY-N-MULTI-ONE$(\mathcal{A}^{\gamma''}, \bar{\sigma}, kp)$;

$\mathcal{A}^{\gamma'} :=$ UNION$(\mathcal{A}^{\gamma'}, \mathcal{A}^{\gamma''})$

end;

$i_2 := 1 + \max(i_0, \left\lceil \frac{(b/p)|\bar{\sigma}_!| + (i_0 - 2)|\bar{\sigma}_?|}{|\bar{\sigma}_?| - |\bar{\sigma}_!|} \right\rceil)$;

$\mathcal{A}^{\delta'_R} := \mathcal{A}^{\delta''_R} := \mathcal{A}_{i_0+1}^{\delta'}$;

$l := (b + (i_0 + 2)p)k|\bar{\sigma}_!| - 4pk|\bar{\sigma}_?|$;

(. . .)

Figure 7.29: Image of a single-queue QDD by the closure of a send-synchronized multisequence of queue operations (continued).

(...)

48: **for** $i := i_0 + 2$ **to** $i_2 + p - 1$ **do**
49: **if** $l \geq 0$ **then**
50: **begin**
51: $\mathcal{A}^{\delta''_R} :=$ APPLY-N-MULTI-ONE$(\mathcal{A}^{\delta''_R}, \bar{\sigma}, kp)$;
52: $\mathcal{A}^{\delta'_R} :=$ UNION$(\mathcal{A}^{\delta'_R}, \mathcal{A}^{\delta''_R})$;
53: $l := l + kp(|\bar{\sigma}_!| - |\bar{\sigma}_?|)$
54: **end**
55: **else**
56: **begin**
57: $\mathcal{A}^{\delta''_R} :=$ APPLY-N-MULTI-ONE$(\mathcal{A}^{\delta''_R}, \{\sigma_{1!}, \sigma_{2!}, \ldots, \sigma_{m!}\}, kp)$;
58: $\mathcal{A}^{\delta'_R} :=$ UNION$(\mathcal{A}^{\delta'_R}, \mathcal{A}^{\delta''_R})$;
59: $l := l + kp|\bar{\sigma}_!|$
60: **end**;
61: $\mathcal{A}^{\delta\epsilon'} :=$ APPEND-N-MULTI-LOOP$(\mathcal{A}^{\delta'_R}, \mu(\sigma_{1!}), \ldots, \mu(\sigma_{m!}), kp)$;
62: $\mathcal{A}^{\delta\epsilon'} :=$ UNION$(\mathcal{A}^{\delta\epsilon'}$, APPLY-N-MULTI-ONE$(\mathcal{A}^{\delta'_R}, \bar{\sigma}, kp))$;
63: $\mathcal{A}^{\delta\epsilon'} :=$ UNION$(\mathcal{A}^{\delta\epsilon'}$, APPLY-N-MULTI-ONE$(\mathcal{A}^{\delta'_R}, \bar{\sigma}, 2kp))$
64: **end**;
65: $\mathcal{A}' :=$ UNION$(\bigcup_{0 \leq i < i_0}$ APPLY-N-MULTI-ONE$(\mathcal{A}[n_1], \bar{\sigma}, k(b - n_1 + pi)), \bigcup_{\xi \in \{\alpha\beta, \gamma, \delta\epsilon\}} \mathcal{A}^{\xi'})$;
66: $\mathcal{A}'' :=$ UNION$(\bigcup_{0 \leq i < b}$ APPLY-N-MULTI-ONE$(\mathcal{A}[0], \bar{\sigma}, ki), \bigcup_{0 \leq i < p}$ APPLY-N-MULTI-ONE$(\mathcal{A}', \bar{\sigma}, ki))$;
67: **return** $\mathcal{A}''$
68: **end**.

Figure 7.30: Image of a single-queue QDD by the closure of a a send-synchronized multisequence of queue operations (continued).

$\mathcal{A}'_i$ being denoted $mtail'(i)$. The notion of robustness and the partitioning of the initial states stay unchanged (except for the transformation of $tail'(i)$ into $mtail'(i)$), as are the definitions of right blocks and of the indices i_0, i_1 and i_2. Adapting the calculations made in Section 7.3.1, we now obtain

$$\begin{aligned}
\bigcup_{i\geq i_0}\Big(L(\mathcal{A}_i^{\alpha'}) \cup L(\mathcal{A}_i^{\beta'})\Big) &= ((\bar\sigma_!)^p)^*\Big(L(\mathcal{A}_{i_0}^{\alpha'}) \cup L(\mathcal{A}_{i_0}^{\beta'})\Big);\\
\bigcup_{i\geq i_0} L(\mathcal{A}_i^{\gamma'}) &= ((\bar\sigma_!)^d)^*\Big(L(\mathcal{A}_{i_0}^{\gamma'})\Big) \quad \text{if } |\bar\sigma_!| \geq |\bar\sigma_?|;\\
\bigcup_{i\geq i_0} L(\mathcal{A}_i^{\gamma'}) &= \bigcup_{i_0\leq i<i_1} L(\mathcal{A}_i^{\gamma'}) \qquad \text{if } |\bar\sigma_!| < |\bar\sigma_?|,\\
\bigcup_{i\geq i_0} L(\mathcal{A}_i^{\delta'}) &= ((\bar\sigma_!)^p)^*\Big(((\bar\sigma_!)^d)^*\Big(L(\mathcal{A}_{i_0+1}^{\delta'})\Big)\Big) \quad \text{if } |\bar\sigma_!| \geq |\bar\sigma_?|;\\
\bigcup_{i\geq i_0}\Big(L(\mathcal{A}_i^{\delta'}) \cup L(\mathcal{A}_i^{\epsilon'})\Big) &= \bigcup_{i>i_0}((\bar\sigma_!)^p)^*\Big(L(\mathcal{A}_i^{\delta'_R})\Big) \cup \bar\sigma^p\left(\bigcup_{i>i_0} L(\mathcal{A}_i^{\delta'_R})\right)\\
&\qquad \cup \bar\sigma^{2p}\left(\bigcup_{i>i_0} L(\mathcal{A}_i^{\delta'_R})\right) \qquad \text{if } |\bar\sigma_!| < |\bar\sigma_?|,
\end{aligned}$$

with $d = (|\bar\sigma_!| - |\bar\sigma_?|)(p/|\bar\sigma_!|)$. The computation of the union of the $L(\mathcal{A}_i^{\delta'_R})$ is unchanged. □

7.4.2 Systems with Any Number of Queues

Let us now generalize the results of Section 7.3.2 to multisequences of queue operations. The first step is to generalize the algorithm for computing the image of a recognizable set of queue-set contents. Like in Section 7.2.3, we remark that two operations involving different queues are independent, i.e., that the result of applying such operations to a queue-set content does not depend on the order in which they are applied. It follows that for every multisequence $\bar\sigma$ of operations involving the queues $q_1, q_2, \ldots, q_n$ and set of queue-set contents U, we have $\bar\sigma(U) = (\bar\sigma|_1; \bar\sigma|_2; \ldots; \bar\sigma|_n)(U)$. Applying $\bar\sigma$ to U can thus be done by applying successively the projections of $\bar\sigma$ onto the different queues of the system. An algorithm formalizing this method is given in Figure 7.31. A generalization of this algorithm to repeated applications of a multisequence of queue operations is given in Figure 7.32.

Theorem 7.37 *Let $q_1, q_2, \ldots, q_n$ be queues of respective alphabets $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$, $\bar\sigma$ be a multisequence of elementary operations on these queues such that for every $i \in \{1, \ldots, n\}$, the multisequence $\bar\sigma|_i$ is send-synchronized, and let $\mathcal{A}$ be a QDD representing the set $U \subseteq \Sigma_1^* \times \Sigma_2^* \times \cdots \times \Sigma_n^*$. APPLY-MULTI($\mathcal{A}, \bar\sigma, \Sigma_1, \Sigma_2, \ldots, \Sigma_n$) is a QDD representing the set $\bar\sigma(U)$.*

Proof Immediate, as a consequence of Theorems 7.11 and 7.32. □

```
function APPLY-MULTI(QDD A, multisequence of queue operations σ̄,
                                        alphabets Σ_1, Σ_2, ..., Σ_n) : QDD
1:  var i : integer;
2:  begin
3:      for i := 1 to n do
4:          A := PERFORM-FUNCTION(A, APPLY-MULTI-ONE, i, σ̄|_i);
5:      return A
6:  end.
```

Figure 7.31: Image of an arbitrary QDD by multisequence of queue operations.

```
function APPLY-N-MULTI(QDD A, multisequence of queue operations σ̄,
                                        alphabets Σ_1, Σ_2, ..., Σ_n, integer k) : QDD
1:  var i : integer;
2:  begin
3:      for i := 1 to k do
4:          A := APPLY-MULTI(A, σ̄, Σ_1, Σ_2, ..., Σ_n);
5:      return A
6:  end.
```

Figure 7.32: Image of an arbitrary QDD by repeated applications of a multisequence of queue operations.

Theorem 7.38 *Let $q_1, q_2, \ldots, q_n$ be queues of respective alphabets $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$, $\bar{\sigma}$ be a multisequence of elementary operations on these queues such that for every $i \in \{1, \ldots, n\}$, the multisequence $\bar{\sigma}|_i$ is send-synchronized, let $k \geq 0$ be an integer, and let $\mathcal{A}$ be a QDD representing the set $U \subseteq \Sigma_1^* \times \Sigma_2^* \times \cdots \times \Sigma_n^*$. APPLY-N-MULTI($\mathcal{A}, \bar{\sigma}, \Sigma_1, \Sigma_2, \ldots, \Sigma_n, k$) is a QDD representing the set $\bar{\sigma}^k(U)$.*

Proof Immediate. □

The next step is to show that the concept of counting multisequences of queue operations can be introduced as a direct generalization of counting sequences of operations.

Definition 7.39 *Let q be a queue of alphabet Σ, and $\bar{\sigma} = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ ($m \geq 0$) be a send-synchronized multisequence of operations on q. The multisequence $\bar{\sigma}$ is* counting *if one of the following conditions is satisfied:*

- $|\Sigma| > 1$ *and* $|\bar{\sigma}_!| > 0$,
- $|\Sigma| = 1$ *and* $|\bar{\sigma}_!| > |\bar{\sigma}_?|$.

Theorem 7.40 *Let q be a queue of alphabet Σ, and $\bar{\sigma} = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ ($m \geq 0$) be a send-synchronized multisequence of operations on q. The sequence $\bar{\sigma}$ is counting if and only if there exists a recognizable set $U \subseteq \Sigma^*$ of queue contents such that $\bar{\sigma}^{k_1}(U) \neq \bar{\sigma}^{k_2}(U)$ for all $k_1, k_2 \in \mathbf{N}$ such that $k_1 \neq k_2$.*

Proof If either $|\Sigma| > 1$ and $\bar{\sigma}$ is not counting, or $|\Sigma| = 1$, then there exists a sequence σ of operations involving q that is equivalent to $\bar{\sigma}$ and that is counting if and only if σ is counting. The result is then a consequence of Theorem 7.25.

It thus remains to show that if $|\Sigma| > 1$ and $\bar{\sigma}$ is counting, then there exists a recognizable set $U \subseteq \Sigma^*$ of queue contents such that $\bar{\sigma}^{k_1}(U) \neq \bar{\sigma}^{k_2}(U)$ for all $k_1, k_2 \in \mathbf{N}$ such that $k_1 \neq k_2$. Let $a \in \Sigma$ be a symbol different from $\bar{\sigma}_?[1]$. Choosing $U = \mu(\bar{\sigma}_?)^* \cdot a$ yields for every $k \in \mathbf{N}$

$$\sigma^k(U) = \mu(\bar{\sigma}_?)^* \cdot a \cdot (\mu(\sigma_{1!}) \cup \mu(\sigma_{2!}) \cup \cdots \cup \mu(\sigma_{m!}))^k.$$

□

We are now able to characterize precisely the send-synchronized multisequences whose closure preserves the recognizability of sets of queue-set contents.

Theorem 7.41 *Let $\bar{\sigma}$ be a multisequence of operations involving the queues $q_1, q_2, \ldots, q_n$ ($n \geq 1$) such that $\bar{\sigma}|_1, \bar{\sigma}|_2, \ldots, \bar{\sigma}|_n$ are all send-synchronized, and let $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ be the alphabets of $q_1, q_2, \ldots, q_n$. The set $\sigma^*(U)$ is recognizable for every recognizable set $U \subseteq \Sigma_1^* \times \Sigma_2^* \times \cdots \times \Sigma_n^*$ if and only if there do not exist $i, j \in \mathbf{N}$ such that $1 \leq i < j \leq n$ and such that $\bar{\sigma}|_i$ and $\bar{\sigma}|_j$ are counting multisequences.*

Proof The proof is along the same lines as the ones of Theorems 7.26 and 7.27. If there exist $i, j \in \mathbf{N}$ such that $1 \leq i < j \leq n$ and such that $\bar{\sigma}|_i$ and $\bar{\sigma}|_j$ are counting, one builds a recognizable set U of queue-set contents exactly like in the proof of Theorem 7.26, and then observes that $\bar{\sigma}^*(U)$ is not recognizable. If there do not exist $i, j \in \mathbf{N}$ such that $1 \leq i < j \leq n$ and such that $\bar{\sigma}|_i$ and $\bar{\sigma}|_j$ are counting, then the operations performed in the proof of Theorem 7.27 directly yield a formula for computing a finite-state representation of $\bar{\sigma}^*(U)$. The only required modification is to replace σ by $\bar{\sigma}$ in the proof. □

This proof provides a way of constructing a QDD representing $\bar{\sigma}^*(U)$ given a multisequence $\bar{\sigma}$ of queue operations that satisfies the requirements of Theorem 7.41. An algorithm implementing this construction is given in Figure 7.33.

Theorem 7.42 *Let $\bar{\sigma} = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ $(m \geq 0)$ be a multisequence of operations involving the queues $q_1, q_2, \ldots, q_n$ $(n \geq 1)$, $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ be the alphabets of those queues, and $\mathcal{A}$ be a QDD representing the set of queue set contents $U \subseteq \Sigma_1^* \times \Sigma_2^* \times \cdots \times \Sigma_n^*$. If $\bar{\sigma}$ is such that for every i such that $1 \leq i \leq n$, $\bar{\sigma}|_i$ is send-synchronized, and if there exists at most one i such that $1 \leq i \leq n$ and $\bar{\sigma}|_i$ is counting, then APPLY-MULTI-STAR($\mathcal{A}, \bar{\sigma}, \Sigma_1, \ldots, \Sigma_n$) is a QDD representing the set $\bar{\sigma}^*(U)$.*

Proof The algorithm of Figure 7.33 is a direct implementation of the computation method described in the proofs of Theorems 7.27 and 7.41. □

It remains to give an implementation of the function MULTI-META-SET that is used for creating multicycle meta-transitions. Recall that this function takes as arguments a finite set of sequences of queue operations, and returns a finite number of memory functions corresponding to multicycle meta-transitions that can be created. The function MULTI-META-SET can be evaluated by first discarding all the input sequences that have more than one counting projection. Then, one partitions the remaining sequences according to their subsequences of receive operations and to the lengths of their subsequences of send operations. Indeed, the operations studied in this section require multisequences to be receive-deterministic and send-synchronized. The last step is to create a multisequence for each set of sequences belonging to the partition. The set of memory functions returned by MULTI-META-SET then contains the closures of those multisequences. An algorithm implementing this construction is given in Figure 7.34.

7.5 Creation of Other Meta-Transitions

In this section, we show that the results presented in Sections 7.2, 7.3 and 7.4 can easily be adapted to *lossy systems*, which are systems whose FIFO queues are unreliable and can non-deterministically lose messages [AJ93, AJ94, CFI96]. This

function APPLY-MULTI-STAR(QDD $(\Sigma, S, \Delta, I, F)$, **multisequence of** queue operations $\bar{\sigma}$, alphabets $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$) : QDD

1: **var** $\mathcal{A}$: **array**$[0, 1, \ldots]$ **of** QDDs;
2: $\mathcal{A}_0, \mathcal{A}'$: QDDs;
3: i, b, p : integers;
4: f : function;
5: **begin**
6: $i := 1$;
7: **while** $i \leq n$ **do**
8: **if** $(|\Sigma_i| > 1 \wedge |(\bar{\sigma}|_{i!})| > 0) \vee (|\Sigma_i| = 1 \wedge |(\bar{\sigma}|_{i!})| > |(\bar{\sigma}|_{i?})|)$ **then** **goto** break
9: **else** $i := i + 1$;
10: break:
11: $f := \Sigma_1^* \cdot \Sigma_2^* \cdots \Sigma_n^* \to \Sigma_1^* \cdot \Sigma_2^* \cdots \Sigma_n^* \ : \ w \mapsto w|_{\neq i}$;
12: $j := 0$;
13: $\mathcal{A}' := \mathcal{A}_0 := (\Sigma, S, \Delta, I, F)$;
14: $\mathcal{A}[0] :=$ APPLY-HOMOMORPHISM$(\mathcal{A}', f)$;
15: **repeat**
16: $j := j + 1$;
17: $\mathcal{A}' :=$ APPLY-MULTI$(\mathcal{A}', \bar{\sigma}, \Sigma_1, \Sigma_2, \ldots, \Sigma_n)$;
18: $\mathcal{A}[j] :=$ APPLY-HOMOMORPHISM$(\mathcal{A}', f)$
19: **until there exists** b **such that** $0 \leq b < j \wedge L(\mathcal{A}[j]) = L(\mathcal{A}[b])$;
20: **if** $i > n$ **then return** $\bigcup_{0 \leq k < j} \mathcal{A}[k]$;
21: $p := j - b$;
22: $\mathcal{A}' :=$ PERFORM-FUNCTION$(\mathcal{A}',$ APPLY-N-MULTI-STAR-ONE$, i, \bar{\sigma}|i, p)$;
23: **return** $\bigcup_{0 \leq k < b}$ APPLY-N-MULTI$(\mathcal{A}_0, \bar{\sigma}, \Sigma_1, \Sigma_2, \ldots, \Sigma_n, k)$ $\cup \bigcup_{0 \leq k < p}$ APPLY-N-MULTI$(\mathcal{A}', \sigma, \Sigma_1, \Sigma_2, \ldots, \Sigma_n, k)$
24: **end**.

Figure 7.33: Image of a QDD by the closure of a multisequence of queue operations whose projections are send-synchronized.

function MULTI-META-SET(**set of sequences of** queue operations S,
alphabets $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$) : **set of** functions;

1: **var** S', S'' **sets of sequences of** queue operations;
2: T : **set of** functions;
3: $\bar{\sigma}$: **multisequence of** queue operations;
4: **begin**
5: $T := \emptyset$;
6: $S := S \setminus \{\sigma \in S \mid (\exists i, j \in \{1, \ldots, n\})(i \neq j \wedge$
$((|\Sigma_i| = 1 \wedge |(\sigma|_{i!})| > |(\sigma|_{i?})|) \vee (|\Sigma_i| > 1 \wedge |(\sigma|_{i!})| > 0)) \wedge$
$((|\Sigma_j| = 1 \wedge |(\sigma|_{j!})| > |(\sigma|_{j?})|) \vee (|\Sigma_j| > 1 \wedge |(\sigma|_{j!})| > 0)))\}$;
7: **for each** $(l_1, \ldots, l_n) \in \mathbf{N}^n$ **such that**
$\{\sigma \in S \mid (\forall i \in \{1, \ldots, n\})(|(\sigma|_i)| = l_i)\} \neq \emptyset$ **do**
8: **begin**
9: $S' := \{\sigma \in S \mid (\forall i \in \{1, \ldots, n\})(|(\sigma|_i)| = l_i)\}$;
10: **for each** $(w_1, \ldots, w_n) \in \Sigma_1^* \times \cdots \times \Sigma_n^*$ **such that**
$\{\sigma \in S' \mid (\forall i \in \{1, \ldots, n\})(\sigma|_{i?} = q_i?w_i)\} \neq \emptyset$ **do**
11: **begin**
12: $\bar{\sigma} := \{\sigma \in S' \mid (\forall i \in \{1, \ldots, n\})(\sigma|_{i?} = q_i?w_i)\}$;
13: $T := T \cup \{\bar{\sigma}^*\}$
14: **end**
15: **end**;
16: **return** T
17: **end**.

Figure 7.34: Creation of multicycle meta-transitions.

adaptation simply consists of adding a new type of meta-transition. The definition of a lossy system is based on the following notion.

Definition 7.43 *Let Σ be a finite alphabet. The word $u \in \Sigma^*$ is a* subword *of the word $v \in \Sigma^*$, which is denoted $u \preceq v$, if there exist $m \in \mathbf{N}_0$ and $u_1, u_2, \ldots, u_m, w_0, w_1, \ldots, w_m \in \Sigma^*$ such that $u = u_1 \cdot u_2 \cdots u_m$ and $v = w_0 \cdot u_1 \cdot w_1 \cdot u_2 \cdot w_2 \cdots u_m \cdot w_m$.*

We are now ready to define lossy systems.

Definition 7.44 *A* Lossy QSMA (LQSMA) *is an ESMA $(C, c_0, M, m_0, Op, T, \bar{T})$ such that*

- *Its memory domain M is of the form $\Sigma_1^* \times \Sigma_2^* \times \cdots \times \Sigma_n^*$, where $n \geq 0$ is the* number of queues *of the LQSMA, and each Σ_i $(1 \leq i \leq n)$ is the* queue alphabet *of the i-th queue q_i of the LQSMA;*
- *Its set of memory operations Op contains only send and receive operations. Formally, we have*

$$Op = \{q_i!u \mid 1 \leq i \leq n \wedge u \in \Sigma_i^*\} \cup \{q_i?u \mid 1 \leq i \leq n \wedge u \in \Sigma_i^*\}.$$

- *Its set of meta-transitions $\bar{T}$ contains for every control location $c \in C$ a meta-transition (c, f_l, c), where f_l is the function*

$$f_l : 2^M \to 2^M : U \mapsto \{(u_1, \ldots, u_n) \mid (\exists (v_1, \ldots, v_n) \in U) \\ (u_1 \preceq v_1 \wedge \cdots \wedge u_n \preceq v_n)\},$$

with $M = \Sigma_1^* \times \cdots \times \Sigma_n^*$. Intuitively, those meta-transitions are introduced in order to model the losses that can occur in each control location.

Transforming a non-lossy system into a lossy one can thus be done by simply adding at each control location $c \in C$ a meta-transition (c, f_l, c). In order to be able to compute the image of a set of states by such a meta-transition, it is therefore necessary to dispose of an algorithm for applying the function f_l to a set of queue-set contents represented as a QDD. Such an algorithm is easily obtained by remarking that the effect of f_l is to choose non-deterministically between removing or keeping unchanged each symbol composing queue contents, this choice being allowed to differ between symbols. A QDD representing the set $f_l(U)$ can be computed from a QDD representing a set of queue-set contents $U \subseteq M$ by bypassing each transition of the QDD by an additional transition labeled by the empty word. An algorithm formalizing this construction is given in Figure 7.35.

Theorem 7.45 *Let $q_1, q_2, \ldots, q_n$ be queues of respective alphabets $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ and $\mathcal{A}$ be a QDD representing the set $U \subseteq \Sigma_1^* \times \Sigma_2^* \times \cdots \times \Sigma_n^*$. APPLY-LOSS($\mathcal{A}$) is a QDD representing the set $f_l(U)$.*

Proof Immediate. □

function APPLY-LOSS(QDD $(\Sigma, S, \Delta, I, F)$) : QDD

1: **begin**
2: $(\Sigma, S, \Delta, I, F) :=$ NORMALIZE($(\Sigma, S, \Delta, I, F)$);
3: $\Delta := \Delta \cup \{(s, \varepsilon, s') \in S \times \{\varepsilon\} \times S \mid (\exists (s_1, w, s_2) \in \Delta)(s_1 = s \wedge s_2 = s')\}$;
4: **return** $(\Sigma, S, \Delta, I, F)$
5: **end**.

Figure 7.35: Image of a QDD by the memory function modeling loss.

7.6 Model Checking with Cycle Meta-Transitions

This section is aimed at providing an algorithm for applying to sequences of queue operations the function ITERABLE required by the algorithms introduced in Chapter 4. In the present context, the purpose of ITERABLE is to determine, given a sequence σ of queue operations, a representation of the set of all the queue-set contents to which σ can be applied infinitely many times. A similar problem, consisting of deciding whether a sequence can be repeatedly applied an infinite number of times to a given queue-set content, is addressed in [JJ93, FM96]. We consider successively the cases of systems with one and then with an arbitrary number of queues.

7.6.1 Systems with One Queue

Let q be a queue of alphabet Σ, and let σ be a sequence of elementary queue operations on q. We first assume that σ is of the form $\sigma = (\sigma_?; \sigma_!)$, i.e., that the sequence begins with all its receive operations.

If there exists a queue content $u \in \Sigma^*$ to which σ can be applied infinitely many times, then σ is such that $|\sigma_!| \geq |\sigma_?|$ (otherwise, the length of the queue content would decrease at each application of σ). Therefore, if $|\sigma_!| < |\sigma_?|$, then ITERABLE$(\sigma) = \emptyset$.

If σ is such that $\sigma = \sigma_!$, then we have ITERABLE$(\sigma) = \Sigma^*$. It thus remains to study the case for which $|\sigma_!| \geq |\sigma_?|$.

Assume that $|\sigma_!| \geq |\sigma_?|$, and let $u \in \Sigma^*$ be a queue content to which σ can be applied infinitely many times. We consider the greatest integer $k \geq 0$ such that $\mu(\sigma_?)^k \in \text{pre}(u)$. The word u is thus of the form $u = \mu(\sigma_?)^k \cdot u'$, with $u' \in \Sigma^*$ and $\mu(\sigma_?) \notin \text{pre}(u')$. Applying k times σ to u, we obtain the word $u_2 = \sigma^k(u) = u' \cdot \mu(\sigma_!)^k$. By hypothesis, σ can be applied one more time to u_2, which implies that we have $u' \in \text{pre}(\mu(\sigma_?)) \setminus \{\mu(\sigma_?)\}$.

Let $g = \gcd(|\sigma_!|, |\sigma_?|)$. By hypothesis, σ can be applied an arbitrary number of

times to u_2. We choose to apply it exactly $l = |\sigma_!|/g$ times. We thus have

$$\mu(\sigma_?)^l \in \mathrm{pre}(u_2 \cdot \mu(\sigma_!)^l),$$

which implies

$$\mu(\sigma_?)^l \in \mathrm{pre}(u' \cdot \mu(\sigma_!)^l).$$

Reasoning on the lengths of the words, we obtain $u' \cdot \mu(\sigma_!)^{l'} = \mu(\sigma_?)^l \cdot u'$, with $l' = |\sigma_?|/g$.

Reciprocally, if $w \in \Sigma^*$ is such that there exists $u' \in \mathrm{pre}(\mu(\sigma_?)) \setminus \{\mu(\sigma_?)\}$ such that

- $w \in (\mu(\sigma_?))^+ \cdot u'$, and
- $u' \cdot \mu(\sigma_!)^{l'} = \mu(\sigma_?)^l \cdot u'$ for $l = |\sigma_!|/\gcd(|\sigma_!|, |\sigma_?|)$ and $l' = |\sigma_?|/\gcd(|\sigma_!|, |\sigma_?|)$,

then σ can be applied infinitely many times to w. It follows that we have

$$\mathrm{ITERABLE}(\sigma) = (\mu(\sigma_?))^+ \cdot U,$$

where $U = \{u' \in \mathrm{pre}(\mu(\sigma_?)) \mid u' \neq \mu(\sigma_?) \wedge u' \cdot \mu(\sigma_!)^{l'} = \mu(\sigma_?)^l \cdot u'\}$.

The problem is thus fully solved when $\sigma = (\sigma_?; \sigma_!)$. Let us now generalize our solution to arbitrary sequences of operations involving q.

Let σ be such a sequence. An interesting property is that σ can always be applied to a queue content $w \in \Sigma^*$ to which the sequence $(\sigma_?; \sigma_!)$ can be applied. As a consequence, the set $\mathrm{ITERABLE}(\sigma)$ is a superset of the set $\mathrm{ITERABLE}((\sigma_?; \sigma_!))$. Computing the former set can thus be reduced to computing the set of all the queue contents to which σ can be applied infinitely many times and to which $(\sigma_?; \sigma_!)$ cannot.

Let w be such a word. Its existence implies $|\sigma_!| \geq |\sigma_?|$. The sequences σ and $(\sigma_?; \sigma_!)$ are indistinguishable when applied to words whose length is greater or equal to $|\sigma_?|$. This yields $|w| < |\sigma_?|$. Since, by hypothesis, σ can be applied to w, we have $w \in \mathrm{pre}(\mu(\sigma_?)) \setminus \{\mu(\sigma_?)\}$.

Reciprocally, if $w \in \mathrm{pre}(\mu(\sigma_?)) \setminus \{\mu(\sigma_?)\}$, then there are two possible situations:

- *If* $|\sigma_!| = |\sigma_?|$. Then, σ can be applied infinitely many times to w if and only if $\sigma(w) = w$;
- *If* $|\sigma_!| > |\sigma_?|$. Then, σ can be applied infinitely many times to w if and only if after applying σ a sufficiently large number of times k to w (the exact condition being $|\sigma^k(w)| \geq |\sigma_?|$), one obtains a queue content $\sigma^k(w)$ that belongs to $\mathrm{ITERABLE}((\sigma_?; \sigma_!))$.

An algorithm[4] formalizing the computation developed in this section is given in Figure 7.36.

[4]In this algorithm, QDDs representing simple languages are denoted by the corresponding regular expressions.

```
function ITERABLE-ONE(sequence of queue operations σ, alphabet Σ) : QDD
 1:  var g, k, l, l' : integers;
 2:      U : set of words;
 3:      A, A' : QDDs;
 4:  begin
 5:      if |σ!| < |σ?| then return ∅;
 6:      if |σ?| = 0 then return Σ*;
 7:      if σ = (σ?; σ!) then
 8:          begin
 9:              g := gcd(|σ?|, |σ!|);
10:              l := |σ!|/g;
11:              l' := |σ?|/g;
12:              U := {u ∈ pre(μ(σ?)) | u ≠ μ(σ?) ∧ u · μ(σ!)^l' = μ(σ?)^l · u};
13:              return ⋃_{u∈U} (μ(σ?))^+ · u
14:          end
15:      else
16:          begin
17:              A := A' := ITERABLE-ONE((σ?; σ!), Σ);
18:              for each u ∈ pre(μ(σ?)) \ {μ(σ?)} do
19:                  begin
20:                      if |σ!| = |σ?|  ∧ σ(u) = u then
21:                          A' := UNION(A', u);
22:                      if |σ!| > |σ?| then
23:                          begin
24:                              k = ⌈(|σ?| − |u|)/(|σ!| − |σ?|)⌉;
25:                              if INCLUDED?(APPLY-ONE(σ^k, u), A)
26:                                  then A' := UNION(A', u)
27:                          end
28:                  end
29:          end;
30:      return A'
31:  end.
```

Figure 7.36: Set of queue contents to which a sequence can be applied infinitely many times (one queue).

```
function ITERABLE(sequence of queue operations σ, alphabets Σ_1, Σ_2, ..., Σ_n) : QDD
1:  var i : integer;
2:      A : QDD;
3:  begin
4:      A := ε;
5:      for i := 1 to n do
6:          A := CONCATENATE(A, ITERABLE-ONE(σ|_i, Σ_i));
7:      return A
8:  end.
```

Figure 7.37: Set of queue-set contents to which a sequence can be applied infinitely many times (any number of queues).

Theorem 7.46 *Let q be a queue of alphabet Σ, and σ be a sequence of operations involving q. ITERABLE-ONE(σ, Σ) is a QDD representing the set of all the queue contents $w \in \Sigma^*$ such that σ can be applied infinitely many times to w.*

Proof The algorithm of Figure 7.36 is a direct implementation of the computation method described in this section. □

7.6.2 Systems with Any Number of Queues

The results of Section 7.6.1 can be straightforwardly generalized to sequences of operations involving more than one queue. Indeed, the sequence σ involving the queues $q_1, q_2, \ldots q_n$ can be applied infinitely many times to a queue-set content $(w_1, w_2, \ldots, w_n)$ if and only if for every $i \in \{1, 2, \ldots, n\}$, the sequence $\sigma|_i$ can be applied infinitely many times to the queue content w_i. The set of queue-set contents to which σ can be applied infinitely many times is thus the Cartesian product of the sets of contents of each queue to which the corresponding projection of σ can be applied infinitely many times. An algorithm formalizing this method is given in Figure 7.37.

Theorem 7.47 *Let $q_1, q_2, \ldots, q_n$ be queues of respective alphabets $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$, and σ be a sequence of operations involving those queues. ITERABLE$(\sigma, \Sigma_1, \Sigma_2, \ldots, \Sigma_n)$ is a QDD representing the set of all the queue-set contents $u \in \Sigma_1^* \times \cdots \times \Sigma_n^*$ such that σ can be applied infinitely many times to u.*

Proof Immediate. □

7.7 Model Checking with Multicycle Meta-Transitions

The computation of ITERABLE by the method exposed in Sections 7.6.1 and 7.6.2 can be generalized to send-synchronized multisequences of queue operations. The basic idea consists of replacing each reference to the send operations composing a sequence by a non-deterministic choice between the sequences of send operations composing the multisequence of interest.

7.7.1 Systems with One Queue

Let q be a queue of alphabet Σ, and let $\bar{\sigma} = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ $(m \geq 0)$ be a send-synchronized multisequence of elementary queue operations on q. We first assume that each component σ_i of $\bar{\sigma}$ is of the form $\sigma_i = (\sigma_{i?}; \sigma_{i!})$.

For the same reasons as in Section 7.6.1, if $|\bar{\sigma}_!| < |\bar{\sigma}_?|$, then $\text{ITERABLE}(\bar{\sigma}) = \emptyset$. Moreover, if $|\bar{\sigma}_?| = 0$, then $\text{ITERABLE}(\bar{\sigma}) = \Sigma^*$.

Assume that $|\bar{\sigma}_!| \geq |\bar{\sigma}_?|$, and let $u \in \Sigma^*$ be a queue content to which $\bar{\sigma}$ can be applied infinitely many times. We consider the greatest integer $k \in \mathbf{N}$ such that $\mu(\bar{\sigma}_?)^k \in \text{pre}(u)$. The word u is thus of the form $u = \mu(\bar{\sigma}_?)^k \cdot u'$, with $u' \in \Sigma^*$ and $\mu(\bar{\sigma}_?) \notin \text{pre}(u')$. Applying k times $\bar{\sigma}$ to u, we obtain the word $u_2 = \bar{\sigma}^k(u) = u' \cdot w_1 \cdot w_2 \cdots w_k$, where $w_1, \ldots, w_k \in \{v \in \mu(\sigma_{i!}) \mid 1 \leq i \leq m\}$. By hypothesis, $\bar{\sigma}$ can be applied one more time to u_2, which implies that we have $u' \in \text{pre}(\mu(\bar{\sigma}_?)) \setminus \{\mu(\bar{\sigma}_?)\}$.

Let $g = \gcd(|\bar{\sigma}_!|, |\bar{\sigma}_?|)$. Applying $\bar{\sigma}$ exactly $l = |\bar{\sigma}_!|/g$ times to u_2 and reasoning on the lengths of the words, we obtain $u' \cdot w_1 \cdot w_2 \cdots w_{l'} = \mu(\bar{\sigma}_?)^l \cdot u'$, with $l' = |\bar{\sigma}_?|/g$ and $w_1, \ldots, w_{l'} \in \{v \in \mu(\sigma_{i!}) \mid 1 \leq i \leq m\}$.

Reciprocally, if $w \in \Sigma^*$ is such that there exist $u' \in \text{pre}(\mu(\bar{\sigma}_?)) \setminus \{\mu(\bar{\sigma}_?)\}$ and $w_1, w_2, \ldots, w_{l'} \in \{v \in \mu(\sigma_{i!}) \mid 1 \leq i \leq m\}$, where $l' = |\bar{\sigma}_?|/\gcd(|\bar{\sigma}_!|, |\bar{\sigma}_?|)$, such that

- $w \in (\mu(\bar{\sigma}_?))^+ \cdot u'$, and
- $u' \cdot w_1 \cdot w_2 \cdots w_{l'} = \mu(\bar{\sigma}_?)^l \cdot u'$ for $l = |\bar{\sigma}_!|/\gcd(|\bar{\sigma}_!|, |\bar{\sigma}_?|)$,

then $\bar{\sigma}$ can be applied infinitely many times to w.

It follows that we have

$$\text{ITERABLE}(\bar{\sigma}) = (\mu(\bar{\sigma}_?))^+ \cdot U,$$

where $U = \{u' \in \text{pre}(\mu(\bar{\sigma}_?)) \mid (\exists w_1, \ldots, w_{l'} \in \{v \in \mu(\sigma_{i!}) \mid 1 \leq i \leq m\})(u' \neq \mu(\bar{\sigma}_?) \wedge u' \cdot w_1 \cdot w_2 \cdots w_{l'} = \mu(\bar{\sigma}_?)^l \cdot u'\}$.

Like in Section 7.6.1, computing the value of ITERABLE for an arbitrary multisequence $\bar{\sigma}$ of queue operations can be reduced to computing the set of all the queue contents to which $\bar{\sigma}$ can be applied infinitely many times and to which $\{(\sigma_{1?}; \sigma_{1!}), (\sigma_{2?}; \sigma_{2!}), \ldots, (\sigma_{m?}; \sigma_{m!})\}$ cannot.

Let w be such a queue content. Its existence implies $|\sigma_!| \geq |\bar{\sigma}_?|$ and $|w| < |\bar{\sigma}_?|$. Since, by hypothesis, $\bar{\sigma}$ can be applied to w, we deduce $w \in \mathrm{pre}(\mu(\bar{\sigma}_?)) \setminus \{\mu(\bar{\sigma}_?)\}$.

Reciprocally, if $w \in \mathrm{pre}(\mu(\bar{\sigma}_?)) \setminus \{\mu(\bar{\sigma}_?)\}$, then there are two possible situations:

- *If* $|\bar{\sigma}_!| = |\bar{\sigma}_?|$. Then, $\bar{\sigma}$ can be applied infinitely many times to w if and only if $w \in \bar{\sigma}(w)$;

- *If* $|\bar{\sigma}_!| > |\bar{\sigma}_?|$. Then, $\bar{\sigma}$ can be applied infinitely many times to w if and only if after applying $\bar{\sigma}$ a sufficiently large number of times k to w (the exact condition being $|\bar{\sigma}^k(w)| \geq |\bar{\sigma}_?|$), one obtains a set of queue contents $\bar{\sigma}^k(w)$ that contains at least one element of ITERABLE$(\{(\sigma_{1?}; \sigma_{1!}), (\sigma_{2?}; \sigma_{2!}), \ldots, (\sigma_{m?}; \sigma_{m!})\})$.

An algorithm formalizing the computation developed in this section is given in Figures 7.38 and 7.39.

Theorem 7.48 *Let q be a queue of alphabet Σ, and $\bar{\sigma}$ be a send-synchronized multisequence of operations involving q. MULTI-ITERABLE-ONE$(\bar{\sigma}, \Sigma)$ is a QDD representing the set of all the queue contents $w \in \Sigma^*$ such that $\bar{\sigma}$ can be applied infinitely many times to w.*

Proof The algorithm of Figures 7.38 and 7.39 is a direct implementation of the computation method described in this section. □

7.7.2 Systems with Any Number of Queues

For multisequences involving more than one queue, the situation is similar to the one of Section 7.6.2. The multisequence $\bar{\sigma}$ involving the queues $q_1, q_2, \ldots q_n$ can be applied infinitely many times to a queue-set content $(w_1, w_2, \ldots, w_n)$ if and only if for every $i \in \{1, 2, \ldots, n\}$, the multisequence $\bar{\sigma}|_i$ can be applied infinitely many times to the queue content w_i. An algorithm computing a representation of the set of queue-set contents to which a multisequence $\bar{\sigma}$ whose components are send-synchronized can be applied infinitely many times is given in Figure 7.40. Its correctness is expressed by the following theorem.

Theorem 7.49 *Let $q_1, q_2, \ldots, q_n$ be queues of respective alphabets $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ and $\bar{\sigma} = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ $(m \geq 0)$ be a multisequence of operations involving those queues, such that for every $i \in \{1, 2, \ldots, m\}$, the sequence $\sigma|_i$ is send-synchronized. MULTI-ITERABLE$(\bar{\sigma}, \Sigma_1, \Sigma_2, \ldots, \Sigma_n)$ returns a QDD representing the set of all the queue-set contents $u \in \Sigma_1^* \times \cdots \times \Sigma_n^*$ such that $\bar{\sigma}$ can be applied infinitely many times to u.*

Proof Immediate. □

function MULTI-ITERABLE-ONE(**multisequence of** queue operations $\bar{\sigma} = \{\sigma_1, \ldots, \sigma_m\}$, alphabet Σ) : QDD

```
 1:  var g, k, l, l' : integers;
 2:      U : set of words;
 3:      A, A' : QDDs;
 4:  begin
 5:      if |σ̄!| < |σ̄?| then return ∅;
 6:      if |σ̄?| = 0 then return Σ*;
 7:      if (∀i ∈ {1,...,m})(σ_i = (σ_i?; σ_i!)) then
 8:          begin
 9:              g := gcd(|σ̄?|, |σ̄!|);
10:              l := |σ̄!|/g;
11:              l' := |σ̄?|/g;
12:              U := {u ∈ pre(μ(σ̄?)) | (∃w_1, ... w_l' ∈ {v ∈ μ(σ_i!) | 1 ≤ i ≤ m})
                                        (u ≠ μ(σ̄?) ∧ u · w_1 ··· w_l' = μ(σ̄?)^l · u)};
13:              return ⋃_{u∈U} (μ(σ̄?))^+ · u
14:          end
15:      else
16:          begin
17:              A := A' := MULTI-ITERABLE-ONE({(σ_1?; σ_1!), (σ_2?; σ_2!),
                                                ..., (σ_m?; σ_m!)}, Σ);
18:              for each u ∈ pre(μ(σ̄?)) \ {μ(σ̄?)} do
19:                  begin
20:                      if |σ̄!| = |σ̄?|  ∧ u ∈ σ̄(u) then
21:                          A' := UNION(A', u);

    (...)
```

Figure 7.38: Set of queue contents to which a send-synchronized multisequence can be applied infinitely many times (one queue).

```
    (...)
22:                          if |σ̄!| > |σ̄?| then
23:                              begin
24:                                  k = ⌈(|σ̄?| − |u|)/(|σ̄!| − |σ̄?|)⌉;
25:                                  if ¬(EMPTY?(INTERSECTION(
                                         APPLY-N-ONE(σ̄, u, k), A))) then
26:                                      A′ := UNION(A′, u)
27:                              end
28:                      end
29:              end;
30:          return A′
31:     end.
```

Figure 7.39: Set of queue contents to which a send-synchronized multisequence can be applied infinitely many times (one queue, continued).

```
function MULTI-ITERABLE( multisequence of queue operations σ̄,
                                        alphabets Σ1, Σ2, ..., Σn) : QDD
1:    var i : integer;
2:        A : QDD;
3:    begin
4:        A := ε;
5:        for i := 1 to n do
6:            A := CONCATENATE(A, MULTI-ITERABLE-ONE(σ̄|i, Σi));
7:        return A
8:    end.
```

Figure 7.40: Set of queue-set contents to which a multisequence can be applied infinitely many times (any number of queues).

7.8 Termination

The goal of this section is to give algorithms for computing over QSMAs the truth value of the predicates required by Sections 5.1 to 5.5. Specifically, we implement the predicates FINITE?, whose purpose is to decide the finiteness of a set of queue-set contents represented as a QDD, and PRECEDES?, which checks whether two sequences σ_1 and σ_2 of queue operations are such that $\sigma_1 \lhd \sigma_2$. We address each problem separately.

7.8.1 Finiteness of Sets of Queue-Set Contents

Deciding the finiteness of a set of queue-set contents represented as a QDD is easy. Since the sequential encoding of queue-set contents is one-to-one, i.e., since it associates exactly one encoding to each queue-set content, this can be done by testing whether the language accepted by the QDD is infinite. A simple way of performing this test is to use a variant of the depth-first search algorithm of Figure 3.6 used for detecting simple cycles in the control graph of SMAs. The adapted algorithm is given in Figure 7.41.

Theorem 7.50 *Let $q_1, q_2, \ldots, q_n$ $(n \geq 0)$ be queues of respective alphabets $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$, and $\mathcal{A}$ be a QDD representing the set $U \subseteq \Sigma_1^* \times \Sigma_2^* \times \cdots \times \Sigma_n^*$. We have FINITE?$(\mathcal{A}) = \mathbf{T}$ if and only if U is finite.*

Proof A finite-state automaton accepts an infinite language if and only if there exists in its transition graph at least one simple cycle $\mathcal{C}$ such that:

- The body of $\mathcal{C}$ is not labeled by the empty word;
- The states visited by $\mathcal{C}$ are reachable from at least one initial state of the automaton;
- There exists at least one accepting state of the automaton that is reachable from the states visited by $\mathcal{C}$.

The goal of Lines 26–28 is to compute the set S of all the reachable states of $\mathcal{A}$. This is done by a depth-first search algorithm analogous to the one in Figure 3.2. Then, at Lines 29–30, a search for a simple cycle satisfying the abovementioned conditions is carried out. This search is a simple variant of the algorithm in Figure 3.6. □

7.8.2 Precedence Relation

Let $q_1, q_2, \ldots, q_n$ $(n \geq 0)$ be queues of respective alphabets $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$. The problem addressed here consists of deciding whether two sequences of queue operations σ_1 and σ_2 involving those queues are such that $\sigma_1 \lhd \sigma_2$, i.e., whether $(\sigma_2; \sigma_1)(U) \subseteq (\sigma_1; \sigma_2)(U)$ for every set of queue-set contents $U \subseteq \Sigma_1^* \times \cdots \times \Sigma_n^*$.

```
function FINITE?(automaton (Σ, S, Δ, I, F)) : {T, F}
 1:  var node : array[0, 1, ...] of states;
 2:      edge : array[0, 1, ...] of words;
 3:      S' : set of states;
 4:      s : state;
 5:  procedure explore-fw(state s)
 6:      begin
 7:          S' := S' ∪ {s};
 8:          for each (s', w, s'') ∈ Δ such that s' = s do
 9:              if s'' ∉ S' then explore-fw(s'')
10:      end;
11:  procedure explore-bw(state s, integer depth)
12:      begin
13:          node[depth] := s;
14:          for each (s'', w, s') ∈ Δ such that s' = s ∧ s'' ∈ S' do
15:              begin
16:                  edge[depth] := w;
17:                  if (∃i, 0 ≤ i ≤ depth) such that node[i] = s'' then
18:                      begin
19:                          if edge[i] · edge[i + 1] ··· edge[depth] ≠ ε then
20:                              return F
21:                      end
22:                  else explore-bw(s'', depth + 1)
23:              end
24:      end;
25:  begin
26:      S' := ∅;
27:      for each s ∈ I do
28:          explore-fw(s);
29:      for each s ∈ F do
30:          explore-bw(s, 0);
31:      return T
32:  end.
```

Figure 7.41: Test of finiteness of the language accepted by an automaton.

An noteworthy property is that two queue operations that are adjacent within a sequence and that involve different queues can be permuted. In other words, if the sets of queues involved in two sequences of queue operations σ and σ' are disjoint, then $(\sigma;\sigma')(U) = (\sigma';\sigma)(U)$ for every set of queue-set contents $U \subseteq \Sigma_1^* \times \cdots \times \Sigma_n^*$. It follows that we have $\sigma_1 \lhd \sigma_2$ if and only if $(\sigma_1|_i) \lhd (\sigma_2|_i)$ for every $i \in \{1, 2, \ldots, n\}$.

We can therefore reduce the problem and assume that σ_1 and σ_2 contain only operations involving a single queue q of alphabet Σ.

If σ_1 and σ_2 are such that $\sigma_1 \lhd \sigma_2$, then we have $\mu(\sigma_{1?};\sigma_{2?}) = \mu(\sigma_{2?};\sigma_{1?})$. Indeed, choosing $U = \{\mu(\sigma_{2?};\sigma_{1?})\}$, we have

$$\begin{array}{rcl}(\sigma_2;\sigma_1)(U) & \subseteq & (\sigma_1;\sigma_2)(U)\\ \{\mu(\sigma_{2!};\sigma_{1!})\} & \subseteq & (\sigma_1;\sigma_2)(\{\mu(\sigma_{2?};\sigma_{1?})\}),\end{array}$$

which implies the result since the right-hand side must be nonempty. Furthermore, developing the previous equation yields

$$\mu(\sigma_{1!};\sigma_{2!}) = \mu(\sigma_{2!};\sigma_{1!}).$$

Hence, if σ_1 and σ_2 are such that $\mu(\sigma_{1?};\sigma_{2?}) \neq \mu(\sigma_{2?};\sigma_{1?})$ or such that $\mu(\sigma_{1!};\sigma_{2!}) \neq \mu(\sigma_{2!};\sigma_{1!})$, then we have $\sigma_1 \not\lhd \sigma_2$.

Assume now that σ_1 and σ_2 are such that $\mu(\sigma_{1?};\sigma_{2?}) = \mu(\sigma_{2?};\sigma_{1?})$ and $\mu(\sigma_{1!};\sigma_{2!}) = \mu(\sigma_{2!};\sigma_{1!})$. For every queue content $w \in \Sigma^*$, if $(\sigma_1;\sigma_2)(\{w\})$ and $(\sigma_2;\sigma_1)(\{w\})$ are both nonempty, then there must be equal to each other. Indeed, both of them then contain the word obtained by removing the first $|\sigma_{1?}| + |\sigma_{2?}|$ symbols from the word $w \cdot \mu(\sigma_{1!};\sigma_{2!})$.

As a consequence, it is sufficient to check whether there exist words w such that $(\sigma_2;\sigma_1)(\{w\})$ is nonempty and $(\sigma_1;\sigma_2)(\{w\})$ is empty. The only words that are worth checking are those of length less than $|\sigma_{2?};\sigma_{1?}|$ (otherwise $(\sigma_2;\sigma_1)(\{w\})$ would be identical to $(\sigma_1;\sigma_2)(\{w\})$) and that belong to the set of prefixes of $\mu((\sigma_{2?};\sigma_{1?}))$ (otherwise, $(\sigma_2;\sigma_1)(\{w\})$ would be empty). Since there are finitely many such words, a simple solution consists of testing them one by one.

An algorithm formalizing this construction is given in Figure 7.42. Its correctness is established by the following theorem.

Theorem 7.51 *Let $q_1, q_2, \ldots, q_n$ be queues of respective alphabets $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$, and let σ_1, σ_2 be two sequences of operations involving those queues. We have PRECEDES?$(\sigma_1,\ \sigma_2, \Sigma_1, \Sigma_2, \ldots, \Sigma_n) = \mathbf{T}$ if and only if $\sigma_1 \lhd \sigma_2$.*

Proof The algorithm directly implements the decision procedure developed in this section. □

```
function PRECEDES?( sequences of queue operations σ1, σ2,
                                          alphabets Σ1, Σ2, ..., Σn) : {T, F}
 1:  var σ, σ' : sequences of queue operations;
 2:      i : integer;
 3:      w : word;
 4:  begin
 5:      for i := 1 to n do
 6:          begin
 7:              σ := (σ1; σ2)|i;
 8:              σ' := (σ2; σ1)|i;
 9:              if σ! ≠ σ'! ∨ σ? ≠ σ'? then
10:                  return F;
11:              for each w ∈ pre(μ(σ'?)) such that w ≠ μ(σ'?) do
12:                  if ¬INCLUDED?(APPLY(w, σ', Σ1, Σ2, ..., Σn),
                                        APPLY(w, σ, Σ1, Σ2, ..., Σn)) then
13:                      return F
14:          end;
15:      return T
16:  end.
```

Figure 7.42: Precedence test for sequences of queue operations.

7.9 Loop Optimization

Regrettably, loop optimization cannot be applied to systems using FIFO queues on which only send and receive operations are performed. The main reason is the relative lack of expressiveness of send and receive operations. Indeed, even though it has been shown that QSMAs are Turing-expressive, it is impossible in general to simulate the effect of a loop with a single sequence of send and receive operations. Let us illustrate this phenomenon in the case of a system with only one queue q whose alphabet is Σ. Let σ be a nonempty sequence of queue operations labeling a cycle, and θ be an elementary queue operation labeling the exit transition from this cycle. Assume that there exists a sequence σ' of queue operations such that for every queue content $w \in \Sigma^*$, $(\sigma^+;\theta)(\{w\}) = \sigma'(\{w\}) \neq \emptyset$. The existence of σ' implies $|\sigma_?| > 0$ (indeed, if $\sigma = \sigma_!$, then $(\sigma^+;\theta)(\{w\})$ contains more than one element and is thus different from $\sigma'(\{w\})$). Let us add the prefix $\mu(\sigma_?)$ to w. We obtain $(\sigma^+;\theta)(\mu(\sigma_?) \cdot w) = \sigma'(w) \cdot \mu(\sigma_!)$ which is, in general, different from $\sigma'(\mu(\sigma_?) \cdot w)$. This proves that σ' cannot be made equivalent to $(\sigma^+;\theta)$. As a consequence, we have EXISTS-LOOP-EQUIV? $\equiv \mathbf{F}$ for the systems studied in this chapter.

Chapter 8

Systems Using Integer Variables

In this chapter, we particularize the results of Chapters 3–6 to a specific class of infinite-state systems: those whose memory is composed of a finite number of unbounded integer variables on which linear operations are performed. Such systems form a very flexible model for reasoning about computer programs and hardware circuits. A particular subset of those systems, the *Petri net* [Pet81, Rei85], is extensively used for modeling various types of hardware and software systems. Like in the case of the FIFO channels studied in Chapter 7, considering unbounded rather than bounded integer variables provides a useful abstraction that allows to reason about systems without being influenced by their implementation details or physical limitations.

The structure of this chapter is similar to the one of Chapter 7. First, we introduce systems using unbounded integer variables and define their syntax, semantics, and elementary memory operations. Then, we show that such systems are Turing-expressive, and propose an encoding scheme which leads to a powerful finite-state representation system for sets of integer-vector values, the *Number Decision Diagram (NDD)*. Finally, we give algorithms implementing with NDDs all the predicates and functions required by Chapters 3 to 6. In particular, we will present an original decision procedure for determining whether the closure of a linear transformation preserves the representability by NDDs of sets of memory contents.

8.1 Basic Notions

8.1.1 Integer SMAs

Let $n \in \mathbf{N}$ be a finite *dimension*. An *integer vector* (or simply *vector*) of dimension n is an object whose value is an element of $\mathbf{Z}^n$. We define one elementary operation over integer vectors, the *linear operation*, which consists of applying a linear transformation to the value of the vector, provided that this value satisfies a condition

expressed as a system of linear inequations. Formally, a linear operation is characterized by an integer $m \in \mathbf{N}$, two matrices $A \in \mathbf{Z}^{n \times n}$ and $P \in \mathbf{Z}^{m \times n}$, two vectors $\vec{b} \in \mathbf{Z}^n$ and $\vec{q} \in \mathbf{Z}^m$, and is denoted $P\vec{x} \leq \vec{q} \rightarrow \vec{x} := A\vec{x} + \vec{b}$. Its semantics is defined by the function

$$(P\vec{x} \leq \vec{q} \rightarrow \vec{x} := A\vec{x} + \vec{b}) : \mathbf{Z}^n \rightarrow \mathbf{Z}^n : \vec{v} \mapsto A\vec{v} + \vec{b} \quad \text{if } P\vec{v} \leq \vec{q}.$$

Let $\theta = (P\vec{x} \leq \vec{q} \rightarrow \vec{x} := A\vec{x} + \vec{b})$ be a linear operation. The system of linear inequations $P\vec{x} \leq \vec{q}$, which expresses the condition that must be satisfied by the vector values in order to perform the operation, is called the *guard* of θ. If that system is trivial, i.e., if it is satisfied by every vector value in $\mathbf{Z}^n$, then θ is said to be *guardless* and the trivial guard can be omitted in the expression of θ. The assignment $\vec{x} := A\vec{x} + \vec{b}$ that characterizes the modification undergone by vector values is called the *transformation* of θ. If this transformation is trivial, i.e., if A is an identity matrix and if $\vec{b} = \vec{0}$, then it can be omitted in the expression of θ. The matrix A and the vector $\vec{b}$ are respectively called the *transformation matrix* and the *transformation offset* of θ.

The domain of linear operations is extended to sets of vector values in the usual way, i.e., we define

$$(P\vec{x} \leq \vec{q} \rightarrow \vec{x} := A\vec{x} + \vec{b}) : 2^{\mathbf{Z}^n} \rightarrow 2^{\mathbf{Z}^n} : U \mapsto \{A\vec{v} + \vec{b} \mid \vec{v} \in U \wedge P\vec{v} \leq \vec{q}\}.$$

We are now ready to define the class of SMAs that will be studied in this chapter.

Definition 8.1 *An* Integer SMA (ISMA) *is an SMA* (C, c_0, M, m_0, Op, T) *such that*

- *Its memory domain* M *is equal to* $\mathbf{Z}^n$*, where* $n \geq 0$ *is the* dimension *of the ISMA. This is equivalent to saying that the SMA has* n *integer variables. Each memory content* $\vec{v} = (v_1, v_2, \ldots, v_n)$ *is a* vector value*. Each component* $v_i \in \mathbf{Z}$ *(with* $i \in \{1, 2, \ldots, n\}$*) of this vector value can be seen as the value of the* i*-th variable* x_i *of the ISMA;*

- *Its set of memory operations Op contains only linear operations. Formally, we have*

$$\begin{aligned} Op \;=\; \{&(P\vec{x} \leq \vec{q} \rightarrow \vec{x} := A\vec{x} + \vec{b}) \mid \\ &(\exists m \in \mathbf{N})(A \in \mathbf{Z}^{n \times n} \wedge \vec{b} \in \mathbf{Z}^n \wedge P \in \mathbf{Z}^{m \times n} \wedge \vec{q} \in \mathbf{Z}^m)\}. \end{aligned}$$

The notion of *Extended ISMA (EISMA)* is defined similarly.

Definition 8.2 *An* Extended ISMA (EISMA) *is an ESMA* $(C, c_0, M, m_0, Op, T, \bar{T})$ *such that its underlying SMA* (C, c_0, M, m_0, Op, T) *is an ISMA.*

An interesting property of linear operations is that they are closed under the sequential composition, i.e., that every finite sequence of linear operations is always equivalent to a single linear operation. Indeed, if we have

$$\theta_1 = (P_1\vec{x} \leq \vec{q_1} \rightarrow \vec{x} := A_1\vec{x} + \vec{b_1})$$

and

$$\theta_2 = (P_2\vec{x} \leq \vec{q_2} \rightarrow \vec{x} := A_2\vec{x} + \vec{b_2}),$$

with $m_1, m_2 \in \mathbf{N}$, $A_1, A_2 \in \mathbf{Z}^{n\times n}$, $\vec{b_1}, \vec{b_2} \in \mathbf{Z}^n$, $P_1 \in \mathbf{Z}^{m_1\times n}$, $P_2 \in \mathbf{Z}^{m_2\times n}$, $\vec{q_1} \in \mathbf{Z}^{m_1}$ and $\vec{q_2} \in \mathbf{Z}^{m_2}$, then we have

$$(\theta_1; \theta_2) = (P\vec{x} \leq \vec{q} \rightarrow \vec{x} := A\vec{x} + \vec{b}),$$

where

$$P \in \mathbf{Z}^{(m_1+m_2)\times n} = \begin{bmatrix} P_1 \\ P_2A_1 \end{bmatrix},$$

$$\vec{q} \in \mathbf{Z}^{(m_1+m_2)} = \begin{bmatrix} \vec{q_1} \\ \vec{q_2} - P_2\vec{b_1} \end{bmatrix},$$

$$A \in \mathbf{Z}^{n\times n} = A_2A_1,$$

and

$$\vec{b} \in \mathbf{Z}^n = A_2\vec{b_1} + \vec{b_2}.$$

8.1.2 Turing Expressiveness

It is well known that state machines using at least two unbounded integer variables can simulate arbitrary Turing machines. This result is expressed by the following theorem.

Theorem 8.3 *Let $n \geq 2$ be a dimension. The class of all the ISMAs that have the memory domain $\mathbf{Z}^n$ is Turing-expressive.*

Proof According to [HU79], it is sufficient to prove that ISMAs having at least two variables can simulate arbitrary two-counter machines. This result is immediate, since the values of the two counters can be directly represented by the values of two dedicated variables x_1 and x_2, and the operations involving the counters can be translated into linear operations involving x_1 and x_2:

- An increment operation on the counter i (with $i \in \{1, 2\}$) is simulated by the linear operation $\vec{x} := A\vec{x} + \vec{b}$, where $A \in \mathbf{Z}^{n\times n}$ is an identity matrix, and $\vec{b} = (b_1, b_2, \ldots, b_n) \in \mathbf{Z}^n$ is such that $b_i = 1$ and $b_j = 0$ for every $j \neq i$;

- A decrement operation on the counter i (with $i \in \{1, 2\}$) is simulated by the linear operation $P\vec{x} \leq \vec{q} \rightarrow \vec{x} := A\vec{x} + \vec{b}$, where $P = [p_1; p_2; \ldots; p_n] \in \mathbf{Z}^{1\times n}$ is such that $p_i = -1$ and $p_j = 0$ for every $j \neq i$, $\vec{q} \in \mathbf{Z}^1$ is equal to (-1), $A \in \mathbf{Z}^{n\times n}$ is an identity matrix, and $\vec{b} = (b_1, b_2, \ldots, b_n) \in \mathbf{Z}^n$ is such that $b_i = -1$ and $b_j = 0$ for every $j \neq i$;
- A test operation on the counter i (with $i \in \{1, 2\}$) is simulated by the linear operation $P\vec{x} \leq \vec{q}$, where $P = [p_1; p_2; \ldots; p_n] \in \mathbf{Z}^{1\times n}$ is such that $p_i = 1$ and $p_j = 0$ for every $j \neq i$, and $\vec{q} \in \mathbf{Z}^1$ is equal to (0). The operation succeeds if and only if the value of the counter is 0.

□

As it has been shown in Chapter 4, a consequence of this theorem is that the emptiness problem is undecidable for ISMAs associated with a set of accepting control locations if their dimension is at least equal to 2.

8.1.3 Number Decision Diagrams

The first step towards obtaining a representation system for sets of integer vector values that is well suited for ISMAs is to define an encoding scheme for vector values. The encoding scheme we propose for vectors is based on an encoding scheme for integers, which consists of expressing an integer as a finite sequence of *digits* belonging to a finite alphabet. Let us define precisely this encoding of integers.

Let $r \in \mathbf{N}$, with $r > 1$, be a *numeration basis* (or simply *basis*). Any positive integer z can be encoded as a finite word $w = a_{p-1} \cdot a_{p-2} \cdots a_1 \cdot a_0$ ($p \geq 0$) of *digits* belonging to $\{0, 1, \ldots, r-1\}$, such that $z = \sum_{0\leq i<p} a_i r^i$. The encoding of z is not unique. Indeed, its length can be increased at will by adding an arbitrary number of leading "0" digits. This encoding scheme is easily generalized to all the integers in $\mathbf{Z}$ by requiring that the encoding of an integer $z \in \mathbf{Z}$ such that $-r^{p-1} \leq z < r^{p-1}$, where $p > 0$, has at least p digits. If $z < 0$, then the encoding of z consists of the last p digits of the encoding of $r^p + z$ (the number $r^p + z$ is called the *r 's complement* of z). As a consequence, the first digit of the encoding of an integer will be equal to 0 if the number is greater or equal to zero, and to $r - 1$ otherwise (this first digit is called the *sign digit*). The fact that the word $w \in \{0, 1, \ldots, r-1\}^*$ encodes the integer $z \in \mathbf{Z}$ in basis r is denoted $w \in [z]_r$.

In order to represent a vector value, we encode each of its components with an identical number of digits and we group together the digits that share the same position. Any element of $\mathbf{Z}^n$ ($n > 0$) thus has an infinite number of possible encodings, the shortest of which having the length required by the component with the highest magnitude. Precisely, the encoding scheme is defined as follows.

Definition 8.4 *Let $n \geq 0$ be a dimension and $r > 1$ be a basis. The* synchronous encoding scheme *$E_{S(r)}$ is the relation that associates to a vector of $\mathbf{Z}^n$ the tuples*

composed of the same-length encodings in basis r of the components of this vector. Formally, we have

$$E_{S(r)} \subseteq M \times V_{S(r)} = \{((v_1, \ldots, v_n), (w_1, \ldots, w_n)) \mid |w_1| = |w_2| = \cdots = |w_n| \\ \wedge\, w_1 \in [v_1]_r \wedge w_2 \in [v_2]_r \wedge \cdots \wedge w_n \in [v_n]_r\},$$

where $M = \mathbf{Z}^n$ is the domain, and $V_{S(r)} = \bigcup_{k \in \mathbf{N}} (\{0, r-1\} \cdot \{0, \ldots, r-1\}^k)^n$ is the set of valid encodings.

An encoding of an element of $\mathbf{Z}^n$ can indifferently be viewed either as a tuple of n words of identical length over the alphabet $\{0, 1, \ldots, r-1\}$, or as a single word over the alphabet $\{0, 1, \ldots, r-1\}^n$.

The synchronous encoding scheme satisfies the requirements of Definition 6.5. Indeed, by definition, $E_{S(r)}$ is complete over M, and is complete and unambiguous over $V_{S(r)}$. The corresponding decoding function $D_{S(r)}$ is given by the formula

$$D_{S(r)} : 2^{V_{S(r)}} \to 2^M : L \mapsto \{(v_1, \ldots, v_n) \in M \mid (\exists (w_1, w_2, \ldots, w_n) \in L) \\ (w_1 \in [v_1]_r \wedge w_2 \in [v_2]_r \wedge \cdots \wedge w_n \in [v_n]_r)\}.$$

We are now ready to define the representation system for sets of vector values.

Definition 8.5 *Let $n \geq 0$ be a dimension and $r > 1$ be a basis. A* Number Decision Diagram (NDD) *is a finite-state representation of a set $U \subseteq \mathbf{Z}^n$ of vector values based on the synchronous encoding scheme $E_{S(r)}$.*

In other words, an NDD representing a set of vector values $U \subseteq \mathbf{Z}^n$ is simply a finite-state automaton accepting the synchronous encodings of the elements of U.

8.1.4 Representable Sets of Vector Values

Finite-state representations of sets of integer vector values have been studied for a long time [BHMV94]. In [Büc60], Büchi gave the first characterization of recognizable sets of vector values in terms of logic. A flaw was discovered in Büchi's proof by MacNaughton [Mac63], and a correct characterization was proposed in [Mac63] and [Bru85]. Simplified proofs of this characterization can be found in [MP86] and [Vil92]. Precisely, the characterization is expressed as the conjunction of a necessary and of a sufficient conditions. The necessary condition is given by the following theorem.

Theorem 8.6 *Let $n \geq 0$ be a dimension, $r > 1$ be a basis, and $U \subseteq \mathbf{Z}^n$ be a set of vector values. If U is recognizable with respect to the synchronous encoding scheme $E_{S(r)}$, then U is definable in the first-order theory $\langle \mathbf{Z}, \leq, +, V_r \rangle$, where V_r is a function defined as*

$$V_r : \mathbf{Z} \to \mathbf{N} : z \mapsto \begin{cases} \textit{the greatest power of } r \textit{ dividing } z & \textit{if } z \neq 0, \\ 1 & \textit{if } z = 0. \end{cases}$$

Proof Our proof is inspired by the one proposed by Villemaire [Vil92]. Let $\mathcal{A} = (\Sigma, S, \Delta, I, F)$ be an NDD representing U. We assume that $\mathcal{A}$ is in strong normal form. The idea of the proof is to give a method for constructing from $\mathcal{A}$ a formula ψ of $\langle \mathbf{Z}, \leq, +, V_r \rangle$ denoting the set U.

The construction is as follows. By definition of NDDs, $\mathcal{A}$ has a finite number of states. Therefore, there exists $l \in \mathbf{N}_0$ such that $|S| < r^l$. It follows that each state $s \in S$ can be unambiguously identified by l digits $d_1(s), d_2(s), \ldots, d_l(s)$ taken in $\{0, 1, \ldots, r-1\}$. This assignment of unique tuples of digits to states can be chosen arbitrarily provided that no state is associated with the tuple $(0, 0, \ldots, 0)$ (the purpose of this restriction is to simplify future computations). A finite sequence of states $s_0, s_1, \ldots, s_m \in S$ $(m \geq 0)$ can then be identified by a tuple of words $(w_1, w_2, \ldots, w_l) \in (\{0, 1, \ldots, r-1\}^{m+1})^l$, such that

$$\begin{aligned} w_1 &= d_1(s_m) \cdot d_1(s_{m-1}) \cdots d_1(s_1) \cdot d_1(s_0), \\ w_2 &= d_2(s_m) \cdot d_2(s_{m-1}) \cdots d_2(s_1) \cdot d_2(s_0), \\ &\vdots \\ w_l &= d_l(s_m) \cdot d_l(s_{m-1}) \cdots d_l(s_1) \cdot d_l(s_0). \end{aligned}$$

Since each w_i $(1 \leq i \leq l)$ is a word over the alphabet $\{0, 1, \ldots, r-1\}$, the word w_i prefixed by any nonzero number of "0" digits encodes a number $z_i \in \mathbf{N}$. It follows that every finite sequence of states of $\mathcal{A}$ can be unambiguously identified by a single vector value $\vec{z} \in \mathbf{N}^l$. The essence of the construction of ψ is to express in $\langle \mathbf{Z}, \leq, +, V_r \rangle$ the fact that $\vec{z}$ identifies an accepting path of $\mathcal{A}$.

Precisely, the formula $\psi : \mathbf{Z}^n \to \{\mathbf{T}, \mathbf{F}\}$ denoting U can be expressed as

$$\psi(\vec{v}) \equiv (\exists \vec{z} \in \mathbf{Z}^l)(\vec{z} \geq \vec{0} \wedge \vec{z} \neq \vec{0} \wedge \psi'(\vec{z}, \vec{v})),$$

where

$$\psi' : \mathbf{N}^l \times \mathbf{Z}^n : (\vec{z}, \vec{v}) \mapsto \begin{cases} \mathbf{T} & \text{if } z \text{ corresponds to a path of } \mathcal{A} \text{ that accepts a} \\ & \text{synchronous encoding of } \vec{v}, \\ \mathbf{F} & \text{otherwise.} \end{cases}$$

It remains to define the predicate ψ' in $\langle \mathbf{Z}, \leq, +, V_r \rangle$. We use the decomposition

$$\psi'(\vec{z}, \vec{v}) \equiv \psi_1(\vec{z}) \wedge \psi_2(\vec{z}, \vec{v}) \wedge \psi_3(\vec{z}),$$

where

- $\psi_1(\vec{z}) = \mathbf{T}$ if and only if $\vec{z}$ identifies a sequence of states of $\mathcal{A}$ beginning in an initial state;
- $\psi_2(\vec{z}, \vec{v}) = \mathbf{T}$ if and only if there exists a path of $\mathcal{A}$ that visits the sequence of states identified by $\vec{z}$ and that reads a synchronous encoding of $\vec{v}$;

- $\psi_3(\vec{z}) = \mathbf{T}$ if and only if $\vec{z}$ identifies a sequence of states of $\mathcal{A}$ ending in an accepting state.

In order to simplify the expressions of ψ_1, ψ_2 and ψ_3, it is useful to introduce the auxiliary predicate $X_r : \mathbf{Z}^3 \to \{\mathbf{T}, \mathbf{F}\}$, such that $X_r(z, u, k) = \mathbf{T}$ if and only if u is a power of r and the coefficient of this power of r in the development of z in basis r is k. Formally, we have

$$X_r(z, u, k) \equiv V_r(u) = u \wedge \bigwedge_{0 \leq i < r} (k = i \Rightarrow X_r^i(z, u)),$$

where for each $i \in \{0, 1, \ldots, r-1\}$,

$$\begin{aligned} X_r^i(z, u) \;\equiv\; & (\exists z_1, z_2 \in \mathbf{Z})(0 \leq z_1 < u \wedge V_r(z_2) > u \wedge z = z_1 + iu + z_2) \\ & \vee (\exists z_1 \in \mathbf{Z})(0 \leq z_1 < u \wedge z = z_1 + iu). \end{aligned}$$

We are now ready to give the expressions of ψ_1, ψ_2 and ψ_3. We have

$$\psi_1(z_1, z_2, \ldots, z_l) \equiv \bigvee_{s_0 \in I} (\bigwedge_{1 \leq i \leq l} X_r(z_i, 1, d_i(s_0))),$$

$$\begin{aligned} \psi_2(z_1, z_2, \ldots, z_l, v_1, v_2, \ldots, v_n) \equiv (\forall u \in \mathbf{Z})(((V_r(u) = u \wedge \bigvee_{1 \leq i \leq l} ru \leq z_i) \Rightarrow \\ \bigvee_{(s_1, (a_1, \ldots, a_n), s_2) \in \Delta} (\bigwedge_{1 \leq i \leq l} (X_r(z_i, u, d_i(s_1)) \wedge X_r(z_i, ru, d_i(s_2))) \\ \wedge \bigwedge_{1 \leq i \leq n} X_r(v_i, u, a_i))) \\ \wedge ((V_r(u) = u \wedge \bigwedge_{1 \leq i \leq l} ru > z_i) \Rightarrow \bigwedge_{1 \leq i \leq n} \bigvee_{k \in \{0, r-1\}} (X_r(v_i, u, k) \\ \wedge (u = 1 \vee X_r(v_i, u/r, k))))), \end{aligned}$$

and

$$\begin{aligned} \psi_3(z_1, z_2, \ldots, z_l) \equiv (\exists u \in \mathbf{Z})(\bigvee_{s \in F} (V_r(u) = u \wedge \bigwedge_{1 \leq i \leq l} X_r(z_i, u, d_i(s)) \wedge \\ (\forall u' \in \mathbf{Z})((V_r(u') = u' \wedge u' > u) \Rightarrow \bigwedge_{1 \leq i \leq l} X_r(z_i, u', 0)))). \end{aligned}$$

(In these expressions, ru is a shorthand for $u + u + \cdots + u$, in which u appears r times, and u/r represents an integer z such that $rz = u$.) $\square$

The second part of the characterization of the sets of vector values that are recognizable with respect to the synchronous encoding scheme consists of establishing that the condition expressed by Theorem 8.6 is also sufficient, i.e., that all the subsets of $\mathbf{Z}^n$ that are definable in the first-order theory $\langle \mathbf{Z}, \leq, +, V_r \rangle$ are recognizable.

We follow the same approach as in [BHMV94]. Intuitively, if a set $U \subseteq \mathbf{Z}^n$ of vector values is definable in the theory $\langle \mathbf{Z}, \leq, +, V_r \rangle$, then it can be expressed in terms of *basic vector sets* which correspond to the atomic formulas of that theory,

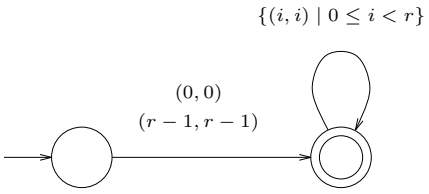

Figure 8.1: NDD representing $U_=$.

and of *basic set operations* which correspond to Boolean operators and first-order quantifiers. The sufficient condition is then proved by showing that the basic vector sets are recognizable, and that the basic set operations preserve the recognizable nature of sets, and can be computed on vector sets represented as NDDs. These two results are established by the four following lemmas.

Lemma 8.7 *Let $r > 1$ be a basis. The vector sets*

$$\begin{aligned} U_= &= \{(v_1, v_2) \in \mathbf{Z}^2 \mid v_1 = v_2\}, \\ U_\leq &= \{(v_1, v_2) \in \mathbf{Z}^2 \mid v_1 \leq v_2\}, \\ U_+ &= \{(v_1, v_2, v_3) \in \mathbf{Z}^3 \mid v_3 = v_1 + v_2\}, \\ U_{V_r} &= \{(v_1, v_2) \in \mathbf{Z}^2 \mid v_2 = V_r(v_1)\}. \end{aligned}$$

are recognizable with respect to the synchronous encoding scheme $E_{S(r)}$.

Proof NDDs[1] representing those sets are given in Figures 8.1, 8.2, 8.3 and 8.4. Let $w_1, w_2, w_3 \in \{0, 1, \ldots, r-1\}^*$ be synchronous encodings of (respectively) $v_1, v_2, v_3 \in \mathbf{Z}$. The principles of operation of the NDDs are the following.

- The NDD representing $U_=$ checks that w_1 and w_2 have the same valid sign digits, and that all their remaining digits are pairwise identical;
- The NDD representing $U_\leq$ first compares the sign digits of w_1 and w_2 and checks that they are valid. If $v_1 < 0$ and $v_2 \geq 0$, then the pair (w_1, w_2) is accepted regardless of the remaining digits. If $v_1 \geq 0$ and $v_2 < 0$, then the pair (w_1, w_2) is not accepted. If v_1 and v_2 have the same sign, then the automaton looks for the leftmost digit that differs between w_1 and w_2. The pair (w_1, w_2) is accepted if the smaller digit is the one in w_1, or if w_1 is identical to w_2;
- The NDD representing U_+ first compares the sign digits of w_1 and w_2 and checks that they are valid. Then, it jumps either to a *carry* or to a *non-carry* state. The transitions of the automata are labeled so as to meet the following requirements:

[1] In these automata, sets of transition labels are introduced as a shorthand for sets of transitions sharing the same origin and end.

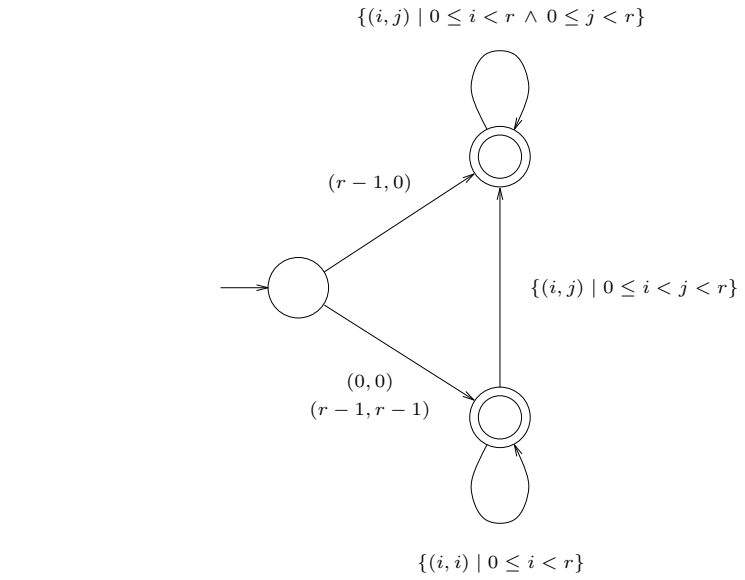

Figure 8.2: NDD representing $U_{\le}$.

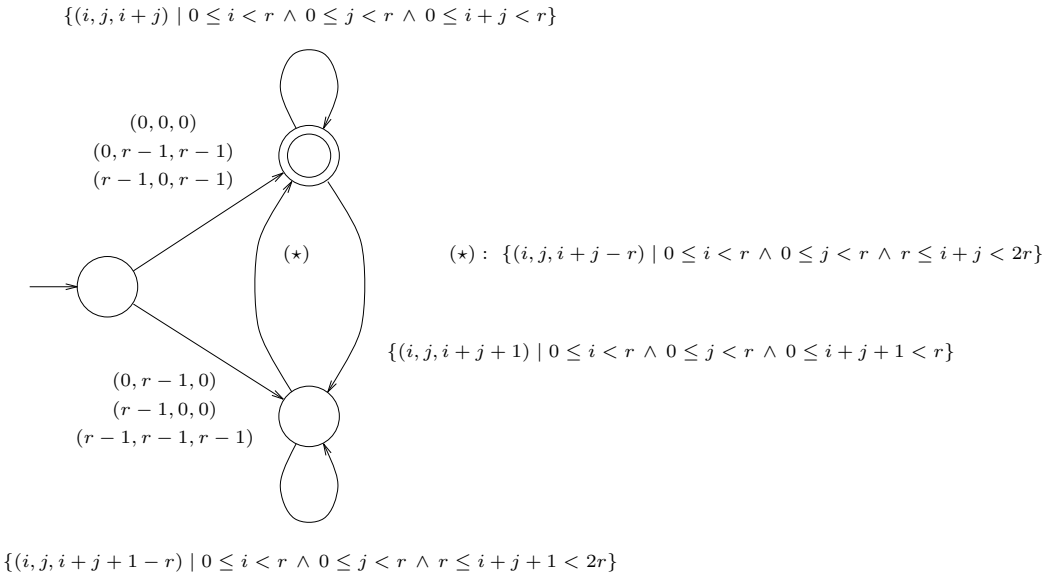

Figure 8.3: NDD representing U_{+}.

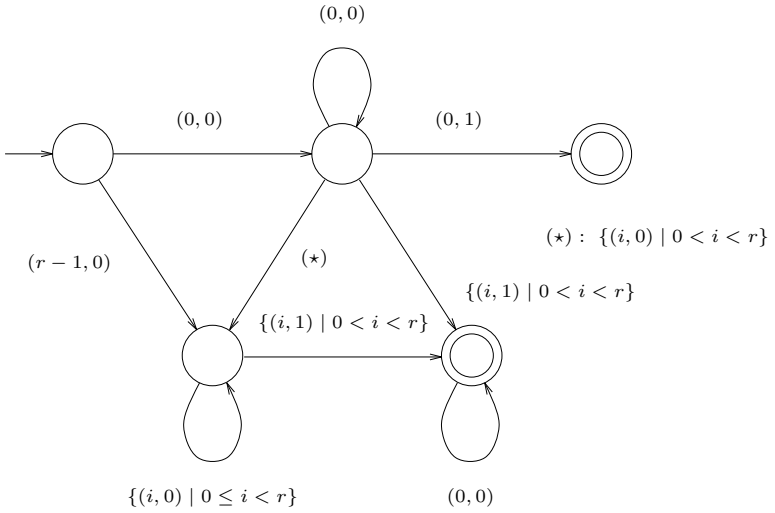

Figure 8.4: NDD representing U_{V_r}.

- If the automaton reaches the carry state after having read the triple (w'_1, w'_2, w'_3), then there exist $v'_1, v'_2, v'_3 \in \mathbf{Z}$ such that $w'_1 \in [v'_1]_r$, $w'_2 \in [v'_2]_r$, $w'_3 \in [v'_3]_r$ and $v'_3 = v'_1 + v'_2 + 1$,
- If the automaton reaches the non-carry state after having read the triple (w'_1, w'_2, w'_3), then there exist $v'_1, v'_2, v'_3 \in \mathbf{Z}$ such that $w'_1 \in [v'_1]_r$, $w'_2 \in [v'_2]_r$, $w'_3 \in [v'_3]_r$ and $v'_3 = v'_1 + v'_2$. The non-carry state is thus accepting;

- The NDD representing U_{V_r} first checks that v_2 is positive, and that w_1 has a valid sign digit. Then, it tests whether w_2 is composed of exactly one 1 digit and of remaining 0 digits. The other restrictions are that either the digit of w_1 at the same position as the 1 in w_2 is different from zero and all the subsequent digits of w_1 are equal to zero, or $v_1 = 0$ and $v_2 = 1$.

□

Lemma 8.8 *Let $r > 1$ be a basis, $n_1, n_2 \in \mathbf{N}$ be dimensions, and $U_1 \subseteq \mathbf{Z}^{n_1}$, $U_2 \subseteq \mathbf{Z}^{n_2}$ be two sets of vector values. If U_1 and U_2 are recognizable with respect to the synchronous encoding scheme, then the sets $U_1 \cup U_2$, $U_1 \cap U_2$, $U_1 \setminus U_2$ and $U_1 \times U_2$ are also recognizable if they are defined. Moreover, NDDs representing those sets can be computed from NDDs representing U_1 and U_2.*

Proof Immediate, as a consequence of Theorem 6.8. If $\mathcal{A}_1$ and $\mathcal{A}_2$ are NDDs representing respectively U_1 and U_2, then representations of $U_1 \cup U_2$, $U_1 \cap U_2$, $U_1 \setminus U_2$ and $U_1 \times U_2$ are respectively given by UNION$(\mathcal{A}_1, \mathcal{A}_2)$, INTERSECTION$(\mathcal{A}_1, \mathcal{A}_2)$, DIFFERENCE$(\mathcal{A}_1, \mathcal{A}_2)$, and PRODUCT$(\mathcal{A}_1, \mathcal{A}_2)$. □

Lemma 8.9 *Let $r > 1$ be a basis, $n \in \mathbf{N}_0$ be a dimension, $U \subseteq \mathbf{Z}^n$ be a set of vector values, and $i \in \mathbf{N}$ be such that $1 \leq i \leq n$. If U is recognizable with respect to the synchronous encoding scheme, then the set*

$$\exists_i(U) = \{(v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n) \in \mathbf{Z}^{n-1} \mid (\exists v_i \in \mathbf{Z})((v_1, \ldots, v_n) \in U)\},$$

called the projection *of U with respect to the vector component x_i, is also recognizable. Moreover, an NDD representing $\exists_i(U)$ can be computed from an NDD representing U.*

Proof Let $\mathcal{A}$ be an NDD representing U. The vector value $(v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n)$ belongs to $\exists_i(U)$ if and only if there exist $v_i \in \mathbf{Z}$ and a path π of $\mathcal{A}$ such that π is labeled by an encoding in basis r of $(v_1, \ldots, v_n)$. Let f be the homomorphism that maps every symbol in $\{0, 1, \ldots, r-1\}^n$ onto the symbol of $\{0, 1, \ldots, r-1\}^{n-1}$ obtained by deleting its i-th component. Formally, we have

$$\begin{array}{rcl} f & : & (\{0,1,\ldots,r-1\}^n)^* \to (\{0,1,\ldots,r-1\}^{n-1})^* \\ & & : \left\{ \begin{array}{rcl} \varepsilon & \mapsto & \varepsilon, \\ (a_1, \ldots, a_n) & \mapsto & (a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n), \\ w_1 \cdot w_2 & \mapsto & f(w_1) \cdot f(w_2). \end{array} \right. \end{array}$$

The automaton $\mathcal{A}'$ = APPLY-HOMOMORPHISM$(\mathcal{A}, f)$ is such that the set of vector values encoded by its accepted language is exactly $\exists_i(U)$. This does not imply that $\mathcal{A}'$ is an NDD representing $\exists_i(U)$, since for every element $\vec{v}$ of $\exists_i(U)$, $\mathcal{A}'$ may accept some encodings of $\vec{v}$ but not all of them.

An NDD representing $\exists_i(U)$ can be obtained by building an automaton $\mathcal{A}''$ that accepts all the encodings of the vector values such that at least one of their encodings is accepted by $\mathcal{A}'$. This can be done as follows. If two encodings $w_1, w_2 \in (\{0, 1, \ldots, r-1\}^n)^*$ of the same vector value $\vec{v}$ differ, then there exist $a \in \{0, r-1\}^n$ and $k \in \mathbf{N}_0$ such that either $w_1 = a^k \cdot w_2$ and $a = w_2[1]$ or $w_2 = a^k \cdot w_1$ and $a = w_1[1]$. Intuitively, this means that the two words only differ by the number of leading copies of the sign digits of the vector components. The automaton $\mathcal{A}''$ can thus be built in such a way that each of its accepted words corresponds to a word accepted by $\mathcal{A}'$ to which the leading symbol has been prefixed an arbitrary number of times, or from which this symbol has been removed an arbitrary number of times if it was repeated more than once. An algorithm formalizing the overall construction of an NDD representing $\exists_i(U)$ is given in Figure 8.5. □

Lemma 8.10 *Let $r > 1$ be a basis, $n \in \mathbf{N}$ be a dimension, $U \subseteq \mathbf{Z}^n$ be a set of vector values, and $(i_1, i_2, \ldots, i_n) \in \mathbf{N}^n$ be a permutation of $\{1, 2, \ldots, n\}$. If U is recognizable with respect to the synchronous encoding scheme, then the set*

$$\rho_{(i_1,\ldots,i_n)}(U) = \{(v_{i_1}, v_{i_2}, \ldots, v_{i_n}) \in \mathbf{Z}^n \mid (v_1, v_2, \ldots, v_n) \in U\},$$

function PROJECT-NDD(basis r, dimension n, NDD $\mathcal{A}$, integer i) : NDD

1: **var** f : function;
2: $(\Sigma, S, \Delta, I, F)$: automaton;
3: a : symbol;
4: s', s'' : states;
5: S' : **set of** states;
6: **begin**
7: $\mathcal{A} :=$ NORMALIZE($\mathcal{A}$);
8: $f := (\{0, 1, \ldots, r-1\}^n)^* \to (\{0, 1, \ldots, r-1\}^{n-1})^*$

$$: \begin{cases} \varepsilon & \mapsto & \varepsilon, \\ (a_1, \ldots, a_n) & \mapsto & (a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n), \\ w_1 \cdot w_2 & \mapsto & f(w_1) \cdot f(w_2); \end{cases}$$

9: $(\Sigma, S, \Delta, I, F) :=$ APPLY-HOMOMORPHISM($\mathcal{A}, f$);
10: **let** $s' \notin S$;
11: $S := S \cup \{s'\}$;
12: **for each** $a \in \{0, r-1\}^{n-1}$ **do**
13: **begin**
14: **let** $s'' \notin S$;
15: $S := S \cup \{s''\}$;
16: $S' := \{s \in S \mid (\exists s_0 \in I, k \in \mathbf{N}_0)((s_0, a^k, s) \in \Delta^*)\}$;
17: $\Delta := \Delta \cup \{(s', a, s'')\} \cup \{(s'', a, s'')\} \cup \{(s'', \varepsilon, s) \mid s \in S'\}$
18: **end**;
19: $I := \{s'\}$;
20: **return** $(\Sigma, S, \Delta, I, F)$
21: **end**.

Figure 8.5: Projection of an NDD with respect to a vector component.

function REORDER-NDD(basis r, dimension n, NDD $\mathcal{A}$, permutation $(i_1, i_2, \ldots, i_n)$) : NDD

1: **var** $(\Sigma, S, \Delta, I, F)$: NDD;
2: **begin**
3: $(\Sigma, S, \Delta, I, F)$:= NORMALIZE($\mathcal{A}$);
4: $\Delta := \{(s, (a_{i_1}, a_{i_2}, \ldots, a_{i_n}), s') \mid (s, (a_1, a_2, \ldots, a_n), s') \in \Delta\} \cup \{(s, \varepsilon, s') \in \Delta\}$;
5: **return** $(\Sigma, S, \Delta, I, F)$
6: **end**.

Figure 8.6: Reordering of an NDD.

called the reordering *of U with respect to the indices $i_1, i_2, \ldots, i_n$, is also recognizable. Moreover, an NDD representing $\rho_{(i_1,\ldots,i_n)}(U)$ can be computed from an NDD representing the set U.*

Proof Immediate, since an NDD $\mathcal{A}'$ representing $\rho_{(i_1,\ldots,i_n)}(U)$ can be computed from an NDD $\mathcal{A}'$ representing U by replacing every symbol $(a_1, a_2, \ldots, a_n) \in \{0, 1, \ldots, r-1\}^n$ labeling the transitions of $\mathcal{A}$ by the symbol $(a_{i_1}, a_{i_2}, \ldots, a_{i_n})$. An algorithm formalizing the construction of $\mathcal{A}'$ is given in Figure 8.6. □

We are now ready to state the sufficient condition on sets of vector values that are recognizable with respect to the synchronous encoding scheme.

Theorem 8.11 *Let $r > 1$ be a basis, $n \in \mathbf{N}$ be a dimension, and $U \subseteq \mathbf{Z}^n$ be a set of vector values. If U is definable in the first-order theory $\langle \mathbf{Z}, \leq, +, V_r \rangle$, then U is recognizable with respect to the synchronous encoding scheme $E_{S(r)}$. Moreover, an NDD representing U can be computed from a formula of $\langle \mathbf{Z}, \leq, +, V_r \rangle$ defining U.*

Proof If U is definable in the theory $\langle \mathbf{Z}, \leq, +, V_r \rangle$, then there exists in this theory a formula $\varphi : \mathbf{Z}^n \to \{\mathbf{T}, \mathbf{F}\}$ such that

$$U = \{(v_1, v_2, \ldots, v_n) \in \mathbf{Z}^n \mid \varphi(v_1, v_2, \ldots, v_n)\}.$$

We then say that φ *denotes* U. The variables $v_1, v_2, \ldots, v_n$ are the *free variables* of φ. The principle of the proof is to translate the formula φ into an NDD $\mathcal{A}$ representing the set U denoted by φ. The construction proceeds by induction on φ.

The atomic formulas of $\langle \mathbf{Z}, \leq, +, V_r \rangle$ are the equality $x = y$, the inequality $x \leq y$ and the equations $z = x + y$ and $y = V_r(x)$, where x, y and z are variables. By introducing as many additional variables as necessary, we can rewrite φ as a formula in which each variable does not appear more than once in a given atomic formula (for instance, the formula $x = V_r(x)$ can be rewritten as $(\exists x' \in \mathbf{Z})(x' = V_r(x) \wedge x = x')$,

where x' is an auxiliary variable). The atomic formulas can be translated into NDDs representing the sets $U_=$, $U_\leq$, U_+ and U_{V_r} that they denote, thanks to Lemma 8.7.

The inductive formulas of $\langle \mathbf{Z}, \leq, +, V_r \rangle$ are the conjunction $\varphi_1 \wedge \varphi_2$, the disjunction $\varphi_1 \vee \varphi_2$, the negation $\neg\varphi_1$, the existential quantification $(\exists x)(\varphi_1)$ and the universal quantification $(\forall x)(\varphi_1)$, where φ_1 and φ_2 are formulas and x is a variable. Since $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$ and $(\forall x)(\varphi_1) \equiv \neg(\exists x)(\neg\varphi_1)$, it is sufficient to consider the conjunction, negation, and existential quantification of formulas.

If the free variables of φ_1 and φ_2 are identical and appear in the same order, then the formula $\varphi_1 \wedge \varphi_2$ denotes the set $U_1 \cap U_2$, where U_1 and U_2 are respectively denoted by φ_1 and φ_2. An NDD representing $U_1 \cap U_2$ can be computed thanks to Lemma 8.8. If the free variables of φ_1 and φ_2 are not identical, then the sets of free variables of each formula can be extended so as to include the free variables of the other one. Formally, adding a free variable to the formula $\varphi_1 : \mathbf{Z}^{n_1} \to \{\mathbf{T}, \mathbf{F}\}$, where $n_1 \geq 0$, yields the formula

$$\varphi_1' : \mathbf{Z}^{n_1+1} \to \{\mathbf{T}, \mathbf{F}\} : (v_1, \ldots, v_{n_1+1}) \mapsto \varphi(v_1, \ldots, v_{n_1}).$$

If $U_1 \subseteq \mathbf{Z}^{n_1}$ is the set denoted by φ_1, then the set $U_1 \subseteq \mathbf{Z}^{n_1+1}$ denoted by φ_1' is such that $U_1' = U_1 \times \mathbf{Z}$. Since $\mathbf{Z} = \exists_1(U_=)$, an NDD $\mathcal{A}_1'$ representing U_1' can be computed from an NDD $\mathcal{A}_1$ representing U_1 as a consequence of Lemmas 8.7, 8.8 and 8.9. Finally, if the free variables of φ_1 and φ_2 are identical but appear in a different order, then thanks to Lemma 8.10, the free variables of one of the formulas can be reordered so as to match those of the other one. This is done by applying the reordering operator ρ to the NDD representing the set denoted by the formula.

The formula $\neg\varphi_1$ denotes the set $U_1' = \mathbf{Z}^{n_1} \setminus U_1$, where $U_1 \subseteq \mathbf{Z}^{n_1}$ is the set denoted by φ_1. Since $\mathbf{Z}^{n_1} = \mathbf{Z} \times \mathbf{Z} \times \cdots \times \mathbf{Z}$, an NDD representing U_1' can be obtained from an NDD representing U_1 thanks to Lemmas 8.7, 8.8 and 8.9. The formula $(\exists x)(\varphi_1)$ denotes the set $U_1'' = \exists_i(U_1)$, where $i \in \mathbf{N}$ is such that x is the i-th free variable of φ_1. An NDD representing U_1'' can be obtained from an NDD representing U_1 thanks to Lemma 8.10. □

Theorems 8.6 and 8.11 can be combined into a necessary and sufficient condition which characterizes precisely the sets of vector values that can be represented as NDDs.

Theorem 8.12 *Let $n \in \mathbf{N}$ be a dimension, $r > 1$ be a basis, and $U \subseteq \mathbf{Z}^n$ be a set of vector values. The set U is recognizable with respect to the synchronous encoding scheme $E_{S(r)}$ if and only if U is definable in the first-order theory $\langle \mathbf{Z}, \leq, +, V_r \rangle$. Moreover, an NDD representing U can be computed from a formula of $\langle \mathbf{Z}, \leq, +, V_r \rangle$ defining U.*

Proof The result is a direct consequence of Theorems 8.6 and 8.11. □

In the sequel, we will denote by NDD(U) any NDD that represents the set of vector values $U \subseteq \mathbf{Z}^n$. Reciprocally, the set of vector values represented by the NDD $\mathcal{A}$ will be denoted SET($\mathcal{A}$). Since the emptiness of sets represented as NDDs is decidable (as a consequence of Theorem 6.9), Theorem 8.12 has the following corollary.

Corollary 8.13 *The first-order theory $\langle \mathbf{Z}, \leq, +, V_r \rangle$ is decidable.*

Proof Immediate, since for every closed formula φ of $\langle \mathbf{Z}, \leq, +, V_r \rangle$, we have $\varphi = \mathbf{T}$ if and only if[2] EMPTY?(NDD($\{\vec{x} \in \mathbf{Z}^n \mid \varphi(\vec{x})\}$)) $= \mathbf{F}$. □

8.1.5 Sets that are Representable in Any Basis

The results presented in Section 8.1.4 give an exact characterization of the sets of vector values that can be represented as NDDs in a given basis $r > 1$. The question addressed here is to characterize the sets that are representable in any basis $r > 1$, with respect to the synchronous encoding scheme.

The problem has been solved by Cobham [Cob69] for subsets of $\mathbf{Z}$, and then generalized to subsets of $\mathbf{Z}^n$ for any $n > 0$ by Semenov [Sem77]. Their characterization, usually referred to as the theorem of Cobham-Semenov, can be stated as follows[3].

Theorem 8.14 *Let $n > 0$ be a dimension, $U \subseteq \mathbf{Z}^n$ be a set of vector values, and $r_1, r_2 > 1$ be two bases. If r_1 and r_2 are such that there do not exist $i, j \in \mathbf{N}_0$ for which $r_1^i = r_2^j$ (r_1 and r_2 are then said to be* multiplicatively independent*), and if U is both definable in $\langle \mathbf{Z}, \leq, +, V_{r_1} \rangle$ and in $\langle \mathbf{Z}, \leq, +, V_{r_2} \rangle$, then U is definable in $\langle \mathbf{Z}, \leq, + \rangle$.*

Proof The proof is quite technical and its presentation is well beyond the scope of this thesis. A comprehensible proof can be found in [MV93] and [MV96]. An elegant proof due to Muchnik [Muc91] and based on an original criterion is presented in [BHMV94]. □

The theorem of Cobham-Semenov has an important corollary that allows to characterize exactly the sets that are representable as NDDs in any basis.

Corollary 8.15 *Let $n > 0$ be a dimension, and $U \subseteq \mathbf{Z}^n$ be a set of vector values. The set U is recognizable in every basis $r > 1$ with respect to the synchronous encoding scheme $E_{S(r)}$ if and only if U is definable in the first-order theory $\langle \mathbf{Z}, \leq, + \rangle$. Moreover, an NDD representing U can be computed from a formula of $\langle \mathbf{Z}, \leq, + \rangle$ defining U.*

[2] The dimension of NDD($\{\vec{x} \in \mathbf{Z}^n \mid \varphi(\vec{x})\}$) is equal to 0, which means that all the transitions of this NDD are labeled by the empty word. This NDD is actually an *inputless* automaton.

[3] The results of Cobham and Semenov actually apply to subsets of $\mathbf{N}^n$. The generalization to subsets of $\mathbf{Z}^n$ is immediate.

Proof One direction is a direct consequence of Theorem 8.14. The other follows from Theorem 8.12, since $\langle \mathbf{Z}, \leq, + \rangle$ is a subset of $\langle \mathbf{Z}, \leq, +, V_r \rangle$ for any $r > 1$. □

The theory $\langle \mathbf{Z}, \leq, + \rangle$ has been studied by Presburger [Pre29] and is usually referred to as *Presburger arithmetic*. It is known [Opp78, FR79] that deciding Presburger arithmetic is 2EXPSPACE-complete.

An advantage of considering the sets of vector values that are definable in Presburger arithmetic rather than the sets definable in $\langle \mathbf{Z}, \leq, +, V_r \rangle$ for some basis $r > 1$ is that lots of techniques have been developed for dealing with Presburger arithmetic, and that efficient implementations of these techniques have been made available. An example of such an implementation is the *Omega Test* [Pug92a] which allows to manipulate formulas of Presburger arithmetic with a surprising efficiency. Another result of interest, due to Boudet and Comon [BC96], shows that the minimal and deterministic NDD representing the set of vector values that satisfies a system of linear equations and inequations is very compact and can be computed efficiently.

On the other hand, there are applications for which using the theory $\langle \mathbf{Z}, \leq, +, V_r \rangle$ for some basis $r > 1$ is nonetheless more advantageous than using Presburger arithmetic. For instance, the model of a hardware circuit performing some arithmetic operation on unbounded binary numbers might very well have a control location in which the set of reachable values is the set of the powers of 2. It can be shown that this set cannot be defined in Presburger arithmetic, even though it is denoted by the formula $\varphi(x) \equiv V_2(x) = x$ which belongs to $\langle \mathbf{Z}, \leq, +, V_2 \rangle$.

Since both theories have advantages, the approach followed in the rest of this chapter is to stay as general as possible. Each result dealing with the possibility of representing a set of vector values as an NDD will thus be expressed twice: once with respect to the theory $\langle \mathbf{Z}, \leq, +, V_r \rangle$ for any $r > 1$, and once with respect to Presburger arithmetic. Intuitively, the former case consists of choosing the numeration basis used by the NDD, and the latter one consists of requiring that the result has to hold in any basis. We will make use of the following definitions.

Definition 8.16 *Let $r > 1$ be a basis, $n \in \mathbf{N}$ be a dimension, and $U \subseteq \mathbf{Z}^n$ be a set of vector values. The set U is* r-recognizable *if it is recognizable with respect to the synchronous encoding $E_{S(r)}$.*

Definition 8.17 *Let $n \in \mathbf{N}$ be a dimension and $U \subseteq \mathbf{Z}^n$ be a set of vector values. The set U is* Presburger-definable *if for every basis $r > 1$, it is recognizable with respect to the synchronous encoding $E_{S(r)}$.*

In the rest of this chapter, we will only consider bases $r > 1$ for which there does not exist $j \in \mathbf{N}$, with $j \geq 2$, such that $r^{(1/j)} \in \mathbf{N}$. This can be done without loss of generality thanks to the following result.

Theorem 8.18 *Let $n \in \mathbf{N}$ be a dimension, $r > 1$ be a basis and $U \subseteq \mathbf{Z}^n$ be a set of vector values. For every $k \in \mathbf{N}_0$, U is r-recognizable if and only if U is r^k-recognizable.*

Proof An automaton $\mathcal{A}_k$ in normal form representing the set U with respect to $E_{S(r^k)}$ can easily be turned into an automaton $\mathcal{A}_1$ representing U with respect to $E_{S(r)}$ by transforming each of its transitions $(s, (a_1, a_2, \ldots, a_n), s')$ into $(s, (a'_{1,1}, a'_{1,2}, \ldots, a'_{1,n}) \cdot (a'_{2,1}, a'_{2,2}, \ldots, a'_{2,n}) \cdots (a'_{k,1}, a'_{k,2}, \ldots, a'_{k,n}), s')$, where for each $i \in \{1, 2, \ldots, n\}$, $a'_{1,i}$, $a'_{2,i}, \ldots, a'_{k,i} \in \{0, 1, \ldots, r-1\}$ and $a_i = \sum_{0 \leq j < k} r^j a'_{(k-j),i}$. The resulting automaton $\mathcal{A}'$ does not necessarily accept all the encodings of the elements of U, and the construction proposed in the proof of Lemma 8.9 can be used to obtain $\mathcal{A}_1$ from $\mathcal{A}'$.

The reciprocal transformation is also possible. Given an automaton $\mathcal{A}_1$, one first computes an automaton $\mathcal{A}$ in strong normal form that accepts the language $L(\mathcal{A}_1) \cap \bigcup_{i \in \mathbf{N}} \{0, 1, \ldots, r-1\}^{ik}$. Then, one constructs $\mathcal{A}_k$ as follows:

- The set of states of $\mathcal{A}_k$ contains the states of $\mathcal{A}$ that are reachable by reading a word w whose length is an integer multiple of k;
- There is a transition $(s, (a_1, a_2, \ldots, a_n), s')$ between the states s and s' of $\mathcal{A}_k$ if and only if there exist $a'_{1,1}, a'_{1,2}, \ldots, a'_{1,n}, a'_{2,1}, a'_{2,2}, \ldots, a'_{2,n}, a'_{k,1}, a'_{k,2}, \ldots, a'_{k,n} \in \{0, 1, \ldots, r-1\}$ such that $\mathcal{A}$ admits a path from s to s' labeled by $(a'_{1,1}, a'_{1,2}, \ldots, a'_{1,n}) \cdot (a'_{2,1}, a'_{2,2}, \ldots, a'_{2,n}) \cdots (a'_{k,1}, a'_{k,2}, \ldots, a'_{k,n})$, and such that for each $i \in \{1, 2, \ldots, n\}$, $a_i = \sum_{0 \leq j < k} r^j a'_{(k-j),i}$;
- The sets of initial and of accepting states of $\mathcal{A}_k$ are identical to those of $\mathcal{A}$.

□

8.1.6 Other Encoding Schemes

Of course, the synchronous encoding scheme is not the only scheme that can be used for encoding vector values. Following [BHMV94], we qualify as "good" an encoding scheme that allows to represent every set that is definable in Presburger arithmetic. In this section, we define two new "good" encoding schemes for vector values, and study how they relate to the synchronous encoding scheme.

The first encoding scheme that we introduce simply consists of reading the digits of the vector components from the least significant one to the most significant one rather than the other way around. We have the following definition.

Definition 8.19 *Let $n \geq 0$ be a dimension and $r > 1$ be a basis. The* reverse synchronous encoding scheme *$E_{R(r)}$ is the relation that associates to a vector of $\mathbf{Z}^n$*

the tuple of the same-length encodings in basis r of the components of this vector written backwards. Formally, we have

$$E_{R(r)} \subseteq M \times V_{R(r)} = \{((v_1, \ldots, v_n), (w_1, \ldots, w_n)) \mid |w_1| = |w_2| = \cdots = |w_n| \\ \wedge\, w_1^R \in [v_1]_r \wedge w_2^R \in [v_2]_r \wedge \cdots \wedge w_n^R \in [v_n]_r\},$$

where $M = \mathbf{Z}^n$ *and* $V_{R(r)} = \bigcup_{k \in \mathbf{N}} (\{0, 1, \ldots, r-1\}^k \cdot \{0, r-1\})^n$, *and for every* $i \in \{1, 2, \ldots, n\}$, w_i^R *denotes the word* w_i *written from right to left.*

The sets of vector values that are recognizable with respect to the reverse synchronous encoding scheme are exactly the ones that are recognizable with respect to the synchronous encoding scheme. Indeed, an automaton $\mathcal{A}_S$ representing the set $U \subseteq \mathbf{Z}^n$ $(n \geq 0)$ with respect to $E_{S(r)}$ $(r > 1)$ can easily be turned into an automaton $\mathcal{A}_R$ representing U with respect to $E_{R(r)}$ by transforming each of its transitions (s, w, s') into (s', w^R, s), where w^R denotes the word w written from right to left, and by exchanging the sets of initial and of accepting states. The reciprocal transformation can be performed identically.

It is worth mentioning that this result does not imply that the synchronous and reverse synchronous encoding schemes are equivalent in practice. Indeed, if one chooses to represent sets of vector values as minimal deterministic finite-state automata, then there exist sets whose representations will be more concise with one scheme than with its counterpart. For instance, representing in basis 2 the set of all the integers whose binary expansion has the coefficient of 2^k $(k \geq 0)$ equal to 1 yields a minimal deterministic automaton of size $O(2^k)$ if the synchronous encoding scheme is used, but only of size $O(k)$ with the reverse synchronous encoding scheme. The encoding scheme used in a particular application must therefore be carefully selected with respect to the sets of vector values that will be potentially represented.

The other encoding scheme studied here consists of reading the digits of the vector components successively, rather than simultaneously, in increasing order of position. This encoding scheme is formally defined as follows.

Definition 8.20 *Let* $n \in \mathbf{N}$ *be a dimension and* $r > 1$ *be a basis. The* synchronous interleaved encoding scheme $E_{I(r)}$ *is the relation*

$$E_{I(r)} \subseteq M \times V_{I(r)} = \{((v_1, \ldots, v_n), w) \mid (\exists l \in \mathbf{N})(|w| = l.n \\ \wedge\, w[1] \cdot w[1+n] \cdots w[1+(l-1)n] \in [v_1]_r \\ \wedge\, w[2] \cdot w[2+n] \cdots w[2+(l-1)n] \in [v_2]_r \\ \vdots \\ \wedge\, w[n] \cdot w[n+n] \cdots w[n+(l-1)n] \in [v_n]_r)\},$$

where $M = \mathbf{Z}^n$ *and* $V_{I(r)} = \{0, r-1\}^n \cdot \bigcup_{k \in \mathbf{N}} \{0, 1, \ldots, r-1\}^{kn}$.

The sets of vector values that are recognizable with respect to the synchronous interleaved encoding scheme are exactly the ones that are recognizable with respect to the synchronous encoding scheme. Indeed, an automaton $\mathcal{A}_S$ in normal form representing the set $U \subseteq \mathbf{Z}^n$ $(n \geq 0)$ with respect to $E_{S(r)}$ $(r > 1)$ can easily be turned into an automaton $\mathcal{A}_I$ representing U with respect to $E_{I(r)}$ by transforming each of its transitions $(s, (a_1, a_2, \ldots, a_n), s')$ into $(s, a_1 \cdot a_2 \cdots a_n, s')$. The reciprocal transformation is also possible. Given an automaton $\mathcal{A}_I$ in strong normal form, the corresponding $\mathcal{A}_S$ can be constructed as follows:

- The set of states of $\mathcal{A}_S$ contains the states of $\mathcal{A}_I$ that are reachable by reading a word w whose length is a multiple of the dimension n;
- There is a transition $(s, (a_1, a_2, \ldots, a_n), s')$ between the states s and s' of $\mathcal{A}_S$ if and only if $\mathcal{A}_I$ admits a path from s to s' labeled by $a_1 \cdot a_2 \cdots a_n$;
- The sets of initial and of accepting states of $\mathcal{A}_S$ are identical to those of $\mathcal{A}_I$.

Once again here, even though the synchronous and the synchronous interleaved encoding schemes have the same expressiveness, they are not equivalent in practice from the point of view of conciseness. For instance, representing the set $\mathbf{Z}^k$, with $k \geq 0$, yields a minimal deterministic automaton of size $O(2^k)$ if the synchronous encoding scheme is used, but only of size $O(k)$ with the synchronous interleaved encoding scheme. The size of the minimal deterministic finite-state representations obtained with the synchronous interleaved encoding scheme is actually always asymptotically smaller than what can be obtained with the synchronous encoding scheme. The drawback of the synchronous interleaved encoding scheme is that the algorithms implementing set operations are slightly more complex than their synchronous counterpart.

The synchronous encoding scheme is used in the rest of this chapter for simplicity. The algorithms that will be developed can however be adapted to the reverse synchronous or the interleaved synchronous encoding schemes with little difficulty.

8.2 Linear Operations

The problem addressed here consists of computing the image of a set of vector values represented as an NDD by the linear operation labeling a transition of an ISMA. Given an NDD $\mathcal{A}$ representing the set of vector values $U \subseteq \mathbf{Z}^n$ $(n \geq 0)$ in the basis $r > 1$ and a linear operation $P\vec{x} \leq \vec{q} \rightarrow \vec{x} := A\vec{x} + \vec{b}$, where $m \in \mathbf{N}$, $P \in \mathbf{Z}^{m \times n}$, $\vec{q} \in \mathbf{Z}^m$, $A \in \mathbf{Z}^{n \times n}$ and $\vec{b} \in \mathbf{Z}^n$, the problem thus consists of computing an NDD $\mathcal{A}'$ representing the set

$$U' = \{A\vec{v} + \vec{b} \mid \vec{v} \in U \wedge P\vec{v} \leq \vec{q}\}.$$

This problem can be solved by first constructing two NDDs $\mathcal{A}_1$ and $\mathcal{A}_2$ representing respectively the sets $U_1 \subseteq \mathbf{Z}^n$ and $U_2 \subseteq \mathbf{Z}^{2n}$ such that $U_1 = \{\vec{v} \in \mathbf{Z}^n \mid P\vec{v} \leq \vec{q}\}$ and $U_2 = \{(\vec{v}, \vec{v}') \mid \vec{v}' = A\vec{v} + \vec{b}\}$. Indeed, we have

$$U' = \exists_1 \exists_2 \cdots \exists_n ((U \cap U_1) \times \mathbf{Z}^n \cap U_2),$$

which yields a way of computing $\mathcal{A}'$ from $\mathcal{A}$, $\mathcal{A}_1$ and $\mathcal{A}_2$.

It remains to show how to build $\mathcal{A}_1$ and $\mathcal{A}_2$. A simple solution consists of translating into NDDs the formulas

$$\varphi_1 : \mathbf{Z}^n \to \{\mathbf{T}, \mathbf{F}\} : \vec{v} \mapsto P\vec{v} \leq \vec{q}$$

and

$$\varphi_2 : \mathbf{Z}^{2n} \to \{\mathbf{T}, \mathbf{F}\} : (\vec{v}, \vec{v}') \mapsto \vec{v}' = A\vec{v} + \vec{b},$$

denoting respectively U_1 and U_2, by applying Theorem 8.11. The requirement stating that φ_1 and φ_2 must be expressed in the theory $\langle \mathbf{Z}, \leq, +, V_r \rangle$ can easily be fulfilled. The only difficulty is to convert a formula of the form $y = cx$, where $c \in \mathbf{Z}$ is a constant and x and y are variables, into a formula of $\langle \mathbf{Z}, \leq, +, V_r \rangle$.

This conversion can be carried out as follows. First, we assume that $c \in \mathbf{N}$. Indeed, if $c < 0$, then $y = cx$ is equivalent to $(\exists y' \in \mathbf{Z})(y + y' = 0 \wedge y' = (-c)x)$. Let $c_{p-1}, c_{p-2}, \ldots, c_1, c_0 \in \{0, 1\}$ $(p > 0)$ be the digits of the binary expansion of c, i.e., let $c_{p-1} \cdot c_{p-2} \cdots c_1 \cdot c_0 \in [c]_2$. We define $q \geq 0$ and $i_1, i_2, \ldots i_q \in \{0, 1, \ldots, p-1\}$ such that $\{i_1, i_2, \ldots, i_q\} = \{i \in \{0, 1, \ldots, p-1\} \mid c_i = 1\}$. The formula $y = cx$ is equivalent to

$$\begin{aligned}(\exists x_0, x_1, \ldots x_{p-1}, y_0, y_1, \ldots, y_q \in \mathbf{Z})(x_0 = x \wedge x_1 = x_0 + x_0 \wedge x_2 = x_1 + x_1 \wedge \\ \cdots \wedge x_{p-1} = x_{p-2} + x_{p-2} \wedge y_0 = y_0 + y_0 \wedge y_1 = y_0 + x_{i_1} \wedge y_2 = y_1 + x_{i_2} \wedge \\ \cdots \wedge y_q = y_{q-1} + x_{i_q} \wedge y = y_q),\end{aligned}$$

which belongs to $\langle \mathbf{Z}, \leq, +, V_r \rangle$. The size of the converted formula is $O(\log_2 c)$, which means that the cost of the conversion is linear in the size of the original formula. In the rest of this chapter, we will allow integer variable coefficients in formulas of $\langle \mathbf{Z}, \leq, +, V_r \rangle$, and assume that the conversion method outlined above is systematically used.

An algorithm formalizing the construction of $\mathcal{A}'$ as a function of $\mathcal{A}$, $\mathcal{A}_1$ and $\mathcal{A}_2$ is given in Figure 8.7.

Theorem 8.21 *Let $n \geq 0$ be a dimension, $r > 1$ be a basis, $\mathcal{A}$ be an NDD representing the set of vector values $U \subseteq \mathbf{Z}^n$, and θ be a linear operation over $\mathbf{Z}^n$. APPLY-LINEAR$(r, n, \mathcal{A}, \theta)$ is an NDD representing the set $\theta(U)$.*

Proof The algorithm in Figure 8.7 is a direct implementation of the computation method developed in this section. □

```
function APPLY-LINEAR(basis r, dimension n, NDD A,
                      operation (Pv ≤ q → v := Av + b)) : NDD
1:  var A1, A2, A' : NDDs;
2:      i : integer;
3:  begin
4:      A1 := NDD({v ∈ Z^n | Pv ≤ q});
5:      A2 := NDD({(v, v') ∈ Z^2n | v' = Av + b});
6:      A' := INTERSECTION(PRODUCT(INTERSECTION(A, A1),
                                   NDD(Z^n)), A2);
7:      for i := 1 to n do
8:          A' := PROJECT-NDD(r, n, A', 1);
9:      return A'
10: end.
```

Figure 8.7: Application of a linear operation to an NDD.

8.3 Creation of Cycle Meta-Transitions

As it has been shown in Section 3.4.1, the creation of cycle meta-transitions is controlled by

- A computable predicate META? defined over the set of potential sequences of operations, whose purpose is to decide whether the meta-transition corresponding to a given sequence can be created, i.e., whether the closure of the sequence can always be applied to sets of memory contents;
- An algorithm for computing the image of any representable set of memory contents by the closure of a sequence of operations satisfying META?.

It has been seen that a finite sequence of linear operations is always equivalent to a single linear operation, whose parameters can be computed. In this section, we provide algorithms for computing a suitable predicate META? over linear operations, and for applying closures of such operations to sets of vector values represented as NDDs. Each result will successively be stated in the context of sets representable in a given basis $r > 1$, and then of sets representable in any basis.

8.3.1 Overview

This section is organized as follows. First, we define some notions of algebra and combinatorics that will be extensively used, and recall some known results. Next,

we extend in an original way the notion of recognizability to sets of vectors with complex components. This generalized notion of recognizability is then used as a powerful tool for establishing necessary and sufficient conditions over guardless linear operations whose closure preserves the recognizability of sets. We will give the most general solution to this problem, in the sense that it will always be possible to compute the closure θ^* of a guardless linear operation θ provided that the image by θ^* of any recognizable set of vector values is recognizable. Computing the truth value of META? for a particular linear operation θ will thus amount to deciding whether the image by θ^* of any recognizable set of vector values is recognizable. In the next part, we will present algorithms implementing with NDDs the decision procedures expressed by the necessary and sufficient conditions. Next we will give a simple extension of the results obtained so far that allows to apply those results to linear operations with guards. Finally, we will conclude the section with some proofs that are omitted from the main text for clarity.

8.3.2 Algebra and Combinatorics Basics

The sets of rational numbers and of complex numbers are respectively denoted by $\mathbf{Q}$ and by $\mathbf{C}$. For every $n \in \mathbf{N}_0$, I_n denotes the identity matrix $I_n = \mathrm{diag}(1, 1, \ldots, 1)$ of dimension n. The successive columns of I_n are denoted $\vec{e}_1, \vec{e}_2, \ldots, \vec{e}_n$. Let $A \in \mathbf{C}^{n\times n}$ be a complex matrix. If $S \subseteq \mathbf{C}^n$ is a set of vector values, then AS is a shorthand for $\{A\vec{x} \mid \vec{x} \in S\}$. Similarly, if $\vec{v} \in \mathbf{C}^n$, then $S + \vec{v}$ denotes the set $\{\vec{x} + \vec{v} \mid \vec{x} \in S\}$. The sets of rows and of columns of A are respectively denoted $\mathrm{row}(A)$ and $\mathrm{col}(A)$. The maximum number of linearly independent rows or columns of A is the *rank* of A. Any $\lambda \in \mathbf{C}$ and $\vec{x} \in (\mathbf{C}^n \setminus \{\vec{0}\})$ such that $A\vec{x} = \lambda\vec{x}$ are respectively called an *eigenvalue* and an *eigenvector* of A. The eigenvalues of A are the roots of the *characteristic polynomial* of A, defined as $\Pi(\lambda) = \det(A - \lambda I_n)$. They are also the roots of the *minimal polynomial* of A, which is defined as the polynomial $\Pi'(\lambda)$ of lowest degree such that $\Pi'(A) = (0)$. If $\lambda_1, \lambda_2, \ldots, \lambda_m$ are the eigenvalues of A, then $\lambda_1^p, \lambda_2^p, \ldots, \lambda_m^p$ are the eigenvalues of A^p for any $p \in \mathbf{N}_0$. For every $n \in \mathbf{N}_0$ and $\lambda \in \mathbf{C}$, the *Jordan block* of dimension n associated to λ is the matrix

$$J_{n,\lambda} = \begin{bmatrix} \lambda & 1 & & \\ & \lambda & 1 & \\ & & \ddots & 1 \\ & & & \lambda \end{bmatrix}.$$

A matrix $A \in \mathbf{C}^{n\times n}$ only composed of Jordan blocks on its main diagonal, in other words such that $A = \mathrm{diag}(J_{n_1,\lambda_1}, J_{n_2,\lambda_2}, \ldots)$, is said to be in *Jordan form*. For every $A \in \mathbf{C}^{n\times n}$, there exists a nonsingular matrix $U \in \mathbf{C}^{n\times n}$ such that $A = UA_JU^{-1}$, with A_J being in Jordan form (U is said to *transform* A into its Jordan form A_J). The Jordan form A_J of A is unique up to the reordering of its diagonal blocks. For

each diagonal block J_{n_i,λ_i} composing A_J, the corresponding λ_i is an eigenvalue of A. Reciprocally, for every eigenvalue λ_i of A, there exists a (possibly non unique) Jordan block J_{n_i,λ_i} that belongs to the set of diagonal blocks of A_J. If the components of A and its eigenvalues belong to $\mathbf{Q}$, then there exists $U \in \mathbf{Q}^{n\times n}$ transforming A into A_J. If the Jordan form of A is diagonal (in other words, if all its Jordan blocks are of size 1), then A is said to be *diagonalizable*.

Let p, $q \in \mathbf{N}$ with $p \leq q$. The *binomial coefficient* $C_q^p \in \mathbf{N}$ is defined as

$$C_q^p = \frac{q!}{(q-p)!\,p!}.$$

Binomial coefficients are related to Jordan blocks in the following way. If $\lambda \in \mathbf{C}$ and n, $m \in \mathbf{N}$ with $0 < n \leq m$, then the m-th power of the Jordan block $J_{n,\lambda}$ is such that

$$J_{n,\lambda}^m = \begin{bmatrix} \lambda^m C_m^0 & \lambda^{m-1} C_m^1 & \lambda^{m-2} C_m^2 & \dots & \lambda^{m-n+1} C_m^{n-1} \\ & \lambda^m C_m^0 & \lambda^{m-1} C_m^1 & \dots & \lambda^{m-n+2} C_m^{n-2} \\ & & \lambda^m C_m^0 & \dots & \lambda^{m-n+3} C_m^{n-3} \\ & & & \ddots & \vdots \\ & & & & \lambda^m C_m^0 \end{bmatrix}.$$

We now define some notions related to cyclotomic fields. It is known that every polynomial with integer coefficients can be factorized into a product of *indivisible* polynomials with integer coefficients. This factorization is unique up to multiplicative constants. For every $n \in \mathbf{N}_0$, the indivisible factors of the polynomial $x^n - 1$ are called *cyclotomic polynomials*. There is a cyclotomic polynomial Φ_m associated to every integer $m \in \mathbf{N}_0$, defined as

$$\Phi_m(x) = \prod_{[k,m]} (x - e^{\frac{2ik\pi}{m}}),$$

where $[k, m]$ stands for $1 \leq k < m \wedge \gcd(k, m) = 1$. Actually, we have

$$x^n - 1 = \prod_{k|n} \Phi_k(x),$$

where $k|n$ means "k divides n". For every $m \in \mathbf{N}_0$, the degree of $\Phi_m(x)$ is equal to $\phi(m)$, where ϕ is the *Euler function*. This function is defined as

$$\phi : \mathbf{N}_0 \to \mathbf{N}_0 : x \mapsto x \left(1 - \frac{1}{p_1}\right)\left(1 - \frac{1}{p_2}\right)\cdots\left(1 - \frac{1}{p_q}\right),$$

where $p_1, p_2, \dots, p_q$ are the (distinct) prime factors of x. Intuitively, $\phi(m)$ is the number of integers in $\{1, 2, \dots, m\}$ that are relatively prime to m.

8.3.3 Recognizability of Sets of Complex Vector Values

Let $n \in \mathbf{N}$ be a dimension. In this section, we generalize the notion of recognizable set of vector values to subsets of $\mathbf{C}^n$. The reason why sets of complex vector values are considered is that Jordan forms of matrices will be heavily used, and that transforming a matrix into its Jordan form is generally not possible within $\mathbf{R}$. Intuitively, the idea behind the generalization of recognizability is the following. Let $S \subseteq \mathbf{Z}^n$ be a set of vector values and let $\theta = (\vec{x} := A\vec{x} + \vec{b})$, where $A \in \mathbf{Z}^{n \times n}$ and $\vec{b} \in \mathbf{Z}^n$, be a guardless linear operation. If the transformation matrix A is nonsingular, then the set S is recognizable (either with respect to a given basis $r > 1$ or to all of them) if and only if the set $\theta(S)$ is recognizable. This shows that the recognizable nature of a set of integer vector values is not influenced by nonsingular linear transformations. It is therefore natural to define a set of complex vector values as "recognizable" if it can be expressed as the image of a recognizable set of integer vector values by some linear transformation. Formally, we have the following definition.

Definition 8.22 *Let $n, r \in \mathbf{N}_0$ with $r > 1$. A set of complex vector values $S \subseteq \mathbf{C}^n$ is* r-definable *if and only if there exist $m \in N_0$, $S' \subseteq \mathbf{Z}^m$ and $U \in \mathbf{C}^{n \times m}$ such that S' is r-recognizable and $S = US'$.*

The following result shows that the notion of r-definability is indeed an generalization of r-recognizability, i.e., that the two notions coincide for sets of integer vector values.

Theorem 8.23 *Let $n, r \in \mathbf{N}_0$ with $r > 1$. A set $S \subseteq \mathbf{Z}^n$ is r-definable if and only if it is r-recognizable.*

Proof The proof is given in Section 8.3.8. □

The next step is to show how to obtain definable sets of complex vector values. The following theorem establishes the definability of some elementary sets, and presents operations that can be used for combining definable sets.

Theorem 8.24 *Let $r \in \mathbf{N}$ with $r > 1$, $n_1, n_2 \in \mathbf{N}_0$, $S_1 \subseteq \mathbf{C}^{n_1}$, $S_2 \subseteq \mathbf{C}^{n_2}$ such that S_1 and S_2 are r-definable, $\vec{v} \in \mathbf{C}^{n_1}$, $p, q \in \mathbf{N}_0$, and $T \in \mathbf{C}^{p \times n_1}$. The following sets are r-definable:*

- *Any finite subset of $\mathbf{C}^{n_1}$,*
- *$S_1 + \vec{v}$,*
- *TS_1,*
- *$S_1 \cup S_2$, provided that $n_1 = n_2$,*
- *$S_1 \cap S_2$, provided that $n_1 = n_2$,*

- $S_1 \times S_2$,
- $\left\{ \begin{bmatrix} \vec{x} \\ \Re(\vec{x}) \\ \Im(\vec{x}) \end{bmatrix} \mid \vec{x} \in S_1 \right\}$,
- $\text{expand}(S_1, r^q) = \{r^{qk}\vec{x} \mid \vec{x} \in S_1 \wedge k \in \mathbf{N}\}$.

Proof The proof is given in Section 8.3.8. □

It is surprising that the intersection and union of two definable sets are always definable themselves. Indeed, S_1 and S_2 are images of recognizable sets of integer vector values by two linear transformations which might be different. It is worth noticing that their intersection or union can always be expressed as the image of a single set of integer vector values by the same transformation. This observation strengthens our claim that definable sets of complex vector values are a "good" generalization of recognizable sets of integer vector values.

Of course, not all sets of complex vector values are definable. The following theorems characterize families of sets that are proved to be undefinable. In Section 8.3.4, those theorems will be used as tools for establishing that the closure of some linear operations does not preserve the definable nature of sets.

Theorem 8.25 *Let $r \in \mathbf{N}$ with $r > 1$, and $a, b, c \in \mathbf{Z}$ with $a \neq 0$. The set*

$$S = \{ak^2 + bk + c \mid k \in \mathbf{N}\}$$

is not r-definable.

Proof The proof is given in Section 8.3.8. □

Theorem 8.26 *Let $r, p \in \mathbf{N}_0$ with $r > 1$, $\lambda \in \mathbf{C}$ such that $\lambda^p = 1$, and $a, b, c, d \in \mathbf{C}$ with $a \notin \mathbf{R} \setminus \mathbf{Q}$. The set*

$$S = \left\{ \lambda^k \begin{bmatrix} (j+a)(k+b)+c \\ j+d \end{bmatrix} \mid j, k \in \mathbf{N} \right\}$$

is not r-definable.

Proof The proof is given in Section 8.3.8. □

Theorem 8.27 *Let $r \in \mathbf{N}$ with $r > 1$, $\lambda \in \mathbf{C}$ such that there do not exist $p \in \mathbf{N}_0$ and $m \in \mathbf{N}$ such that $\lambda^p = r^m$. The set*

$$S = \{\lambda^k \mid k \in \mathbf{N}\}$$

is not r-definable.

Proof The proof is given in Section 8.3.8. □

Theorem 8.28 *Let $r, p, m \in \mathbf{N}_0$ with $r > 1$, $\lambda \in \mathbf{C}$ such that $\lambda^p = r^m$, and $a \in \mathbf{C}$ such that $a \notin \mathbf{R} \setminus \mathbf{Q}$. The set*

$$S = \{\lambda^k(k + a) \mid k \in \mathbf{N}\}$$

is not r-definable.

Proof The proof is given in Section 8.3.8. □

Theorem 8.29 *Let $r, p, m \in \mathbf{N}_0$ with $r > 1$, and $\lambda \in \mathbf{C}$ such that $\lambda^p = r^m$. The set*

$$S = \left\{ \begin{bmatrix} k \\ \lambda^k \end{bmatrix} \mid k \in \mathbf{N} \right\}$$

is not r-definable.

Proof The proof is given in Section 8.3.8. □

Theorem 8.30 *Let $r, p, m \in \mathbf{N}_0$ with $r > 1$, $\lambda \in \mathbf{C}$ such that $\lambda^p = r^m$, and $a \in \mathbf{C}$. The set*

$$S = \left\{ \begin{bmatrix} \lambda^k(j + a) \\ j \end{bmatrix} \mid j, k \in \mathbf{N} \right\}$$

is not r-definable.

Proof The proof is given in Section 8.3.8. □

Theorem 8.31 *Let $r, p_1, p_2, m_1, m_2 \in \mathbf{N}_0$ with $r > 1$, and $\lambda_1, \lambda_2 \in \mathbf{C}$ such that $\lambda_1^{p_1} = r^{m_1}$, $\lambda_2^{p_2} = r^{m_2}$ and $|\lambda_1| \neq |\lambda_2|$. The set*

$$S = \left\{ \begin{bmatrix} \lambda_1^k \\ \lambda_2^k \end{bmatrix} \mid k \in \mathbf{N} \right\}$$

is not r-definable.

Proof The proof is given in Section 8.3.8. □

8.3.4 Necessary Conditions

Here, we give conditions that must be verified by A if the guardless linear operation $\theta = (\vec{x} := A\vec{x} + \vec{b})$ is such that θ^* preserves the definable nature of sets. Those conditions consist of conditions on the eigenvalues of A, and on the size of the blocks of the Jordan form of A. For clarity sake, each group of conditions is presented separately. A summary of all the necessary conditions follows.

The idea behind the necessary conditions that will be developed is to show that the violation of any of them implies that there exists a set that is at the same time r-definable and not r-definable. The sets that are considered are related to the Jordan form of the transformation matrix. Precisely, we have the following result.

Theorem 8.32 *Let $n, r \in \mathbf{N}_0$ with $r > 1$, $\theta = (\vec{x} := A\vec{x} + \vec{b})$ with $A \in \mathbf{Z}^{n\times n}$ and $\vec{b} \in \mathbf{Z}^n$, $U \in \mathbf{C}^{n\times n}$ transforming A into its Jordan form A_J, $J_{m,\lambda}$ be a Jordan block of A_J with $m \in N_0$, $\lambda \in C$, π be the projection mapping A_J onto $J_{m,\lambda}$, and S be a r-definable subset of $\mathbf{Z}^n$. If $\theta^*(S)$ is r-definable, then the set*

$$S' = \{J_{m,\lambda}^k \vec{x} + \sum_{0\leq i<k} J_{m,\lambda}^i \vec{b'} \mid k \in \mathbf{N} \wedge \vec{x} \in \pi(U^{-1}S)\},$$

with $\vec{b'} = \pi(U^{-1}\vec{b})$, is r-definable.

Proof We have

$$\begin{aligned} \theta^*(S) &= \{A^k\vec{x} + \sum_{0\leq i<k} A^i\vec{b} \mid k \in \mathbf{N} \wedge \vec{x} \in S\} \\ &= \{UA_J^kU^{-1}\vec{x} + \sum_{0\leq i<k} UA_J^iU^{-1}\vec{b} \mid k \in \mathbf{N} \wedge \vec{x} \in S\}. \end{aligned}$$

If this set is r-definable, then applying Theorem 8.24 shows that $\pi(U^{-1}\theta^*(S))$ is r-definable. Hence the result. □

We are now ready to state the necessary conditions on the eigenvalues of the transformation matrix. The first condition expresses a relationship that must exist between those eigenvalues and the numeration basis.

Theorem 8.33 *Let $n, r \in \mathbf{N}_0$ with $r > 1$ and $\theta = (\vec{x} := A\vec{x} + \vec{b})$ with $A \in \mathbf{Z}^{n\times n}$ and $\vec{b} \in \mathbf{Z}^n$ be such that for every $\vec{v} \in \mathbf{Z}^n$, the set $\theta^*(\{\vec{v}\})$ is r-definable. For every nonzero eigenvalue λ of A, there exist $p \in \mathbf{N}_0$ and $m \in \mathbf{N}$ such that $\lambda^p = r^m$.*

Proof Let λ be a nonzero eigenvalue of A, A_J be the Jordan form of A, $J_{m,\lambda}$ be a block of A_J associated with λ ($m \in \mathbf{N}_0$), and π be the projection mapping A_J onto $J_{m,\lambda}$. From Theorem 8.32, it follows that for every $\vec{v} \in \mathbf{Z}^n$, the set

$$S' = \{J_{m,\lambda}^k \vec{v}' + \sum_{0\leq i<k} J_{m,\lambda}^i \vec{b'} \mid k \in \mathbf{N}\},$$

with $\vec{v}' = \pi(U^{-1}\vec{v})$ and $\vec{b}' = \pi(U^{-1}\vec{b})$, is r-definable. Let π' be the projection mapping each vector onto its component of highest index. There are two possible situations.

- *If* $\pi'(\vec{b}') = 0$. We choose $\vec{v} \in \mathbf{Z}^n$ such that $\pi'(\pi(U^{-1}\vec{v})) \neq 0$ (this is always possible, otherwise U^{-1} would be singular). According to Theorem 8.24, this implies that the set

$$\frac{1}{\pi'(\pi(U^{-1}\vec{v}))}\pi'(S') = \{\lambda^k \mid k \in \mathbf{N}\}$$

  is r-definable.

- *If* $\pi'(\vec{b}') \neq 0$. We choose $\vec{v} = \vec{0}$. According to Theorem 8.24, this implies that the following sets are r-definable:

$$\frac{1}{\pi'(\vec{b}')}\pi'(S') = \{\sum_{0 \leq i < k} \lambda^i \mid k \in \mathbf{N}\},$$

$$\{\lambda^k - 1 \mid k \in \mathbf{N}\},$$

$$\{\lambda^k \mid k \in \mathbf{N}\}.$$

We have thus established that the set

$$\{\lambda^k \mid k \in \mathbf{N}\}$$

is r-definable. The existence of $p \in \mathbf{N}_0$ and $m \in \mathbf{N}$ such that $\lambda^p = r^m$ is then a consequence of Theorem 8.27. □

The property expressed by Theorem 8.33 is easily adapted to sets of vector values that are definable in any basis.

Corollary 8.34 *Let* $n \in \mathbf{N}_0$ *and* $\theta = (\vec{x} := A\vec{x} + \vec{b})$ *with* $A \in \mathbf{Z}^{n \times n}$ *and* $\vec{b} \in \mathbf{Z}^n$ *be such that for every* $\vec{v} \in \mathbf{Z}^n$, *the set* $\theta^*(\{\vec{v}\})$ *is Presburger-definable. For every nonzero eigenvalue* λ *of* A, *there exists* $p \in \mathbf{N}_0$ *such that* $\lambda^p = 1$.

Proof Since every Presburger-definable set of integer vector values is r-definable in any basis $r > 1$, the result follows from applying Theorem 8.33 to two relatively prime bases r_1 and r_2 (chosen arbitrarily). □

Now, we go further and establish a correlation between the different eigenvalues of the transformation matrix.

Theorem 8.35 *Let* $n, r \in \mathbf{N}_0$ *with* $r > 1$, *and* $\theta = (\vec{x} := A\vec{x} + \vec{b})$ *with* $A \in \mathbf{Z}^{n \times n}$ *and* $\vec{b} \in \mathbf{Z}^n$ *be such that for every* $\vec{v} \in \mathbf{Z}^n$, *the sets* $\theta^*(\{\vec{v}\})$ *and* $\theta^*(\{j\vec{v} \mid j \in \mathbf{N}\})$ *are* r*-definable. Every pair of nonzero eigenvalues* (λ_1, λ_2) *of* A *is such that* $|\lambda_1| = |\lambda_2|$.

Proof The proof is by contradiction. Let $U \in \mathbf{C}^{n\times n}$ be a matrix transforming A into its Jordan form A_J. Let S be either equal to $\{\vec{v}\}$ or to $\{j\vec{v} \mid j \in \mathbf{N}\}$, with $\vec{v} \in \mathbf{Z}^n$. The set

$$\begin{aligned}\theta^*(S) &= \{A^k\vec{x} + \sum_{0\leq i<k} A^i\vec{b} \mid k \in \mathbf{N} \wedge \vec{x} \in S\} \\ &= \{UA_J^kU^{-1}\vec{x} + \sum_{0\leq i<k} UA_J^iU^{-1}\vec{b} \mid k \in \mathbf{N} \wedge \vec{x} \in S\}\end{aligned}$$

is r-definable. Suppose that A has two nonzero eigenvalues λ_1 and λ_2 such that $|\lambda_1| \neq |\lambda_2|$. Without loss of generality, we may assume that $|\lambda_1| < |\lambda_2|$. Let J_{m_1,λ_1} and J_{m_2,λ_2} $(m_1, m_2 \in \mathbf{N}_0)$ be two blocks of A_J respectively associated to λ_1 and to λ_2, and let π be the projection onto the two components matching the positions of the last line of J_{m_1,λ_1} and of the one of J_{m_2,λ_2} in A_J. According to Theorem 8.24, the set $S' = \pi(U^{-1}\theta^*(S))$ is r-definable. We have

$$S' = \left\{ \begin{bmatrix} \lambda_1^k & 0 \\ 0 & \lambda_2^k \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \sum_{0\leq i<k} \begin{bmatrix} \lambda_1^i & 0 \\ 0 & \lambda_2^i \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \mid \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in S'', k \in \mathbf{N} \right\},$$

with $S'' = \pi(U^{-1}S)$ and $\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \pi(U^{-1}\vec{b})$. We distinguish several situations.

- *If* $\lambda_1 = 1$ *and* $b_1 = 0$. We have

$$S' = \left\{ \begin{bmatrix} x_1 \\ \lambda_2^k x_2 + \frac{\lambda_2^k - 1}{\lambda_2 - 1} b_2 \end{bmatrix} \mid \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in S'', k \in \mathbf{N} \right\}.$$

  Let $\vec{v} \in \mathbf{Z}^n$ be such that the two components of $\pi(U^{-1}\vec{v})$ are different from zero (such a $\vec{v}$ always exists, otherwise U^{-1} would be singular). Choosing $S = \{j\vec{v} \mid j \in \mathbf{N}\}$, we obtain that the set

$$S' = \left\{ \begin{bmatrix} jv_1 \\ j\lambda_2^k v_2 + \frac{\lambda_2^k - 1}{\lambda_2 - 1} b_2 \end{bmatrix} \mid j, k \in \mathbf{N} \right\},$$

  with $\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \pi(U^{-1}\vec{v})$, is r-definable. From Theorem 8.24, it follows that the set

$$\left\{ \begin{bmatrix} \lambda_2^k \left(j + \frac{b_2}{v_2(\lambda_2 - 1)}\right) \\ j \end{bmatrix} \mid j, k \in \mathbf{N} \right\}$$

  is r-definable, which contradicts Theorem 8.30.

- *If* $\lambda_1 = 1$ *and* $b_1 \neq 0$. We have

$$S' = \left\{ \begin{bmatrix} x_1 + kb_1 \\ \lambda_2^k x_2 + \frac{\lambda_2^k - 1}{\lambda_2 - 1} b_2 \end{bmatrix} \mid \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in S'', k \in \mathbf{N} \right\}.$$

Let $\vec{v} \in \mathbf{Z}^n$ be such that the second component of $\pi(U^{-1}\vec{v})$ is different from $\frac{b_2}{1-\lambda_2}$ (such a $\vec{v}$ always exists, otherwise U^{-1} would be singular). Choosing $S = \{\vec{v} \mid j \in \mathbf{N}\}$, we obtain that the set

$$S' = \left\{ \begin{bmatrix} v_1 + kb_1 \\ \lambda_2^k v_2 + \frac{\lambda_2^k - 1}{\lambda_2 - 1} b_2 \end{bmatrix} \mid k \in \mathbf{N} \right\},$$

with $\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \pi(U^{-1}\vec{v})$, is r-definable. From Theorem 8.24, it follows that the set

$$\left\{ \begin{bmatrix} k \\ \lambda_2^k \end{bmatrix} \mid k \in \mathbf{N} \right\}$$

is r-definable, which contradicts Theorem 8.29.

- *If* $\lambda_1 \neq 1$. We have

$$S' = \left\{ \begin{bmatrix} \lambda_1^k x_1 + \frac{\lambda_1^k - 1}{\lambda_1 - 1} b_1 \\ \lambda_2^k x_2 + \frac{\lambda_2^k - 1}{\lambda_2 - 1} b_2 \end{bmatrix} \mid \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in S'', k \in \mathbf{N} \right\}.$$

Let $\vec{v} \in \mathbf{Z}^n$ be such that the two components of $\pi(U^{-1}\vec{v})$ are respectively different from $\frac{b_1}{1-\lambda_1}$ and from $\frac{b_2}{1-\lambda_2}$ (such a $\vec{v}$ always exists, otherwise U^{-1} would be singular). Choosing $S = \{\vec{v} \mid j \in \mathbf{N}\}$, we obtain that the set

$$S' = \left\{ \begin{bmatrix} \lambda_1^k v_1 + \frac{\lambda_1^k - 1}{\lambda_1 - 1} b_1 \\ \lambda_2^k v_2 + \frac{\lambda_2^k - 1}{\lambda_2 - 1} b_2 \end{bmatrix} \mid k \in \mathbf{N} \right\},$$

with $\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \pi(U^{-1}\vec{v})$, is r-definable. From Theorem 8.24, it follows that the set

$$\left\{ \begin{bmatrix} \lambda_1^k \\ \lambda_2^k \end{bmatrix} \mid k \in \mathbf{N} \right\}$$

is r-definable, which contradicts Theorem 8.31.

□

Before establishing the conditions that involve the Jordan blocks of the transformation matrix, we need to give a few lemmas.

Lemma 8.36 *Let $n, r \in \mathbf{N}_0$ with $n > 1, r > 1$, $\lambda \in \mathbf{C}$ such that $\lambda \neq 1$, $p \in \mathbf{N}_0$, $m \in \mathbf{N}$ such that $\lambda^p = r^m$, $q \in \mathbf{N}$ with $1 < q \leq n$, $V \in \mathbf{C}^{q \times n}$ of rank q, and $\vec{b} \in \mathbf{Z}^n$. There exists a r-definable set $S \subseteq \mathbf{Z}^n$ such that the set*

$$S' = \{J_{q,\lambda}^k \vec{x} + \sum_{0 \leq i < k} J_{q,\lambda}^i \vec{b'} \mid \vec{x} \in VS \wedge k \in \mathbf{N}\},$$

where $\vec{b'} = V\vec{b}$, is not r-definable.

Proof The proof is given in Section 8.3.8. □

Lemma 8.37 *Let $n, r \in \mathbf{N}_0$ with $n > 1, r > 1$, $q \in \mathbf{N}$ with $1 < q \leq n$, $V \in \mathbf{Q}^{q \times n}$ of rank q, and $\vec{b} \in \mathbf{Z}^n$. There exists a r-definable set $S \subseteq \mathbf{Z}^n$ such that the set*

$$S' = \{J_{q,1}^k \vec{x} + \sum_{0 \leq i < k} J_{q,1}^i \vec{b'} \mid \vec{x} \in VS \wedge k \in \mathbf{N}\},$$

where $\vec{b'} = V\vec{b}$, is not r-definable.

Proof The proof is given in Section 8.3.8. □

Lemma 8.38 *Let $n \in \mathbf{N}_0$ and $A \in \mathbf{Z}^{n \times n}$. There exists a nonsingular matrix $U \in \mathbf{C}^{n \times n}$ transforming A into its Jordan form A_J, and such that every row of U^{-1} at the same position as a line of a Jordan block $J_{q,\lambda}$ in A_J contains only rational components provided that λ is rational.*

The proof is given in Section 8.3.8. □

We are now ready to state the necessary condition on the size of the Jordan blocks of the transformation matrix.

Theorem 8.39 *Let $n, r \in \mathbf{N}_0$ with $r > 1$ and $\theta = (\vec{x} := A\vec{x} + \vec{b})$ with $A \in \mathbf{Z}^{n \times n}$ and $\vec{b} \in \mathbf{Z}^n$ be such that for every r-definable set $S \subseteq \mathbf{Z}^n$, the set $\theta^*(S)$ is r-definable. Let A_J be the Jordan form of A. Every Jordan block of A_J corresponding to a nonzero eigenvalue of A is of size 1.*

Proof The proof is by contradiction. Suppose that A_J has a Jordan block $J_{m,\lambda}$ such that $m > 1$. Let $U \in C^{n \times n}$ transforming A into A_J, and such that its rows at the same position as a line of $J_{m,\lambda}$ in A_J contain only rational components if $\lambda = 1$ (according to Lemma 8.38, such a U always exists). Let π be the projection mapping A_J onto $J_{m,\lambda}$. Applying Theorem 8.32, we have that for every r-definable set $S \subseteq \mathbf{Z}^n$, the set

$$S' = \{J_{m,\lambda}^k \vec{x} + \sum_{0 \leq i < k} J_{m,\lambda}^i \vec{b'} \mid k \in \mathbf{N} \wedge \vec{x} \in \pi(U^{-1}S)\},$$

with $\vec{b'} = \pi(U^{-1}\vec{b})$, is r-definable. Depending on the value of λ, this contradicts either Lemma 8.36 or Lemma 8.37. □

The necessary conditions are now complete. They can be summarized as follows.

Theorem 8.40 *Let $n, r \in \mathbf{N}_0$ with $r > 1$ and $\theta = (\vec{x} := A\vec{x} + \vec{b})$ with $A \in \mathbf{Z}^{n \times n}$ and $\vec{b} \in \mathbf{Z}^n$. If θ is such that for every r-definable set $S \subseteq \mathbf{Z}^n$, the set $\theta^*(S)$ is r-definable, then*

1. *There exist $p \in \mathbf{N}_0$ and $m \in \mathbf{N}$ such that every nonzero eigenvalue λ of A satisfies $\lambda^p = r^m$, and*

2. *The Jordan form of A is such that all the blocks corresponding to a nonzero eigenvalue are of size 1.*

Proof This result is a direct consequence of Theorems 8.33, 8.35, and 8.39. □

Corollary 8.41 *Let $n, r \in \mathbf{N}_0$ with $r > 1$, and $\theta = (\vec{x} := A\vec{x} + \vec{b})$ with $A \in \mathbf{Z}^{n \times n}$ and $\vec{b} \in \mathbf{Z}^n$. If θ is such that for every r-definable set $S \subseteq \mathbf{Z}^n$, the set $\theta^*(S)$ is r-definable, then there exists $p \in N_0$ such that*

1. *A^p has at most one nonzero eigenvalue λ, and*

2. *λ (if any) is an integer power of r, and*

3. *A^p is diagonalizable.*

Proof If θ is as required, then Theorem 8.40 implies that there exist $p' \in \mathbf{N}_0$ and $m' \in \mathbf{N}$ such that every nonzero eigenvalue λ' of A satisfies $(\lambda')^{p'} = r^{m'}$. Moreover, the Jordan form of A is such that all the blocks corresponding to a nonzero eigenvalue are of size 1. Let $a \in \mathbf{N}_0$ be such that $a > n/p'$, and let $p = ap'$, $m = am'$. Since every eigenvalue λ of A^p is the p-th power of an eigenvalue of A, we have $\lambda = r^m$. Furthermore, every matrix transforming A into its Jordan form A_J transforms A^p into A_J^p. This last matrix is diagonal, for any power of a block of size one is of size one, and the n-th power of a block associated to the eigenvalue zero is only composed of zeroes. □

Theorem 8.42 *Let $n \in \mathbf{N}_0$ and $\theta = (\vec{x} := A\vec{x} + \vec{b})$ with $A \in \mathbf{Z}^{n \times n}$ and $\vec{b} \in \mathbf{Z}^n$. If θ is such that for every Presburger-definable set $S \subseteq \mathbf{Z}^n$ the set $\theta^*(S)$ is Presburger-definable, then there exists $p \in N_0$ such that*

1. *The eigenvalues of A^p belong to $\{0, 1\}$, and*

2. *A^p is diagonalizable.*

Proof The result is obtained by applying the same reasoning as in the proofs of Theorems 8.33, 8.35, 8.39 and 8.40 with two relatively prime bases r_1 and r_2 (chosen arbitrarily). This can be done only because the sets $\{\vec{v}\}$ and $\{j\vec{v} \mid j \in \mathbf{N}\}$ used in the proof of Theorem 8.35 and in the ones of Lemmas 8.36 and of 8.37 are Presburger-definable. □

8.3.5 Sufficient Conditions

Here, we show that the necessary conditions given in Section 8.3.4 are also sufficient. In other words, if a guardless linear operation satisfies the conditions expressed by Corollary 8.41, then its closure preserves the definable nature of sets of vector values. This property is formalized as follows.

Theorem 8.43 *Let $n, r \in \mathbf{N}_0$ with $r > 1$ and $\theta = (\vec{x} := A\vec{x} + \vec{b})$ with $A \in \mathbf{Z}^{n\times n}$ and $\vec{b} \in \mathbf{Z}^n$. If there exists $p \in N_0$ such that A^p is diagonalizable, A^p has at most one nonzero eigenvalue λ, and λ (if any) is an integer power of r, then for any r-definable set $S \subseteq \mathbf{Z}^n$, the set $\theta^*(S)$ is r-definable.*

Proof Suppose that there exists such a p. For any r-definable set $S \subseteq \mathbf{Z}^n$, we have

$$\begin{aligned}\theta^*(S) &= \bigcup_{0\le j<p,\, k\in\mathbf{N}} \theta^{pk+j}(S)\\ &= \bigcup_{0\le j<p} \theta^j\left(\bigcup_{k\in\mathbf{N}} \theta^{pk}(S)\right)\\ &= \bigcup_{0\le j<p} \theta^j\left((\theta^p)^*(S)\right).\end{aligned}$$

According to Theorem 8.24, every θ^j preserves the r-definable nature of sets, as does the finite union of sets. Therefore, it is sufficient to prove that $(\theta^p)^*$ preserves r-definability. Let $S' = (\theta^p)^*(S)$, J be the Jordan form of A^p (we know that it is diagonal), and $U \in \mathbf{Q}^{n\times n}$ be a matrix transforming A^p into J. We have

$$S' = \{A^{pk}\vec{x} + \sum_{0\le i<k} A^{pi}\vec{b'} \mid \vec{x} \in S \wedge k \in \mathbf{N}\},$$

with $\vec{b'} = \sum_{0\le i<p} A^i\vec{b}$. Hence,

$$S' = \{UJ^kU^{-1}\vec{x} + \sum_{0\le i<k} UJ^iU^{-1}\vec{b'} \mid \vec{x} \in S \wedge k \in \mathbf{N}\}.$$

We distinguish two situations.

- *If all the eigenvalues of A^p belong to $\{0, 1\}$.* We have

$$\begin{aligned}S' &= S \cup \{UJU^{-1}\vec{x} + (k-1)UJU^{-1}\vec{b'} + \vec{b'} \mid \vec{x} \in S \wedge k \in \mathbf{N}_0\}\\ &= S \cup \{A^p\vec{x} + kA^p\vec{b'} + \vec{b'} \mid \vec{x} \in S \wedge k \in \mathbf{N}\}.\end{aligned}$$

  Since the last member of this equation is expressed in Presburger arithmetic, the set denoted by this term is r-definable, and so is S'.

- *If all the eigenvalues of A^p belong to $\{0, r^m\}$, with $m \in \mathbf{N}_0$.* We have

$$\begin{aligned}
S' &= \{UJ^kU^{-1}\vec{x} + \sum_{0\le i<k} UJ^iU^{-1}\vec{b'} \mid \vec{x} \in S \wedge k \in \mathbf{N}\} \\
&= S \cup \{r^{m(k-1)}UJU^{-1}\vec{x} + \sum_{0<i<k} r^{m(i-1)}UJU^{-1}\vec{b'} + \vec{b'} \\
&\qquad\qquad \mid \vec{x} \in S \wedge k \in \mathbf{N}_0\} \\
&= S \cup \{r^{mk}A^p\vec{x} + \sum_{0\le i<k} r^{mi}A^p\vec{b'} + \vec{b'} \mid \vec{x} \in S \wedge k \in \mathbf{N}\} \\
&= S \cup \{r^{mk}A^p\vec{x} + \frac{r^{mk}-1}{r^m-1}A^p\vec{b'} + \vec{b'} \mid \vec{x} \in S \wedge k \in \mathbf{N}\} \\
&= S \cup \left\{\frac{1}{r^m-1}\left[r^{mk}\left((r^m-1)A^p\vec{x} + A^p\vec{b'}\right) - A^p\vec{b'}\right] + \vec{b'}\right. \\
&\qquad\qquad \left. \mid \vec{x} \in S \wedge k \in \mathbf{N}\right\} \\
&= S \cup \frac{1}{r^m-1}\left[\text{expand}\left((r^m-1)A^pS + A^p\vec{b'}, r^m\right) - A^p\vec{b'}\right] + \vec{b'}
\end{aligned}$$

  According to Theorem 8.24, the last formula denotes a r-definable set.

□

A similar result holds for Presburger-definable sets.

Theorem 8.44 *Let $n, r \in \mathbf{N}_0$ with $r > 1$ and $\theta = (\vec{x} := A\vec{x} + \vec{b})$ with $A \in \mathbf{Z}^{n\times n}$ and $\vec{b} \in \mathbf{Z}^n$. If there exists $p \in N_0$ such that A^p is diagonalizable and has its eigenvalues in $\{0, 1\}$, then for any Presburger-definable set $S \subseteq \mathbf{Z}^n$, the set $\theta^*(S)$ is Presburger-definable.*

Proof The proof is identical to the first part of the proof of Theorem 8.43. □

8.3.6 Algorithms

The necessary and sufficient conditions given in Sections 8.3.4 and 8.3.5 are not easy to use in practice. Indeed, they are defined in terms of eigenvalues and of Jordan blocks, which can in general only be computed up to a limited accuracy. In this section, we give an algorithm for determining whether a given linear transformation with integer coefficients satisfies the necessary and sufficient conditions expressed by Theorem 8.40. This decision procedure is only based on integer arithmetic. An algorithm is also given for computing a finite-state representation of the set $\theta^*(S)$ given a representation of the set of vector values $S \subseteq \mathbf{Z}^n$ and a guardless linear operation θ that satisfies the necessary and sufficient conditions for preserving definability.

Let $r, n \in \mathbf{N}_0$ with $r > 1$, and θ be the guardless linear operation $\vec{x} := A\vec{x} + \vec{b}$ with $A \in \mathbf{Z}^{n\times n}$ and $\vec{b} \in \mathbf{Z}^n$. The first problem consists of checking whether $\theta^*(S)$ is

r-definable for every r-definable set $S \subseteq \mathbf{Z}^n$. In addition, if the answer is positive, we would like to compute $m \in \mathbf{N}$ and $p \in \mathbf{N}_0$ such that A^p is diagonalizable and has all its eigenvalues in $\{0, r^m\}$.

First, we check whether the eigenvalues of A satisfy the conditions required by Theorem 8.40. We know that those eigenvalues are the roots of the characteristic polynomial $\Pi_1(x)$ of A. Since this polynomial has integer coefficients, the product a of all its roots can easily be computed, as the ratio of its nonzero coefficients of lowest and of highest degree (this implies $a \in \mathbf{Q}$). According to Theorem 8.40, all the roots of $\Pi_1(x)$ must be of the same magnitude, and this magnitude must be equal to some rational power of r. Therefore, if $|a|$ is not a rational power of r, then θ does not preserve the r-definable nature of sets of vector values.

Let us now assume that $|a| = r^{(u/v)}$, with $u \in \mathbf{Z}$, $v \in \mathbf{N}_0$ and $\gcd(u, v) = 1$. The eigenvalues of A satisfy the conditions expressed by Theorem 8.40 if and only if every nonzero root of $\Pi_1(x)$ has the magnitude $|a|^{(1/n')}$, where n' is the difference between the highest and the lowest degrees of the nonzero coefficients of $\Pi_1(x)$. If $n' = 0$, then zero is the only root of $\Pi_1(x)$ and the condition is trivially satisfied. If $n' > 0$, then let $z = (n'v)/\gcd(n'v, u)$ and $y = (zu)/(n'v)$. Every eigenvalue of A^z that is different from zero must have the magnitude r^y. Therefore, each root of the characteristic polynomial $\Pi_2(x)$ of A^z must be either equal to zero or of magnitude r^y. Hence, if $k \in \mathbf{N}$ is the greatest integer such that $\Pi_2(x)$ is divisible by the polynomial x^k, then all the roots of the polynomial $\Pi_3(x) = \Pi'_2(r^y x)$, where $\Pi'_2(x) = \Pi_2(x)/x^k$, must be complex roots of 1.

The problem consisting of checking whether the eigenvalues of A satisfy the conditions expressed by Theorem 8.40 has thus been reduced to checking whether all the roots of $\Pi_3(x)$ are complex roots of 1. This is the case if and only if there exists $l \in \mathbf{N}_0$ such that $\Pi_3(x)$ divides $x^l - 1$. Since the polynomial $\Pi_3(x)$ has integer coefficients, such an integer l exists if and only if $\Pi_3(x)$ is a product of cyclotomic polynomials. Checking this by trying successively to divide $\Pi_3(x)$ by $\Phi_1(x), \Phi_2(x), \Phi_3(x), \ldots$ introduces two difficulties. First, given an integer $i \in \mathbf{N}_0$, computing the coefficients of $\Phi_i(x)$ is tedious. One must therefore find a way of testing the divisibility of $\Pi_3(x)$ by $\Phi_i(x)$ without computing explicitly $\Phi_i(x)$. Second, one must find an upper bound on the indices i of the $\Phi_i(x)$ that have to be considered.

The first problem is solved by the following theorem.

Theorem 8.45 *Let $i \in N_0$ and $\Pi(x)$ be a polynomial with integer coefficients such that for every $0 < j < i$, $\Pi(x)$ is not divisible by the cyclotomic polynomial $\Phi_j(x)$. The polynomial $\Pi(x)$ is divisible by $\Phi_i(x)$ if and only if the degree of the polynomial $\gcd(x^i - 1, \Pi(x))$ is at least equal to 1.*

Proof We have $x^i - 1 = \Phi_i(x)\Phi_{j_1}(x) \cdots \Phi_{j_q}(x)$, where each j_k is such that $0 < j_k < i$. Since the factorization of $x^i - 1$ into cyclotomic polynomials is unique, the result is immediate. □

As a consequence of this theorem, trying successively to divide $\Pi_3(x)$ by $\Phi_1(x)$, $\Phi_2(x), \Phi_3(x), \ldots$ can be done by dividing successively $\Pi_3(x)$ by its common factors with $x-1, x^2-1, x^3-1, \ldots$. The conditions on the eigenvalues of A are satisfied if and only if one eventually obtains a polynomial of degree 0.

It remain to give an upper bound on the indices i of the cyclotomic polynomials $\Phi_i(x)$ that can potentially divide $\Pi_3(x)$. Intuitively, the idea is that it is useless to consider the $\Phi_i(x)$ whose degree is greater than the one of Π_3. We have the following theorem.

Theorem 8.46 *For every integer $k \in \mathbf{N}_0$ and for every degree $d \in \mathbf{N}$ such that $k > 210 \left(\frac{d}{48}\right)^{\log_{10} 11}$, we have* $\text{degree}(\Phi_k(x)) > d$.

Proof It is known [IR90] that $\text{degree}(\Phi_k(x)) = \phi(k)$, where ϕ is the Euler function, defined as

$$\phi(k) = k \left(1 - \frac{1}{p_1}\right)\left(1 - \frac{1}{p_2}\right) \cdots \left(1 - \frac{1}{p_q}\right),$$

where $p_1, p_2, \ldots, p_q$ are the (distinct) prime factors of k.

Assume first that $q \geq 5$, i.e., that k has at least five distinct prime factors. We have $p_1 \geq 2$, $p_2 \geq 3$, $p_3 \geq 5$, $p_4 \geq 7$, as well as $p_i \geq 11$ for all $i \geq 5$. These inequalities imply

$$\left(1 - \frac{1}{p_1}\right)\left(1 - \frac{1}{p_2}\right)\left(1 - \frac{1}{p_3}\right)\left(1 - \frac{1}{p_4}\right) \geq \left(1 - \frac{1}{2}\right)\left(1 - \frac{1}{3}\right)\left(1 - \frac{1}{5}\right)\left(1 - \frac{1}{7}\right) \tag{8.1}$$

and

$$\left(1 - \frac{1}{p_5}\right)\left(1 - \frac{1}{p_6}\right) \cdots \left(1 - \frac{1}{p_q}\right) \geq \left(1 - \frac{1}{11}\right)^{(q-4)}. \tag{8.2}$$

Moreover, since $k \geq p_1 \cdots p_q$, we have $k \geq 2.3.5.7.11^{(q-4)}$, and hence $q - 4 \leq \log_{11}(k/210)$. Replacing into Equation (8.2), we obtain

$$\left(1 - \frac{1}{p_5}\right)\left(1 - \frac{1}{p_6}\right) \cdots \left(1 - \frac{1}{p_q}\right) \geq \left(1 - \frac{1}{11}\right)^{\log_{11}\left(\frac{k}{210}\right)}. \tag{8.3}$$

Introducing Equations (8.1) and (8.3) into the expression of $\phi(k)$, we obtain

$$\phi(k) \geq k \left(1 - \frac{1}{2}\right)\left(1 - \frac{1}{3}\right)\left(1 - \frac{1}{5}\right)\left(1 - \frac{1}{7}\right)\left(1 - \frac{1}{11}\right)^{\log_{11}\left(\frac{k}{210}\right)}.$$

Now, let us show that the previous inequality also holds if $q < 5$, i.e., if k does not have more than four distinct prime factors. Let $p'_1 = 2$, $p'_2 = 3$, $p'_3 = 5$ and $p'_4 = 7$. We have

$$\phi(k) \quad = \quad k \left(1 - \frac{1}{p_1}\right) \cdots \left(1 - \frac{1}{p_q}\right)$$

$$\begin{aligned} &\geq k\left(1-\frac{1}{p'_1}\right)\cdots\left(1-\frac{1}{p'_q}\right) \\ &= k\left(1-\frac{1}{2}\right)\left(1-\frac{1}{3}\right)\left(1-\frac{1}{5}\right)\left(1-\frac{1}{7}\right)\left(1-\frac{1}{11}\right)^{\log_{11}\left(\frac{k}{210}\right)}\varphi(k), \end{aligned}$$

with

$$\varphi(k)=\frac{1}{\left(1-\frac{1}{p'_{q+1}}\right)\cdots\left(1-\frac{1}{p'_4}\right)\left(1-\frac{1}{11}\right)^{\log_{11}\left(\frac{k}{210}\right)}}.$$

It is thus sufficient to show that $\varphi(k)\geq 1$, i.e., that

$$\left(1-\frac{1}{p'_{q+1}}\right)\cdots\left(1-\frac{1}{p'_4}\right)\leq\left(1-\frac{1}{11}\right)^{-\log_{11}\left(\frac{k}{210}\right)}.$$

For every $i\in\{q+1,\ldots,4\}$, we have

$$\left(1-\frac{1}{p'_i}\right)\leq 1-\frac{1}{11}\leq\left(1-\frac{1}{11}\right)^{\log_{11}p'_i},$$

which yields

$$\begin{aligned} \left(1-\frac{1}{p'_{q+1}}\right)\cdots\left(1-\frac{1}{p'_4}\right) &\leq \left(1-\frac{1}{11}\right)^{(\log_{11}p'_{q+1}+\cdots+\log_{11}p'_4)} \\ &= \left(1-\frac{1}{11}\right)^{\log_{11}(p'_{q+1}\cdots p'_4)}. \end{aligned}$$

Since $k\geq p'_1\cdots p'_q$ and $p'_1\cdots p'_4=210$, we have

$$p'_{q+1}\cdots p'_4=\frac{210}{p'_1\cdots p'_q}\geq\frac{210}{k}.$$

Therefore,

$$\begin{aligned} \left(1-\frac{1}{11}\right)^{\log_{11}(p'_{q+1}\cdots p'_4)} &\leq \left(1-\frac{1}{11}\right)^{\log_{11}\left(\frac{210}{k}\right)} \\ &= \left(1-\frac{1}{11}\right)^{-\log_{11}\left(\frac{k}{210}\right)} \end{aligned}$$

and hence $\varphi(k)\geq 1$.

In summary, we have for every $k\in\mathbf{N}_0$

$$\phi(k)\geq k\left(1-\frac{1}{2}\right)\left(1-\frac{1}{3}\right)\left(1-\frac{1}{5}\right)\left(1-\frac{1}{7}\right)\left(1-\frac{1}{11}\right)^{\log_{11}\left(\frac{k}{210}\right)}.$$

This yields

$$\begin{aligned}\phi(k) &\geq \frac{8k}{35} e^{(\log_{11}(10)-1)\log_e \frac{k}{210}} \\ &= \frac{8k}{35}\left(\frac{k}{210}\right)^{\log_{11}(10)-1} \\ &= 48\left(\frac{k}{210}\right)^{\log_{11} 10}.\end{aligned}$$

If k is such that $k > 210\left(\frac{d}{48}\right)^{\log_{10} 11}$, then this last expression implies $\phi(k) > d$. □

Remark that the same reasoning can be followed so as to obtain a better bound (up to an arbitrary amount of accuracy), by considering a greater number of prime factors in the expansion of $\phi(k)$. The choice of expanding only the first five prime factors was motivated by an explicit computation of the first few hundred cyclotomic polynomials, which demonstrated that the bound expressed by Theorem 8.46 is nearly optimal for these polynomials.

It remains to check whether the sizes of the Jordan blocks of A satisfy the conditions required by Theorem 8.40. We assume that the conditions on the eigenvalues of A are satisfied. Let $i_1, i_2, \ldots, i_q$ ($q \in \mathbf{N}$) be all the integers i such that $\Pi_3(x)$ has common factors with $x^i - 1$. The least common multiple l of $i_1, i_2, \ldots, i_q$ is such that the l-th power of every root of $\Pi_3(x)$ is exactly equal to 1. This means that all the nonzero eigenvalues of A^{zl} are equal to r^{yl}. Let

$$l' = \begin{cases} l & \text{if } zl \geq n \text{ or } \Pi_2(x) = \Pi_2'(x), \\ l\lceil n/(zl)\rceil & \text{if } zl < n \text{ and } \Pi_2(x) \neq \Pi_2'(x), \end{cases}$$

and let $m = yl'$, $p = zl'$. All the eigenvalues of A^p belong to $\{0, r^m\}$. If A^p has the eigenvalue 0, then the definition of l' yields $p \geq n$, which implies that the Jordan blocks of A^p associated to the eigenvalue 0 are only composed of zeroes. The condition on the size of the Jordan blocks of A will thus be satisfied if and only if A^p is diagonalizable. This can be checked thanks to the following result.

Theorem 8.47 *A square matrix is diagonalizable if and only if its minimal polynomial has only simple roots.*

Proof A proof of this well-known result can be found in [Bod59] or [Fra68]. □

In the present case, we know that the minimal polynomial of A^p has to be either 0, x, $x - r^m$ or $x(x - r^m)$, depending on the eigenvalues of A. This can be checked explicitly.

An algorithm formalizing the decision procedure that has just been developed is given in Figures 8.8 and 8.9. (In this algorithm, the test performed at Line 11 can easily be carried out by comparing the prime factors of a_0, a_1 and r.)

function DEFINABLE-CLOSURE?(basis r, dimension n, integer matrix A) : $\{\mathbf{T}, \mathbf{F}\} \times \mathbf{N} \times \mathbf{N}_0$

1: **var** Π_1, Π_2, Π : polynomials with integer coefficients;
2: $d_0, d_1, a_0, a_1, a, u, v, n', z, y, i, m, p, l$: integers;
3: M : integer matrix;
4: **begin**
5: $\Pi_1(x)$:= characteristic polynomial of A;
6: d_0 := lowest degree of the nonzero terms of $\Pi_1(x)$;
7: d_1 := highest degree of the nonzero terms of $\Pi_1(x)$;
8: a_0 := coefficient of $\Pi_1(x)$ with the degree d_0;
9: a_1 := coefficient of $\Pi_1(x)$ with the degree d_1;
10: $a := a_0/a_1$;
11: **if** $(r > 1 \wedge \log_r(|a|) \notin \mathbf{Q}) \vee (r = 1 \wedge |a| \neq 1)$ **then return** $(\mathbf{F}, 0, 0)$;
12: **if** $r = 1$ **then** $(u, v) := (1, 1)$
13: **else let** $u/v := \log_r(|a|)$ **such that** $u \in \mathbf{Z} \wedge v \in \mathbf{N}_0 \wedge \gcd(u, v) = 1$;
14: $n' := d_1 - d_0$;
15: **if** $n' = 0$ **then return** $(\mathbf{T}, 0, n)$;
16: $z := (n'v)/\gcd(n'v, u)$;
17: $y := (zu)/(n'v)$;
18: $\Pi_2(x)$:= characteristic polynomial of A^z;
19: $n' := n$;
20: **while** x divides $\Pi_2(x)$ **do**
21: **begin**
22: $\Pi_2(x) := \Pi_2(x)/x$;
23: $n' := n' - 1$
24: **end**;
25: $\Pi_3(x) := \Pi_2(r^y x)$;
26: $l := 1$;

(. . .)

Figure 8.8: Decision procedure for the preservation of r-definability by the closure of a guardless linear operation.

```
    (...)
27:     for i := 1 to ⌊210(n'/48)^(log10 11)⌋ do
28:         begin
29:             Π(x) := gcd(x^i − 1, Π3(x));
30:             if degree(Π(x)) > 0 then
31:                 begin
32:                     l := lcm(l, i);
33:                     while Π(x) divides Π3(x) do
34:                         Π3(x) := Π3(x)/Π(x)
35:                 end
36:         end;
37:     if degree(Π3(x)) > 0 then return (F, 0, 0);
38:     if zl < n ∧ n' < n then l := l⌈n/(zl)⌉;
39:     (m, p) := (yl, zl);
40:     M := I_n;
41:     if n' > 0 then M := (A^p − r^m I_n)M;
42:     if n' < n then M := A^p M;
43:     if A^p = (0) ∨ M = (0) then return (T, m, p);
44:     return (F, 0, 0)
45: end.
```

Figure 8.9: Decision procedure for the preservation of r-definability by the closure of a guardless linear operation (continued).

```
function META-BASIS?(basis r, dimension n, linear operation x⃗ := Ax⃗ + b⃗) : {T, F}
1:  var m, p : integers;
2:      t : boolean;
3:  begin
4:      (t, m, p) := DEFINABLE-CLOSURE?(r, n, A);
5:      return t
6:  end.
```

Figure 8.10: Implementation of META? for guardless linear operations in a given basis.

Theorem 8.48 *Let $r, n \in \mathbf{N}_0$ and θ be the guardless linear operation $\vec{x} := A\vec{x} + \vec{b}$ with $A \in \mathbf{Z}^{n\times n}$ and $\vec{b} \in \mathbf{Z}^n$. The set $\theta^*(S)$ is r-definable for every r-definable set $S \subseteq \mathbf{Z}^n$ if and only if DEFINABLE-CLOSURE?(r, n, A) returns a triple of the form $(\mathbf{T}, m, p)$, with $m \in \mathbf{N}$ and $p \in \mathbf{N}_0$. If this is the case, then m and p are such that A^p is diagonalizable and has all its eigenvalues in $\{0, r^m\}$.*

Proof The algorithm in Figures 8.8 and 8.9 is a direct implementation of the computation method discussed in this section. In Lines 41–42, the condition on the minimal polynomial of A^p is checked by taking advantage of the facts that $n' > 0$ if and only if A^p has the eigenvalue r^m, and that $n' < n$ if and only if A^p has the eigenvalue 0. □

Theorem 8.49 *Let $n \in \mathbf{N}_0$ and θ be the guardless linear operation $\vec{x} := A\vec{x}+\vec{b}$ with $A \in \mathbf{Z}^{n\times n}$ and $\vec{b} \in \mathbf{Z}^n$. The set $\theta^*(S)$ is Presburger-definable for every Presburger-definable set $S \subseteq \mathbf{Z}^n$ if and only if DEFINABLE-CLOSURE?$(1, n, A)$ returns a triple of the form $(\mathbf{T}, m, p)$, with $m \in \mathbf{N}$ and $p \in \mathbf{N}_0$. If this is the case, then p is such that A^p is diagonalizable and has all its eigenvalues in $\{0, 1\}$.*

Proof The result is a direct consequence of Theorems 8.42 and 8.48. □

Algorithms implementing the predicate META? for guardless linear operations with respect to sets of vector values representable in a given basis $r > 1$ or in any basis are respectively given in Figures 8.10 and 8.11.

It remains to give an algorithm for computing the image of a definable set of vector values $S \subseteq \mathbf{Z}^n$ ($n \in \mathbf{N}$) by the closure of a guardless linear operation $\vec{x} := A\vec{x} + \vec{b}$ that satisfies META?. An expression of this image in terms of S and of operations preserving the definable nature of sets has already been obtained in the proof of Theorem 8.43. Algorithms based on that result are given in Figures 8.12 and 8.13.

function META-PRESBURGER?(dimension n, linear operation $\vec{x} := A\vec{x} + \vec{b}$) : $\{\mathbf{T}, \mathbf{F}\}$

1: **var** m, p : integers;
2: t : boolean;
3: **begin**
4: $(t, m, p) :=$ DEFINABLE-CLOSURE?$(1, n, A)$;
5: **return** t
6: **end**.

Figure 8.11: Implementation of META? for guardless linear operations in any basis.

function APPLY-STAR-GUARDLESS-BASIS(basis r, dimension n, NDD $\mathcal{A}$, linear operation $\vec{x} := A\vec{x} + \vec{b}$) : NDD

1: **var** m, p : integers;
2: $\vec{b}'$: integer vector;
3: $\mathcal{A}'$: NDD;
4: **begin**
5: $(\mathbf{T}, m, p) :=$ DEFINABLE-CLOSURE?(r, n, A);
6: $\vec{b}' := \sum_{0 \le i < p} A^i \vec{b}$;
7: **if** $m = 0$ **then**
8: $\mathcal{A}' := \mathrm{NDD}\Big(\mathrm{SET}(\mathcal{A}) \cup \{\vec{y} \in \mathbf{Z}^n \mid (\exists k \in \mathbf{N}, \vec{x} \in \mathrm{SET}(\mathcal{A}))(\vec{y} = A^p\vec{x} + kA^p\vec{b}' + \vec{b}')\}\Big)$
9: **else**
10: $\mathcal{A}' := \mathrm{NDD}\Big(\mathrm{SET}(\mathcal{A}) \cup (1/(r^m - 1))\Big[\mathrm{expand}\Big((r^m - 1)A^p\,\mathrm{SET}(\mathcal{A}) + A^p\vec{b}', r^m\Big) - A^p\vec{b}'\Big] + \vec{b}'\Big)$;
11: **return** $\mathrm{NDD}\left(\bigcup_{0 \le k < p}\left(A^k\,\mathrm{SET}(\mathcal{A}') + \sum_{0 \le i < k} A^i\vec{b}\right)\right)$
12: **end**.

Figure 8.12: Image of an NDD by the closure of a guardless linear operation in a given basis.

function APPLY-STAR-GUARDLESS-PRESBURGER(dimension n, NDD $\mathcal{A}$,
linear operation $\vec{x} := A\vec{x} + \vec{b}$) : NDD

1: **var** p : integer;
2: $\vec{b'}$: integer vector;
3: $\mathcal{A}'$: NDD;
4: **begin**
5: $(\mathbf{T}, 0, p) :=$ DEFINABLE-CLOSURE?$(1, n, A)$;
6: $\vec{b'} := \sum_{0 \le i < p} A^i \vec{b}$;
7: $\mathcal{A}' :=$ NDD$\Big(\text{SET}(\mathcal{A}) \cup \{\vec{y} \in \mathbf{Z}^n \mid (\exists k \in \mathbf{N}, \vec{x} \in \text{SET}(\mathcal{A}))$
$(\vec{y} = A^p\vec{x} + kA^p\vec{b} + \vec{b'})\}\Big)$;
8: **return** NDD $\left(\bigcup_{0 \le k < p} \left(A^k \,\text{SET}(\mathcal{A}') + \sum_{0 \le i < k} A^i \vec{b}\right)\right)$
9: **end**.

Figure 8.13: Image of an NDD by the closure of a guardless linear operation in any basis.

Theorem 8.50 *Let $r, n \in \mathbf{N}_0$ with $r > 1$ and θ be the guardless linear operation $\vec{x} := A\vec{x} + \vec{b}$, with $A \in \mathbf{Z}^{n \times n}$ and $\vec{b} \in \mathbf{Z}^n$, such that META-BASIS?$(r, n, \theta) = \mathbf{T}$. If $\mathcal{A}$ is an NDD representing the set of vector values $S \subseteq \mathbf{Z}^n$ in basis r, then APPLY-STAR-GUARDLESS-BASIS$(r, n, \mathcal{A}, \theta)$ is an NDD representing the set $\theta^*(S)$ in basis r.*

Proof The algorithm in Figure 8.12 is a direct implementation of the computation performed in the proof of Theorem 8.43. □

Theorem 8.51 *Let $n \in \mathbf{N}_0$ and θ be the guardless linear operation $\vec{x} := A\vec{x} + \vec{b}$, with $A \in \mathbf{Z}^{n \times n}$ and $\vec{b} \in \mathbf{Z}^n$, such that META-PRESBURGER?$(n, \theta) = \mathbf{T}$. If $\mathcal{A}$ is an NDD representing the Presburger-definable set of vector values $S \subseteq \mathbf{Z}^n$ in some basis $r > 1$, then APPLY-STAR-GUARDLESS-PRESBURGER$(n, \mathcal{A}, \theta)$ is an NDD representing the Presburger-definable set $\theta^*(S)$ in basis r.*

Proof The algorithm in Figure 8.13 is a direct implementation of the computation performed in the proof of Theorem 8.43. □

8.3.7 Linear Operations with Guard

The problem addressed here consists of checking whether a linear operation $\theta = (P\vec{x} \le \vec{q} \to \vec{x} := A\vec{x} + \vec{b})$, where $n, m \in \mathbf{N}$, $P \in \mathbf{Z}^{m \times n}$, $\vec{q} \in \mathbf{Z}^m$, $A \in \mathbf{Z}^{n \times n}$ and

$\vec{b} \in \mathbf{Z}^n$, is such that $\theta^*(S)$ is definable for every definable set of vector values $S \subseteq \mathbf{Z}^n$. In addition, we would like to compute an NDD representing $\theta^*(S)$ from an NDD representing S.

We do not provide a general solution to this problem. Instead, we show that the results developed in Sections 8.3.4, 8.3.5 and 8.3.6 can be adapted with little difficulty to linear operations with guards, in the form of a *sufficient* condition for the preservation of definability by the closure of a linear operation.

Precisely, the sufficient condition is a consequence of a remarkable property: if θ is such that its underlying guardless operation $\vec{x} := A\vec{x} + \vec{b}$ satisfies the necessary and sufficient conditions expressed by Theorem 8.40, then for every definable set $S \subseteq \mathbf{Z}^n$, the set $\theta^*(S)$ is definable. Moreover, an NDD representing $\theta^*(S)$ can be computed from an NDD representing S. Formally, we have the following result.

Theorem 8.52 *Let $n \in \mathbf{N}$, $r \in \mathbf{N}$ with $r > 1$, $m \in \mathbf{N}$, and $\theta = (\vec{x} := P\vec{x} \leq \vec{q} \rightarrow A\vec{x} + \vec{b})$ with $P \in \mathbf{Z}^{m \times n}$, $q \in \mathbf{Z}^m$, $A \in \mathbf{Z}^{n \times n}$ and $\vec{b} \in \mathbf{Z}^n$. If there exists $p \in N_0$ such that A^p is diagonalizable, has at most one nonzero eigenvalue λ, and λ (if any) is an integer power of r, then for any r-definable set $S \subseteq \mathbf{Z}^n$, the set $\theta^*(S)$ is r-definable.*

Proof Suppose that there exists a suitable p. Let $S \subseteq \mathbf{Z}^n$ be a r-definable set, θ' be the guardless linear operation $(\vec{x} := A\vec{x} + \vec{b})$, and $V = \{\vec{x} \in \mathbf{Z}^n \mid P\vec{x} \leq \vec{q}\}$. We have

$$\begin{aligned}
\theta^*(S) &= \{(\theta')^k(\vec{x}) \mid \vec{x} \in S \wedge k \in \mathbf{N} \wedge \bigwedge_{0 \leq i < k} (\theta')^i(\vec{x}) \in V\} \\
&= \{(\theta')^{pk+j}(\vec{x}) \mid \vec{x} \in S \wedge k \in \mathbf{N} \wedge 0 \leq j < p \\
&\qquad \wedge \bigwedge_{0 \leq i < j} [(\theta')^i(\vec{x}) \in V] \wedge \bigwedge_{0 \leq i < k} \bigwedge_{0 \leq l < p} [(\theta')^l((\theta')^{pi+j}(\vec{x})) \in V]\}.
\end{aligned}$$

Let

$$V' = \{\vec{x} \in \mathbf{Z}^n | \bigwedge_{0 \leq l < p} [(\theta')^l(\vec{x}) \in V]\}.$$

The expression of $\theta^*(S)$ becomes

$$\begin{aligned}
\theta^*(S) &= \{(\theta')^{pk+j}(\vec{x}) \mid \vec{x} \in S \wedge k \in \mathbf{N} \wedge 0 \leq j < p \\
&\qquad \wedge \bigwedge_{0 \leq i < j} [(\theta')^i(\vec{x}) \in V] \wedge \bigwedge_{0 \leq i < k} [(\theta')^{pi+j}(\vec{x}) \in V']\} \\
&= \bigcup_{0 \leq j < p} S_j,
\end{aligned}$$

with for every $j \in \{0, 1, \ldots, p-1\}$,

$$S_j = \{(\theta')^{pk+j}(\vec{x}) \mid \vec{x} \in S \wedge k \in \mathbf{N} \wedge \bigwedge_{0 \leq i < j} [(\theta')^i(\vec{x}) \in V] \wedge \bigwedge_{0 \leq i < k} [(\theta')^{pi+j}(\vec{x}) \in V']\}.$$

Let us define

$$U_j = \{\vec{x} \in \mathbf{Z}^n | (\exists \vec{x}\,' \in S)(\vec{x} = (\theta')^j(\vec{x}\,') \wedge \bigwedge_{0 \leq i < j} [(\theta')^i(\vec{x}\,') \in V])\}.$$

We obtain

$$S_j = \{(\theta')^{pk}(\vec{x}) \mid k \in \mathbf{N} \wedge \vec{x} \in U_j \wedge \bigwedge_{0 \leq i < k} [(\theta')^{pi}(\vec{x}) \in V']\}.$$

By construction, V' is a convex set. Moreover, it follows from the algorithm in Figure 8.12 that all the the vectors belonging to $\{(\theta')^{pi}(\vec{x}), (\theta')^{p(i+1)}(\vec{x}), \ldots\}$ are colinear. It follows that for any $k > 1$, the condition

$$\bigwedge_{0 \leq i < k} [(\theta')^{pi}(\vec{x}) \in V']$$

is equivalent to

$$\vec{x} \in V' \wedge (\theta')^{p}(\vec{x}) \in V' \wedge (\theta')^{(k-1)p}(\vec{x}) \in V'.$$

Therefore, we have

$$\begin{aligned} S_j &= U_j \cup \{(\theta')^{p}(\vec{x}) \mid \vec{x} \in U_j \cap V'\} \\ &\quad \cup \{(\theta')^{pk}(\vec{x}) \mid k \in \mathbf{N} \wedge k \geq 2 \wedge \vec{x} \in U_j \cap V' \\ &\qquad \wedge (\theta')^{p}(\vec{x}) \in V' \wedge (\theta')^{p(k-1)}(\vec{x}) \in V'\} \\ &= (\theta')^{p}(U_j \cap V') \cup (\theta')^{p}((\theta')^{p}([(\theta')^{p}]^{*}(U_j \cap V' \cap V'')) \cap V'), \end{aligned}$$

with $V'' = \{\vec{x} \in \mathbf{Z}^n \mid (\theta')^{p}(\vec{x}) \in V'\}$. Since V', V'' and every U_j are Presburger-definable (and thus r-definable), every S_j is r-definable. It follows that $\theta^*(S)$ is r-definable as well. □

Unfortunately, the reciprocal of Theorem 8.52 is not true. Indeed, there are guarded linear operations that preserve the r-definable nature of sets of vector values, but whose underlying guardless operator does not. The conditions expressed by Theorem 8.52 are thus sufficient, but not necessary. Obtaining necessary and sufficient conditions over guarded linear operations that preserve the r-definable nature of sets of vector values seems to be a very difficult problem[4].

A result similar to Theorem 8.52 holds for Presburger-definable sets.

Theorem 8.53 *Let $n \in \mathbf{N}$, $m \in \mathbf{N}$, and $\theta = (\vec{x} := P\vec{x} \leq \vec{q} \rightarrow A\vec{x} + \vec{b})$, with $P \in \mathbf{Z}^{m \times n}$, $q \in \mathbf{Z}^m$, $A \in \mathbf{Z}^{n \times n}$ and $\vec{b} \in \mathbf{Z}^n$. If there exists $p \in N_0$ such that A^p is diagonalizable, has at most one nonzero eigenvalue λ, and $\lambda = 1$, then for any Presburger-definable set $S \subseteq \mathbf{Z}^n$, the set $\theta^*(S)$ is Presburger-definable.*

[4]Intuitively, the difficulty originates from the fact that, if a linear operation θ does not satisfy the hypotheses of Theorem 8.52, then the trajectory $\{\theta^k(\vec{v}) \mid k \in \mathbf{N}\}$ of an individual vector value $\vec{v} \in \mathbf{Z}^n$ to which θ is repeatedly applied is in general non-linear. This makes a manageable description of $\theta^*(S)$, for a subset S of $\mathbf{Z}^n$, much more difficult to obtain.

Identical to the proof of Theorem 8.52. □

As a consequence of the two previous theorems, the implementations of the predicate META? for guarded linear operations with respect to a given basis or to any basis are identical to those developed for guardless linear operations. The two algorithms have been given in Figures 8.10 and 8.11.

It remains to give an algorithm for computing the image of a definable set of vector values $S \subseteq \mathbf{Z}^n$ ($n \in \mathbf{N}$) by the closure of a guarded linear operation ($P\vec{x} \leq \vec{q} \rightarrow \vec{x} := A\vec{x} + \vec{b}$) that satisfies META?. An expression of this image in terms of S and of operations preserving the definable nature of sets is given in the proof of Theorem 8.52. Algorithms based on that result are given in Figures 8.14 and 8.15.

Theorem 8.54 *Let $r, n \in \mathbf{N}_0$ with $r > 1$, and θ be the linear operation ($P\vec{x} \leq \vec{q} \rightarrow \vec{x} := A\vec{x} + \vec{b}$), with $m \in \mathbf{N}$, $P \in \mathbf{Z}^{m \times n}$, $q \in \mathbf{Z}^m$, $A \in \mathbf{Z}^{n \times n}$ and $\vec{b} \in \mathbf{Z}^n$ such that META-BASIS?$(r, n, \theta) = \mathbf{T}$. If $\mathcal{A}$ is an NDD representing the set of vector values $S \subseteq \mathbf{Z}^n$ in basis r, then APPLY-STAR-BASIS$(r, n, \mathcal{A}, \theta)$ is an NDD representing the set $\theta^*(S)$ in basis r.*

Proof The algorithm in Figure 8.14 is a direct implementation of the computation performed in the proof of Theorem 8.52. □

Theorem 8.55 *Let $n \in \mathbf{N}_0$ and θ be the linear operation ($P\vec{x} \leq \vec{q} \rightarrow \vec{x} := A\vec{x} + \vec{b}$), with $m \in \mathbf{N}$, $P \in \mathbf{Z}^{m \times n}$, $q \in \mathbf{Z}^m$, $A \in \mathbf{Z}^{n \times n}$ and $\vec{b} \in \mathbf{Z}^n$ such that META-PRESBURGER?$(n, \theta) = \mathbf{T}$. If $\mathcal{A}$ is an NDD representing the Presburger-definable set of vector values $S \subseteq \mathbf{Z}^n$ in some basis $r > 1$, then APPLY-STAR-PRESBURGER$(n, \mathcal{A}, \theta)$ is an NDD representing the Presburger-definable set $\theta^*(S)$ in basis r.*

Proof The algorithm in Figure 8.15 is a direct implementation of the computation performed in the proof of Theorem 8.52. □

8.3.8 Proofs of Auxiliary Results

This section contains the proofs that were omitted from Sections 8.3.3 and 8.3.4 for clarity. They are presented according to their order of appearance in the main text.

Theorem 8.23 *Let $n, r \in \mathbf{N}_0$ with $r > 1$. A set $S \subseteq \mathbf{Z}^n$ is r-definable if and only if it is r-recognizable.*

Proof

- *If S is r-definable, then S is r-recognizable.* If S is r-definable, then there exist $m \in \mathbf{N}_0$, $S' \subseteq \mathbf{Z}^m$ and $U \in \mathbf{C}^{n \times m}$ such that S' is r-recognizable and $S = US'$. Let $B \subset \mathbf{Z}^m$ be a finite generator of S', i.e., a finite subset of S' such that

function APPLY-STAR-BASIS(basis r, dimension n, NDD $\mathcal{A}$,
linear operation $(P\vec{x} \leq \vec{q} \rightarrow \vec{x} := A\vec{x} + \vec{b})$) : NDD

1: **var** m, p, j : integers;
2: θ' : guardless linear operation;
3: $\mathcal{A}', \mathcal{A}'', \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4$: NDDs;
4: **begin**
5: $(\mathbf{T}, m, p) :=$ DEFINABLE-CLOSURE?(r, n, A);
6: $\theta' := (\vec{x} := A\vec{x} + \vec{b})$;
7: $\mathcal{A}_1 :=$ NDD$(\{\vec{x} \in \mathbf{Z}^n \mid P\vec{x} \leq \vec{q}\})$;
8: $\mathcal{A}_2 :=$ NDD$(\{\vec{x} \in \mathbf{Z}^n \mid \bigwedge_{0 \leq l < p} P(\theta')^l(\vec{x}) \leq \vec{q}\})$;
9: $\mathcal{A}_3 :=$ NDD$(\{\vec{x} \in \mathbf{Z}^n \mid \bigwedge_{0 \leq l < p} P(\theta')^{l+p}(\vec{x}) \leq \vec{q}\})$;
10: $\mathcal{A}' :=$ NDD$(\emptyset)$;
11: **for** $j := 0$ **to** $p - 1$ **do**
12: **begin**
13: $\mathcal{A}_4 :=$ NDD$\Big(\{\vec{x} \in \mathbf{Z}^n \mid (\exists \vec{x}\,' \in \text{SET}(\mathcal{A}))(\vec{x} = (\theta')^j(\vec{x}\,') \wedge \bigwedge_{0 \leq i < j} [(\theta')^i(\vec{x}\,') \in \text{SET}(\mathcal{A}_1)]\}\Big)$;
14: $\mathcal{A}' := \mathcal{A}' \cup \mathcal{A}_4 \cup$ NDD$((\theta')^p(\text{SET}(\mathcal{A}_4) \cap \text{SET}(\mathcal{A}_2)))$;
15: $\mathcal{A}'' :=$ APPLY-STAR-GUARDLESS-BASIS?$(r, n, \mathcal{A}_2 \cap \mathcal{A}_3 \cap \mathcal{A}_4, A^p, \sum_{0 \leq k < p} A^k\vec{b})$;
16: $\mathcal{A}' := \mathcal{A}' \cup$ NDD$((\theta')^p((\theta')^p(\text{SET}(\mathcal{A}'')) \cap \text{SET}(\mathcal{A}_2)))$
17: **end**;
18: **return** $\mathcal{A}'$
19: **end**.

Figure 8.14: Image of an NDD by the closure of a linear operation in a given basis.

function APPLY-STAR-PRESBURGER(dimension n, NDD $\mathcal{A}$,
hfill linear operation $(P\vec{x} \leq \vec{q} \rightarrow \vec{x} := A\vec{x} + \vec{b})$) : NDD

1: **var** p, j : integers;
2: θ' : guardless linear operation;
3: $\mathcal{A}', \mathcal{A}'', \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4$: NDD;
4: **begin**
5: $(\mathbf{T}, 0, p) := \text{DEFINABLE-CLOSURE?}(n, A)$;
6: $\theta' := (\vec{x} := A\vec{x} + \vec{b})$;
7: $\mathcal{A}_1 := \text{NDD}(\{\vec{x} \in \mathbf{Z}^n \mid P\vec{x} \leq \vec{q}\})$;
8: $\mathcal{A}_2 := \text{NDD}(\{\vec{x} \in \mathbf{Z}^n \mid \bigwedge_{0 \leq l < p} P(\theta')^l(\vec{x}) \leq \vec{q}\})$;
9: $\mathcal{A}_3 := \text{NDD}(\{\vec{x} \in \mathbf{Z}^n \mid \bigwedge_{0 \leq l < p} P(\theta')^{l+p}(\vec{x}) \leq \vec{q}\})$;
10: $\mathcal{A}' := \text{NDD}(\emptyset)$;
11: **for** $j := 0$ **to** $p - 1$ **do**
12: **begin**
13: $\mathcal{A}_4 := \text{NDD}\Big(\{\vec{x} \in \mathbf{Z}^n \mid (\exists \vec{x}\,' \in \text{SET}(\mathcal{A}))(\vec{x} = (\theta')^j(\vec{x}\,') \wedge \bigwedge_{0 \leq i < j} [(\theta')^i(\vec{x}\,') \in \text{SET}(\mathcal{A}_1)]\}\Big)$;
14: $\mathcal{A}' := \mathcal{A}' \cup \mathcal{A}_4 \cup \text{NDD}((\theta')^p(\text{SET}(\mathcal{A}_4) \cap \text{SET}(\mathcal{A}_2)))$;
15: $\mathcal{A}'' := \text{APPLY-STAR-GUARDLESS-PRESBURGER}(n, \mathcal{A}_2 \cap \mathcal{A}_3 \cap \mathcal{A}_4, A^p, \sum_{0 \leq k < p} A^k\vec{b})$;
16: $\mathcal{A}' := \mathcal{A}' \cup \text{NDD}((\theta')^p((\theta')^p(\text{SET}(\mathcal{A}'')) \cap \text{SET}(\mathcal{A}_2)))$
17: **end**;
18: **return** $\mathcal{A}'$
19: **end**.

Figure 8.15: Image of an NDD by the closure of a linear operation in any basis.

each vector value in S' is a linear combination of vector values in B. There exists $a \in \mathbf{N}_0$ such that every vector value in S' is a linear combination with integer coefficients of vector values in $(1/a)B$. Let p be the number of vector values in B, and $T \in \mathbf{Q}^{m \times p}$ be a matrix such that $\mathrm{col}(T) = (1/a)B$. Since S' is r-recognizable, the set

$$S'' = \{\vec{x} \in \mathbf{Z}^p \mid T\vec{x} \in S'\}$$

is r-recognizable as well. We have $S' = TS''$, hence $S = (UT)S''$. Every column $\vec{c}$ of T belongs to $(1/a)S'$, and thus is such that $U\vec{c}$ belongs to $(1/a)S$. It follows that $UT \in \mathbf{Q}^{n \times p}$, and therefore the equation $S = (UT)S''$ leads to a definition of S in the first-order theory $\langle \mathbf{Z}, \leq, +, V_r \rangle$ (recall that S'' is r-recognizable). It follows that S is r-recognizable.

- *If S is r-recognizable, then S is r-definable.* Let $U = I_n$ and $S' = S$. We have $S = US'$, where S' is a r-recognizable subset of $\mathbf{Z}^n$, hence S is r-definable.

□

Before proving Theorem 8.24, we introduce the following lemma.

Lemma 8.56 *Let $r \in \mathbf{N}$ with $r > 1$, $n, m_1, m_2 \in \mathbf{N}_0$, $U_1 \in \mathbf{C}^{n \times m_1}$, and $U_2 \in \mathbf{C}^{n \times m_2}$. The set*

$$\left\{ \begin{bmatrix} \vec{x}_1 \\ \vec{x}_2 \end{bmatrix} \in \mathbf{Z}^{m_1+m_2} \mid U_1\vec{x}_1 = U_2\vec{x}_2 \right\}$$

is r-definable.

Proof It is sufficient to prove that for any $m \in \mathbf{N}_0$ and $\vec{u} \in \mathbf{C}^m$, the set S of all the vector values $\vec{x} \in \mathbf{Z}^m$ satisfying $\vec{u} \cdot \vec{x} = 0$ is Presburger-definable. Indeed, applying this result to $m = m_1 + m_2$ and

$$\vec{u} = \begin{bmatrix} \vec{u}_1 \\ -\vec{u}_2 \end{bmatrix},$$

where $\vec{u}_1$ and $\vec{u}_2$ are lines at the same position in U_1 and in U_2, shows that the set of all the vector values

$$\begin{bmatrix} \vec{x}_1 \\ \vec{x}_2 \end{bmatrix} \in \mathbf{Z}^{m_1+m_2}$$

such that $\vec{u_1} \cdot \vec{x}_1 = \vec{u_2} \cdot \vec{x}_2$ is Presburger-definable. The intersection of the sets obtained for each pair of matching lines in U_1 and U_2 is thus Presburger-definable, and therefore r-definable.

It remains to prove that the set S of all the solutions in $\mathbf{Z}^m$ of $\vec{u} \cdot \vec{x} = 0$ is Presburger-definable. This set is an additive subgroup of $\mathbf{R}^m$. An additive subgroup of $\mathbf{R}^m$ is finitely generable if and only if it is discrete (Theorem 6.1 in [ST79]). Since

$S \subseteq \mathbf{Z}^m$, S is discrete and thus finitely generable. Let $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_p$ be the generators of S. We have

$$S = \{a_1\vec{v}_1 + \cdots + a_p\vec{v}_p \mid a_1, \ldots, a_p \in \mathbf{Z}\}.$$

This expression can be rewritten as

$$S = \{\vec{x} \in \mathbf{Z}^m \mid (\exists a_1, \ldots, a_p \in \mathbf{Z})(\vec{x} = a_1\vec{v}_1 + \cdots + a_p\vec{v}_p)\},$$

which is a formula of Presburger arithmetic defining S. □

Theorem 8.24 *Let $r \in \mathbf{N}$ with $r > 1$, $n_1, n_2 \in \mathbf{N}_0$, $S_1 \subseteq \mathbf{C}^{n_1}$, $S_2 \subseteq \mathbf{C}^{n_2}$ such that S_1 and S_2 are r-definable, $\vec{v} \in \mathbf{C}^{n_1}$, $p, q \in \mathbf{N}_0$, and $T \in \mathbf{C}^{p \times n_1}$. The following sets are r-definable:*

- *Any finite subset of $\mathbf{C}^{n_1}$,*
- $S_1 + \vec{v}$,
- TS_1,
- $S_1 \cup S_2$, *provided that* $n_1 = n_2$,
- $S_1 \cap S_2$, *provided that* $n_1 = n_2$,
- $S_1 \times S_2$,
- $\left\{ \begin{bmatrix} \vec{x} \\ \Re(\vec{x}) \\ \Im(\vec{x}) \end{bmatrix} \mid \vec{x} \in S_1 \right\}$,
- $\mathrm{expand}(S_1, r^q) = \{r^{qk}\vec{x} \mid \vec{x} \in S_1 \wedge k \in \mathbf{N}\}$.

Proof

- *Any finite subset of $\mathbf{C}^{n_1}$ is r-definable.* Let $S_1 = \{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_m\}$. Defining $U = [\vec{v}_1; \ldots; \vec{v}_m]$ and $S' = \{\vec{e}_1, \vec{e}_2, \ldots, \vec{e}_n\}$, we obtain $S_1 = US'$, where $S' \subseteq \mathbf{Z}^n$ is r-definable. It follows that S_1 is r-definable.

- *$S_1 + \vec{v}$ is r-definable.* There exist $m \in \mathbf{N}_0$, $U \in \mathbf{C}^{n \times m}$ and $S' \subseteq \mathbf{Z}^m$ such that S' is r-definable and $S_1 = US'$. Since $S_1 + \vec{v} = [U; \vec{v}]\,(S' \times \{1\})$, the set $S_1 + \vec{v}$ is r-definable.

- *TS_1 is r-definable.* There exist $m \in \mathbf{N}_0$, $U \in \mathbf{C}^{n \times m}$ and $S' \subseteq \mathbf{Z}^m$ such that S' is r-definable and $S_1 = US'$. Since $TS_1 = (TU)S'$, the set TS_1 is r-definable.

- *$S_1 \cup S_2$ is r-definable.* There exist $m_1, m_2 \in \mathbf{N}_0$, $U_1 \in \mathbf{C}^{n \times m_1}$, $U_2 \in \mathbf{C}^{n \times m_2}$, $S'_1 \subseteq \mathbf{Z}^{m_1}$ and $S'_2 \subseteq \mathbf{Z}^{m_2}$ such that S'_1 and S'_2 are r-definable, $S_1 = U_1S'_1$, and $S_2 = U_2S'_2$. Since $S_1 \cup S_2 = [U_1; U_2]\,((S'_1 \times (0)^{m_2}) \cup ((0)^{m_1} \times S'_2))$, the set $S_1 \cup S_2$ is r-definable.

- $S_1 \cap S_2$ *is r-definable.* There exist $m_1, m_2 \in \mathbf{N}_0$, $U_1 \in \mathbf{C}^{n\times m_1}$, $U_2 \in \mathbf{C}^{n\times m_2}$, $S_1' \subseteq \mathbf{Z}^{m_1}$ and $S_2' \subseteq \mathbf{Z}^{m_2}$ such that S_1' and S_2' are r-definable, $S_1 = U_1 S_1'$, and $S_2 = U_2 S_2'$. Let $V \in \mathbf{Z}^{m_1+m_2}$ be the set

$$V = \left\{ \begin{bmatrix} \vec{x}_1 \\ \vec{x}_2 \end{bmatrix} \in \mathbf{Z}^{m_1+m_2} \mid U_1\vec{x}_1 = U_2\vec{x}_2 \right\},$$

and S' be the set

$$S' = \{\vec{x}_1 \in \mathbf{Z}^{n_1} \mid \vec{x}_1 \in S_1' \wedge (\exists \vec{x}_2 \in S_2')(\begin{bmatrix} \vec{x}_1 \\ \vec{x}_2 \end{bmatrix} \in V)\}.$$

According to Lemma 8.56, V is r-definable. It follows that the set S' is also r-definable. Since $S_1 \cap S_2 = U_1 S'$, the set $S_1 \cap S_2$ is r-definable.

- $S_1 \times S_2$ *is r-definable.* There exist $m_1, m_2 \in \mathbf{N}_0$, $U_1 \in \mathbf{C}^{n\times m_1}$, $U_2 \in \mathbf{C}^{n\times m_2}$, $S_1' \subseteq \mathbf{Z}^{m_1}$ and $S_2' \subseteq \mathbf{Z}^{m_2}$ such that S_1' and S_2' are r-definable, $S_1 = U_1 S_1'$, and $S_2 = U_2 S_2'$. Since $S_1 \times S_2 = \mathrm{diag}(U_1, U_2)(S_1' \times S_2')$, the set $S_1 \times S_2$ is r-definable.

- $\left\{ \begin{bmatrix} \vec{x} \\ \Re(\vec{x}) \\ \Im(\vec{x}) \end{bmatrix} \mid \vec{x} \in S_1 \right\}$ *is r-definable.* There exist $m \in \mathbf{N}_0$, $U \in \mathbf{C}^{n\times m}$ and $S' \subseteq \mathbf{Z}^m$ such that S' is r-definable and $S_1 = US'$. Since

$$\left\{ \begin{bmatrix} \vec{x} \\ \Re(\vec{x}) \\ \Im(\vec{x}) \end{bmatrix} \mid \vec{x} \in S_1 \right\} = \begin{bmatrix} U \\ \Re(U) \\ \Im(U) \end{bmatrix} S',$$

this set is r-definable.

- $\mathrm{expand}(S_1, r^q)$ *is r-definable.* There exist $m \in \mathbf{N}_0$, $U \in \mathbf{C}^{n\times m}$ and $S' \subseteq \mathbf{Z}^m$ such that S' is r-definable and $S_1 = US'$. Let L be the language of the shortest synchronous encodings in basis r of the vector values in S', expressed over the alphabet $\{0, \ldots, r-1\}^m$. Since S' is r-definable, L is regular[5]. The language $L' = L \cdot ((0^m)^q)^*$ is thus also regular. It follows that the set $S'' \subseteq Z^m$ encoded by L' is r-definable. Since this set obeys

$$S'' = \{r^{qk}\vec{x} \mid \vec{x} \in S' \wedge k \in \mathbf{N}\},$$

we have $US'' = \mathrm{expand}(S_1, r^q)$, from which it follows that $\mathrm{expand}(S_1, r^q)$ is r-definable.

[5]Indeed, this language is denoted by the expression

$$L = E_{S(r)}(S) \setminus \bigcup_{a\in\{0,r-1\}^n} (a \cdot a \cdot \Sigma^*),$$

where $\Sigma = \{0, 1, \ldots, r-1\}^n$.

□

Theorem 8.25 *Let $r \in \mathbf{N}$ with $r > 1$, and $a, b, c \in \mathbf{Z}$ with $a \neq 0$. The set*

$$S = \{ak^2 + bk + c \mid k \in \mathbf{N}\}$$

is not r-definable.

Proof The proof is by contradiction. Suppose that the set $S = \{ak^2 + bk + c \mid k \in \mathbf{N}\}$ is r-definable. This implies that $-S = \{-x \mid x \in S\}$ is r-definable as well. Therefore, we may assume that $a \geq 1$. Let P be the characteristic predicate of S:

$$P(y) \equiv (\exists k \in \mathbf{N})(y = ak^2 + bk + c).$$

Since S is r-definable, P is definable in $\langle \mathbf{Z}, \leq, +, V_r \rangle$. Let $n \in \mathbf{N}$ be greater than $-b/2a$, and $F(x, y)$ be the predicate

$$F(x, y) \equiv y = ax^2 + bx + c \wedge x \geq n.$$

This predicate is definable in $\langle \mathbf{Z}, \leq, +, V_r \rangle$:

$$\begin{aligned} F(x,y) \;\equiv\; & P(y) \wedge P(y + 2ax + a + b) \wedge x \geq n \\ & \wedge (\forall z)(\neg P(z) \vee z \leq y \vee z \geq y + 2ax + a + b). \end{aligned}$$

Indeed, $f(x) = ax^2 + bx + c$ is strictly increasing for $x \geq n$, and the second line of the expression of $F(x, y)$ states that y and $y + 2ax + a + b$ are two consecutive values $f(z)$ and $f(z + 1)$ of the function f. Resolving

$$\left\{ \begin{array}{lcl} y & = & az^2 + bz + c \\ y + 2ax + a + b & = & a(z+1)^2 + b(z+1) + c \end{array} \right.$$

yields $x = z$, hence $y = f(x)$. Now, let $M(x, y, z)$ be the predicate

$$M(x, y, z) \equiv x \geq 0 \wedge y \geq 0 \wedge z = xy.$$

This predicate is definable in $\langle \mathbf{Z}, \leq, +, V_r \rangle$:

$$\begin{aligned} M(x,y,z) \;\equiv\; & (\exists z_1, z_2, z_3, z_4)(F(x + y + n, z_1) \\ & \wedge F(x + n, z_2) \wedge F(y + n, z_3) \wedge F(n, z_4) \\ & \wedge 2az = z_1 - z_2 - z_3 + z_4). \end{aligned} \tag{8.4}$$

Indeed,

$$\begin{aligned} z_1 &= a(x + y + n)^2 + b(x + y + n) + c \\ z_2 &= a(x + n)^2 + b(x + n) + c \\ z_3 &= a(y + n)^2 + b(y + n) + c \\ z_4 &= an^2 + bn + c \end{aligned}$$

implies $z_1 - z_2 - z_3 + z_4 = 2axy$. From Equation (8.4), it follows that the first-order theory $\langle \mathbf{N}, +, . \rangle$ is a subset of the theory $\langle \mathbf{Z}, \leq, +, V_r \rangle$. This is clearly a contradiction, since the latter is decidable (as a consequence of Corollary 8.13) and the former is not [Chu36]. □

In order to prove Theorem 8.26, we need to establish the following result.

Theorem 8.57 *Let $r \in \mathbf{N}$ with $r > 1$, and $p, q \in \mathbf{Z}$ with $p \neq 0$. The set*

$$S = \left\{ \begin{bmatrix} (pj+q)k \\ j \end{bmatrix} \mid j, k \in \mathbf{N} \right\}$$

is not r-definable.

Proof The proof is by contradiction. Suppose that S is r-definable. Let P be the characteristic predicate of S:

$$P(y, x) \equiv (\exists k \in \mathbf{N})(y = k(px + q)).$$

Since S is r-definable, P is definable in $\langle \mathbf{Z}, \leq, +, V_r \rangle$. The predicate $D(y, x)$ over $\mathbf{Z}^2$ which is true if and only if y is different from 0 and is divisible by $px + q$ is straightforwardly defined in terms of P:

$$D(y, x) \equiv y \neq 0 \ \wedge \ (P(y, x) \ \vee \ P(-y, x))$$

For every $x \in \mathbf{Z}$, we have $\gcd(px+q, p(x+1)+q) = \gcd(p, px+q) = \gcd(p, q)$, from which we deduce

$$\operatorname{lcm}(px + q, p(x + 1) + q) = \frac{1}{\gcd(p, q)}(px + q)(p(x + 1) + q).$$

If a number can be divided by two others, then it can be divided by their least common multiple. Therefore, for every y verifying

$$D(y, x) \ \wedge \ D(y, x + 1), \tag{8.5}$$

there exists $k \in \mathbf{Z}$ such that

$$y = \frac{k}{\gcd(p, q)}(px + q)(p(x + 1) + q).$$

Moreover, if we have $x > |q/p| + 1$, then the integer y verifying Equation (8.5) that has the smallest magnitude corresponds to $k = 1$. From this argument, it follows that the predicate

$$\begin{aligned} Q(y, x) \ \equiv \ & x > \left|\frac{q}{p}\right| + 1 \ \wedge \ D(y, x) \ \wedge \ D(y, x + 1) \ \wedge \\ & (\forall z)(|z| \geq |y| \ \vee \ \neg D(z, x) \ \vee \ \neg D(z, x + 1)), \end{aligned}$$

which is definable in $\langle \mathbf{Z}, \leq, +, V_r \rangle$, is such that

$$Q(y,x) \equiv x > \left|\frac{q}{p}\right| + 1 \wedge y = \frac{1}{\gcd(p,q)}(px+q)(p(x+1)+q).$$

Let $l \in \mathbf{N}$ be such that $l > |q/p| + 1$, and $R(y)$ be the predicate

$$R(y) \equiv (\exists x, z)(y = \gcd(p,q).z \wedge x \geq 0 \wedge Q(z, x+l)).$$

This predicate is definable in $\langle \mathbf{Z}, \leq, +, V_r \rangle$, and satisfies

$$R(y) \equiv (\exists k)(k \geq 0 \wedge y = (p(k+l)+q)(p(k+l+1)+q)).$$

It follows that the set

$$\{(p(k+l)+q)(p(k+l+1)+q) \mid k \in \mathbf{N}\}$$

is r-definable, which contradicts Theorem 8.25. □

Theorem 8.26 *Let $r, p \in \mathbf{N}_0$ with $r > 1$, $\lambda \in \mathbf{C}$ such that $\lambda^p = 1$, and $a, b, c, d \in \mathbf{C}$ with $a \notin \mathbf{R} \setminus \mathbf{Q}$. The set*

$$S = \left\{ \lambda^k \begin{bmatrix} (j+a)(k+b)+c \\ j+d \end{bmatrix} \mid j, k \in \mathbf{N} \right\}$$

is not r-definable.

Proof Without loss of generality, we may assume that p is such that $\lambda^i \neq 1$ for every $i \in \{1, 2, \ldots, p-1\}$. The proof is by contradiction. We suppose that S is r-definable. Let us show that this assumption implies that the set

$$S_0 = \left\{ \begin{bmatrix} (j+a)(k+b)+c \\ j+d \end{bmatrix} \mid j, \frac{k}{p} \in \mathbf{N} \right\}$$

is also r-definable. We have

$$(\forall j, k \in \mathbf{N}, 0 \leq k < p)((\exists l \in \mathbf{N})(\lambda^k(j+d) = l+d \wedge l > \lfloor 2|d| \rfloor) \\ \Leftrightarrow k = 0 \wedge j > \lfloor 2|d| \rfloor).$$

Indeed,

- $\lambda^k(j+d) = l+d \wedge l > \lfloor 2|d| \rfloor \Rightarrow |j+d| = |l+d| \wedge l > \lfloor 2|d| \rfloor$
  $\Rightarrow j = l \wedge l > \lfloor 2|d| \rfloor$
  $\Rightarrow \lambda^k(j+d) = j+d \wedge j > \lfloor 2|d| \rfloor$
  $\Rightarrow k = 0 \wedge j > \lfloor 2|d| \rfloor.$

- $$\begin{array}{rcl} k = 0 \wedge j > \lfloor 2|d| \rfloor & \Rightarrow & \lambda^k(j+d) = l + d \wedge j > \lfloor 2|d| \rfloor \\ & \Rightarrow & j + d = l + d \wedge j > \lfloor 2|d| \rfloor \\ & \Rightarrow & \lambda^k(j+d) = l + d \wedge l > \lfloor 2|d| \rfloor. \end{array}$$

It follows that $S_0 = S_{01} \cup S_{02}$, with

$$S_{01} = \left\{ \left[\begin{array}{c} (j+a)(k+b)+c \\ j+d \end{array} \right] \mid j \in \mathbf{N} \wedge 0 \leq j \leq \lfloor 2|d| \rfloor \wedge \frac{k}{p} \in \mathbf{N} \right\}$$

and

$$S_{02} = \left\{ \left[\begin{array}{c} (j+a)(k+b)+c \\ j+d \end{array} \right] \mid j \in \mathbf{N} \wedge \frac{k}{p} \in \mathbf{N} \right.$$
$$\left. \wedge\ (\exists l \in \mathbf{N})(\lambda^k(j+d) = l + d \wedge l > \lfloor 2|d| \rfloor) \right\}.$$

In order to prove that S_0 is r-definable, we show that S_{01} and S_{02} are r-definable.

- *S_{01} is r-definable.* The set S_{01} is a finite union of sets of the form

  $$S_{01j} = \left\{ \left[\begin{array}{c} (j+a)(k+b)+c \\ j+d \end{array} \right] \mid \frac{k}{p} \in \mathbf{N} \right\},$$

  with $j \in \mathbf{N}$. Each of those sets is the image of the set $\{k \mid \frac{k}{p} \in \mathbf{N}\}$ by a linear transformation, and is thus r-definable (Theorem 8.24).

- *S_{02} is r-definable.* We have

  $$\begin{array}{rcl} S_{02} & = & \left\{ \left[\begin{array}{c} x_1 \\ x_2 \end{array} \right] \in S \mid (\exists l \in \mathbf{N})(x_2 = l + d \wedge l > \lfloor 2|d| \rfloor) \right\} \\ & = & S \cap (\pi_1(S) \times \{l + d \mid l \in \mathbf{N} \wedge l > \lfloor 2|d| \rfloor\}), \end{array}$$

  where $\pi_1(S)$ denotes the projection of S over the first vector component. By Theorem 8.24, S_{02} is r-definable.

We have thus proved that S_0 is r-definable. Applying Theorem 8.24, it follows that the following sets are also r-definable:

$$\left\{ \left[\begin{array}{c} (j+a)(k+b)+c \\ j \end{array} \right] \mid j, \frac{k}{p} \in \mathbf{N} \right\},$$

$$\left\{ \left[\begin{array}{c} (j+a)(k+b) - jb - ab \\ j \end{array} \right] \mid j, \frac{k}{p} \in \mathbf{N} \right\},$$

$$\left\{ \left[\begin{array}{c} (j+a)k \\ j \end{array} \right] \mid j, \frac{k}{p} \in \mathbf{N} \right\},$$

$$\left\{ \left[\begin{array}{c} (j+a)k \\ j \end{array} \right] \mid j, k \in \mathbf{N} \right\}.$$

Let us show that the fact that the last set is r-definable leads to a contradiction. There are two possible cases.

- *If* $a \in \mathbf{Q}$. Let $q \in \mathbf{N}$ be such that $qa \in \mathbf{Z}$. The set

$$\left\{ \begin{bmatrix} (qj+qa)k \\ j \end{bmatrix} \mid j,k \in \mathbf{N} \right\}$$

  is r-definable, which contradicts Theorem 8.57.

- *If* $a \in \mathbf{C} \setminus \mathbf{R}$. Applying Theorem 8.24, the following sets are r-definable:

$$\left\{ \begin{bmatrix} (j+a)k \\ \Im((j+a)k) \\ j \end{bmatrix} \mid j,k \in \mathbf{N} \right\} = \left\{ \begin{bmatrix} (j+a)k \\ \Im(a)k \\ j \end{bmatrix} \mid j,k \in \mathbf{N} \right\},$$

$$\left\{ \begin{bmatrix} (j+a)k \\ k \\ j \end{bmatrix} \mid j,k \in \mathbf{N} \right\},$$

$$\left\{ \begin{bmatrix} jk \\ j \end{bmatrix} \mid j,k \in \mathbf{N} \right\}.$$

The fact that the last set is r-definable contradicts Theorem 8.57. □

In order to be able to prove Theorem 8.27, we need an additional lemma.

Lemma 8.58 *Let* $n, r \in \mathbf{N}_0$ *with* $r > 1$, $S \subseteq \mathbf{Z}^n$ *be* r*-definable, and* $\vec{u} \in \mathbf{C}^n$. *If* $\{\vec{u} \cdot \vec{x} \mid \vec{x} \in S\}$ *is infinite, then there exist* $\vec{y}_1, \vec{y}_2 \in \mathbf{Q}^n$ *and* $m \in \mathbf{N}_0$ *such that* $\{\vec{y}_1 + r^{mk}\vec{y}_2 \mid k \in \mathbf{N}\} \subseteq S$ *and* $\vec{u} \cdot \vec{y}_2 \neq 0$.

Proof First, S must be infinite. Since it is r-definable, the language L of the shortest synchronous encodings of its elements in basis r is regular[6]. Hence, there exists a finite-state automaton $\mathcal{A}$ accepting L. Let $|\mathcal{A}|$ denote the number of states of $\mathcal{A}$. Every word $w \in L$ such that $|w| \geq |\mathcal{A}|$ must be accepted by a path of $\mathcal{A}$ that contains at least one cycle, which can be suppressed or further repeated. One can thus decompose w into $w_1 \cdot w_2 \cdot w_3$, with $|w_2| > 0$ and $w_1 \cdot w_2^k \cdot w_3 \in L$ for every $k \in \mathbf{N}$. The language $w_1 \cdot w_2^k \cdot w_3$ encodes a subset S' of S satisfying

$$S' = \{\vec{x}_1 + \sum_{0 \leq i < k} r^{mi}\vec{x}_2 + r^{mk}\vec{x}_3 \mid k \in \mathbf{N}\},$$

with $m = |w_2| \in \mathbf{N}_0$, $\vec{x}_1, \vec{x}_2, \vec{x}_3 \in \mathbf{Z}^n$, and $\vec{x}_2 \neq \vec{0}$. Indeed, $\vec{x}_1$ is the vector encoded by $0^n \cdot w_3$, $\vec{x}_2$ is the vector encoded by $0^n \cdot w_2$ multiplied by $r^{|w_3|}$, and $\vec{x}_3$ is the

[6]Indeed, this language is denoted by the expression

$$L = E_{S(r)}(S) \setminus \bigcup_{a \in \{0,r-1\}^n} (a \cdot a \cdot \Sigma^*),$$

where $\Sigma = \{0, 1, \ldots, r-1\}^n$.

vector encoded by w_1 multiplied by $r^{|w_3|}$. By defining $\vec{y}_1 = \vec{x}_1 - (1/(r^m - 1))\vec{x}_2$ and $\vec{y}_2 = (1/(r^m - 1))\vec{x}_2 + \vec{x}_3$, we obtain

$$S' = \{\vec{y}_1 + r^{mk}\vec{y}_2 \mid k \in \mathbf{N}\},$$

with $\vec{y}_1, \vec{y}_2 \in \mathbf{Q}^n$ and $\vec{y}_2 \neq \vec{0}$. It remains to prove that it is always possible to choose $w \in L$ such that the corresponding $\vec{y}_2$ verifies $\vec{u} \cdot \vec{y}_2 \neq 0$. The proof is by contradiction. Suppose that for every $w \in L$ such that $|w| \geq |\mathcal{A}|$, we obtain $\vec{u} \cdot \vec{y}_2 = 0$. By removing an occurrence of the cycle labeled by w_2 from a path of $\mathcal{A}$ accepting w, we obtain $w' = w_1 \cdot w_3 \in L$. Let $\vec{x}$ and $\vec{x}\,'$ be the elements of S respectively encoded by w and w'. We have $\vec{x} = \vec{y}_1 + r^m\vec{y}_2$ and $\vec{x}\,' = \vec{y}_1 + \vec{y}_2$, and therefore $\vec{u} \cdot \vec{x} = \vec{u} \cdot \vec{x}\,'$. One can thus repeat the same operation so as to remove successively all the occurrences of cycles in w, finally obtaining w'' such that $|w''| < |\mathcal{A}|$. The word w'' encodes $\vec{x}\,'' \in S$, with $\vec{u} \cdot \vec{x} = \vec{u} \cdot \vec{x}\,''$. Since there is only a finite set of w'' such that $|w''| < |\mathcal{A}|$, the set $\{\vec{u} \cdot \vec{x} \mid \vec{x} \in S\}$ is finite, which contradicts an hypothesis of the lemma. □

Theorem 8.27 *Let $r \in \mathbf{N}$ with $r > 1$, $\lambda \in \mathbf{C}$ such that there do not exist $p \in \mathbf{N}_0$ and $m \in \mathbf{N}$ such that $\lambda^p = r^m$. The set*

$$S = \{\lambda^k \mid k \in \mathbf{N}\}$$

is not r-definable.

Proof The proof is by contradiction. Suppose that S is r-definable. There are two possible cases.

- *If S is finite.* Then, there exist $k_1, k_2 \in \mathbf{N}$ such that $k_1 < k_2$ and $\lambda^{k_1} = \lambda^{k_2}$. Choosing $p = k_2 - k_1$ and $m = 0$ leads to a contradiction.

- *If S is infinite.* Since S is r-definable, there exist $n \in \mathbf{N}_0$, $\vec{u} \in \mathbf{C}^n$ and a r-definable set $S' \subseteq \mathbf{Z}^n$ such that $S = \{\vec{u} \cdot \vec{x} \mid \vec{x} \in S'\}$. By Lemma 8.58, there exist $\vec{y}_1$, $\vec{y}_2 \in \mathbf{C}^n$ and $m \in \mathbf{N}_0$ such that

  $$\{\vec{y}_1 + r^{mk}\vec{y}_2 \mid k \in \mathbf{N}\} \subseteq S',$$

  and $\vec{u} \cdot \vec{y}_2 \neq 0$. Let S'' denote the set $\{\vec{y}_1 + r^{mk}\vec{y}_2 \mid k \in \mathbf{N}\}$. Since $S'' \subseteq S'$, we have

  $$\{\vec{u} \cdot \vec{x} \mid \vec{x} \in S''\} \subseteq \{\lambda^k \mid k \in \mathbf{N}\}.$$

  Let $g = \vec{u} \cdot \vec{y}_2$ and $h = \vec{u} \cdot \vec{y}_1$. We have

  $$\{gr^{mk} + h \mid k \in \mathbf{N}\} \subseteq \{\lambda^k \mid k \in \mathbf{N}\},$$

with $g \neq 0$. Since the left-hand side of this equation is an unbounded set, it follows that $|\lambda| > 1$. We have

$$\lim_{k\to\infty} \frac{gr^{m(k+1)} + h}{gr^{mk} + h} = r^m,$$

which gives

$$(\forall \varepsilon \in \mathbf{R}_0^+)(\exists k \in \mathbf{N}) \left(\left| \frac{gr^{m(k+1)} + h}{gr^{mk} + h} - r^m \right| < \varepsilon \right).$$

There must exist $p_1, p_2 \in \mathbf{N}$ with $p_1 < p_2$ such that $gr^{mk} + h = \lambda^{p_1}$ and $gr^{m(k+1)} + h = \lambda^{p_2}$. Therefore, by choosing $p = p_2 - p_1$,

$$(\forall \varepsilon \in \mathbf{R}_0^+)(\exists p \in \mathbf{N})(|\lambda^p - r^m| < \varepsilon),$$

where $\mathbf{R}_0^+$ denotes the set of strictly positive real numbers. Since there can only be a finite number of integers $p \in \mathbf{N}$ such that $|\lambda^p - r^m| < 1$, taking $\varepsilon = 1/2^k$, $k = 1, 2, \ldots$ eventually leads to

$$\lambda^p = r^m$$

for some $p \in \mathbf{N}$. This contradicts an hypothesis of the theorem.

□

Before proving Theorem 8.28, we need to establish two auxiliary results.

Lemma 8.59 *Let $u, v \in \mathbf{R}$ with $u > 1$, $p, q \in \mathbf{N}_0$ with $p > 1$, and $\Pi(x)$ be a polynomial of degree greater than zero with its coefficients in $\mathbf{R}$. We have*

$$\{(u^{pk} + v)^q \mid k \in \mathbf{N}\} \not\subseteq \{u^{k'}\Pi(k') \mid k' \in \mathbf{N}\}.$$

Proof The proof is by contradiction. Suppose that we have

$$\{(u^{pk} + v)^q \mid k \in \mathbf{N}\} \subseteq \{u^{k'}\Pi(k') \mid k' \in \mathbf{N}\}.$$

This is equivalent to

$$(\forall k \in \mathbf{N})(\exists k' \in \mathbf{N})((u^{pk} + v)^q = u^{k'}\Pi(k')). \tag{8.6}$$

For sufficiently large values of k, the left-hand side of this equation is strictly increasing with respect to k. Since Π is a polynomial, that implies that there exists $m > 0$ such that

$$(\forall l_2 > l_1 > m)(\Pi(l_2) > \Pi(l_1) > 0).$$

Let $z = \max_{0 \leq x \leq m} u^x \Pi(x)$, and $n > 0$ be such that $(\forall k \geq n)((u^{pk} + v)^q > z)$. Equation (8.6) associates a unique $k' \in \mathbf{N}$ to every $k \in \mathbf{N}$ such that $k \geq n$. This k' satisfies $k' = l(k)$, where l is a function $\mathbf{R} \to \mathbf{R}$ verifying

$$(\forall x \in \mathbf{R}, x \geq n)((u^{px} + v)^q = u^{l(x)} \Pi(l(x))). \tag{8.7}$$

From this equation, we obtain for $x \geq n$

$$\frac{\mathrm{d}}{\mathrm{d}x}\left((u^{px} + v)^q\right) = \frac{\mathrm{d}}{\mathrm{d}l}\left(u^l \Pi(l)\right) . \frac{\mathrm{d}}{\mathrm{d}x} l(x).$$

The left-hand side and the first factor of the right-hand side of this equation being strictly positive for $x \geq n$ (and thus $l \geq m$), the second factor of the right-hand side is strictly positive as well, from which we deduce that $l(x)$ is strictly increasing for $x \geq n$. Let us compute the derivative $l'(x)$ of $l(x)$ with respect to x. For $x \geq n$, Equation (8.7) gives

$$(u^{px} + v)^q = u^{l(x)} \Pi(l(x)).$$

Taking the natural logarithm of both sides, we obtain

$$q \log(u^{px} + v) = l(x) \log u + \log \Pi(l(x)).$$

Deriving with respect to x, and defining $\Pi'(x) = \mathrm{d}\Pi(x)/\mathrm{d}x$, we get

$$\frac{pq(\log u)u^{px}}{u^{px} + v} = (\log u) l'(x) + \frac{\Pi'(l(x)) l'(x)}{\Pi(l(x))},$$

from which we extract

$$l'(x) = \frac{pq}{1 + \dfrac{v}{u^{px}}} \cdot \frac{1}{1 + \dfrac{1}{\log u} \cdot \dfrac{\Pi'(l(x))}{\Pi(l(x))}}.$$

This result implies that $\lim_{x \to +\infty} l'(x) = pq$, and therefore

$$(\forall \varepsilon > 0)(\exists n' \geq n)(\forall x > n')(pq - \varepsilon < l'(x) < pq + \varepsilon).$$

Let us take $\varepsilon = 1$. According to the previous result, there exists $n' \geq n$ such that

$$(\forall x > n')(pq - 1 < l'(x) < pq + 1). \tag{8.8}$$

For any $k \in \mathbf{N}$ such that $k > n'$, we have

$$l(k+1) = l(k) + \int_k^{k+1} l'(x) \mathrm{d}x,$$

and it follows from Equation (8.8) that

$$pq - 1 < l(k+1) - l(k) < pq + 1.$$

Remark that pq, $l(k)$ and $l(k+1)$ are integer numbers. The only integer number between $pq-1$ and $pq+1$ is pq, hence

$$(\forall k > n')(l(k+1) - l(k) = pq),$$

which gives

$$(\forall k > n')(l(k) = l_0 + pqk),$$

with $l_0 \in \mathbf{Z}$. Replacing $l(k)$ by its value in (8.7), we obtain for any $k > n'$

$$(u^{pk} + v)^q = u^{l_0+pqk}\Pi(l_0 + pqk),$$

hence

$$\Pi(l_0 + pqk) = u^{-l_0}(1 + \frac{v}{u^{pk}})^q.$$

This is clearly impossible, since

$$\lim_{k\to+\infty} \Pi(l_0 + pqk) = +\infty,$$

and

$$\lim_{k\to+\infty} u^{-l_0}(1 + \frac{v}{u^{pk}})^q = u^{-l_0}.$$

□

Theorem 8.60 *Let $r, l, a, b \in \mathbf{N}$ with $r > 1, l > 1$ and $a \geq 1$, such that $r^a = l^b$. If $\Pi(x)$ is a polynomial of degree greater than zero with its coefficients in $\mathbf{Z}$, then the set*

$$S = \{l^k\Pi(k) \mid k \in \mathbf{N}\}$$

is not r-definable.

Proof The proof is by contradiction. Suppose that S is r-definable. After applying Lemma 8.58 with $\vec{u} = (1)$, we obtain that there exist $m \in \mathbf{N}_0$ and $y_1, y_2 \in \mathbf{Q}$ such that $y_2 \neq 0$ and

$$\{y_1 + r^{mk}y_2 \mid k \in \mathbf{N}\} \subseteq S,$$

which can be rewritten as

$$\{(r^m)^k + \frac{y_1}{y_2} \mid k \in \mathbf{N}\} \subseteq \{l^k\Pi(k) \mid k \in \mathbf{N}\}.$$

This result implies

$$\{(r^m)^{ak} + \frac{y_1}{y_2} \mid k \in \mathbf{N}\} \subseteq \{l^k\Pi(k) \mid k \in \mathbf{N}\},$$

and thus, since $l^b = r^a$,

$$\{((r^{am})^k + \frac{y_1}{y_2})^{bm} \mid k \in \mathbf{N}\} \subseteq \{(r^{am})^k\Pi(k) \mid k \in \mathbf{N}\}.$$

Applying Lemma 8.59 to this result directly leads to a contradiction. □

We are now ready to prove Theorem 8.28.

Theorem 8.28 *Let $r, p, m \in \mathbf{N}_0$ with $r > 1$, $\lambda \in \mathbf{C}$ such that $\lambda^p = r^m$, and $a \in \mathbf{C}$ such that $a \notin \mathbf{R} \setminus \mathbf{Q}$. The set*

$$S = \{\lambda^k(k + a) \mid k \in \mathbf{N}\}$$

is not r-definable.

Proof Without loss of generality, we assume that p and m are relatively prime, and that there does not exist $j \in \mathbf{N}_0$ such that $j \geq 2$ and $r^{(1/j)} \in \mathbf{N}$ (thanks to Theorem 8.18). The proof is by contradiction. Suppose that S is r-definable. There are two possible cases, depending on the value of a. For each of them, we will show that our assumption implies that the set

$$S' = \{\lambda^k(k + a) \mid \frac{k}{p} \in \mathbf{N}\}$$

is r-definable, and that this result leads to a contradiction. For each $k \in \mathbf{N}$, we define $y_k = \lambda^k(k + a)$.

- *If $a \in \mathbf{Q}$.* For each $k \in \mathbf{N}$ such that $k > 2|a|$ and p divides k, we have

  $$\Im(y_k) = 0 \wedge \Re(y_k) > |\lambda|^{\lfloor 2|a| \rfloor}(2|a| + a).$$

  Reciprocally, for each $k \in \mathbf{N}$ such that y_k satisfies the previous formula, we have $k > 2|a|$ and p divides k. It follows that we have $S' = S'_1 \cup S'_2$, with

  $$S'_1 = \{\lambda^k(k + a) \mid \frac{k}{p} \in \mathbf{N}, k \leq 2|a|\},$$

  $$S'_2 = \{y_k \in S \mid \Im(y_k) = 0 \wedge \Re(y_k) > l\},$$

  and

  $$l = |\lambda|^{\lfloor 2|a| \rfloor}(2|a| + a).$$

  The set S'_1 is finite, hence it is r-definable (Theorem 8.24). In order to prove that S' is r-definable, it remains to show that S'_2 is r-definable. Let $q \in \mathbf{N}$ be such that $qa \in \mathbf{Z}$. We have

  $$S'_2 = S \cap \frac{1}{q}\{x \in \mathbf{N} \mid x > ql\},$$

  whose r-definability follows from Theorem 8.24. Let us now show that the fact that S' is r-definable leads to a contradiction. We have

  $$\begin{aligned} S' &= \{\lambda^k(k + a) \mid \frac{k}{p} \in \mathbf{N}\} \\ &= \{r^{\left(\frac{mk}{p}\right)}(k + a) \mid \frac{k}{p} \in \mathbf{N}\}. \end{aligned}$$

Theorem 8.24 implies that the set

$$\{r^{\left(\frac{mk}{p}\right)}(qk+qa) \mid \frac{k}{p} \in \mathbf{N}\}$$

is also r-definable, which contradicts Theorem 8.60.

- *If* $a \in \mathbf{C} \setminus \mathbf{R}$. We can assume without loss of generality that $\Im(a) > 0$. Indeed, Theorem 8.24 implies that the set

$$\overline{S} = \{\overline{\lambda}^k(k+\overline{a}) \mid k \in \mathbf{N}\},$$

where for every $z \in \mathbf{C}$, $\overline{z}$ denotes the complex conjugate of z, is r-definable if and only if S is r-definable. Let $N \in \mathbf{N}$ be such that $N > 2|a|$ and $0 < \arg(N+a) < \frac{2\pi}{p}$.

  - For every $k > N$ such that p divides k, we have

$$\lambda^k = r^{\left(\frac{mk}{p}\right)} \Rightarrow \arg(y_k) = \arg(k+a) \Rightarrow 0 < \arg(y_k) < \frac{2\pi}{p}.$$

  - For every $k > N$ such that p does not divide k, we have

$$\arg(y_k) > \frac{2\pi}{p}. \tag{8.9}$$

Let $M \in \mathbf{N}$ be such that $M > N$ and $M > |\lambda|^N|N+a|$, and let $\alpha = \arg(M+a)$. Remark that $0 < \arg(\alpha) < \frac{\pi}{2}$.

  - For every $k > M$ such that p divides k, we have $0 < \arg(y_k) < \alpha \wedge \Im(y_k) > \Im(a)$.
  - For every $k > M$ such that p does not divide k, we have $\arg(y_k) > \alpha$ (according to Inequation (8.9)).
  - For every $k \leq M$, we have $\arg(y_k) \geq \alpha \vee \Im(y_k) \leq \Im(a)$. (Indeed, $0 < \arg(y_k) < \alpha \wedge \Im(y_k) > \Im(a)$ implies $k > M$.)

In summary, we have for each $k \in \mathbf{N}$:

$$k > M \wedge p \text{ divides } k \Leftrightarrow 0 < \arg(y_k) < \alpha \wedge \Im(y_k) > \Im(a).$$

It follows that we have $S' = S'_1 \cup S'_2$, with

$$S'_1 = \{\lambda^k(k+a) \mid \frac{k}{p} \in \mathbf{N}, k \leq M\}$$

and

$$S'_2 = \{y_k \in S \mid 0 < \arg(y_k) < \alpha \wedge \Im(y_k) > \Im(a)\}.$$

The set S'_1 is finite, hence it is r-definable (Theorem 8.24). In order to prove that S' is r-definable, it remains to show that S'_2 is r-definable. Let us consider the transformation $y_k \rightarrow \vec{x}$ such that

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = T \begin{bmatrix} \Re(y_k) \\ \Im(y_k) \end{bmatrix},$$

with

$$T = \begin{bmatrix} 1 & -\frac{\Re(a)}{\Im(a)} \\ 0 & \frac{1}{\Im(a)} \end{bmatrix}.$$

This transformation can be inverted as follows.

$$\begin{bmatrix} \Re(y_k) \\ \Im(y_k) \end{bmatrix} = \begin{bmatrix} 1 & \Re(a) \\ 0 & \Im(a) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$

By Theorem 8.24, the set

$$S''_2 = \left\{ T \begin{bmatrix} \Re(y_k) \\ \Im(y_k) \end{bmatrix} \mid y_k \in S'_2 \right\}$$

is r-definable if and only if S'_2 is r-definable. Remark that every $y_k \in S'_2$ is such that

$$T \begin{bmatrix} \Re(y_k) \\ \Im(y_k) \end{bmatrix} = \begin{bmatrix} \lambda^k k \\ \lambda^k \end{bmatrix} \in \mathbf{N}^2.$$

Let S'' be the set

$$S'' = \left\{ T \begin{bmatrix} \Re(y_k) \\ \Im(y_k) \end{bmatrix} \mid y_k \in S \right\}.$$

We thus have $S'' \subseteq \mathbf{N}^2$. From the previous results, we deduce

$$\begin{aligned} S''_2 &= \left\{ \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbf{N}^2 \mid 0 < \arg(x_1 + \Re(a)x_2 + i\Im(a)x_2) < \alpha \right. \\ &\qquad \left. \wedge\ \Im(a)x_2 > \Im(a) \right\} \cap S'' \\ &= \left\{ \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbf{N}^2 \mid \frac{\Im(a)x_2}{\Re(a)x_2 + x_1} < \frac{\Im(M+a)}{\Re(M+a)} \wedge x_2 > 1 \right\} \cap S'' \\ &= \left\{ \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbf{N}^2 \mid x_2(\Re(a) + M) < x_2\Re(a) + x_1 \wedge x_2 > 1 \right\} \cap S'' \\ &= \left\{ \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbf{N}^2 \mid x_1 > Mx_2 \wedge x_2 > 1 \right\} \cap S''. \end{aligned}$$

This set is r-definable (Theorem 8.24), hence S'_2 and S' are r-definable. Let us now show that the fact that S' is r-definable leads to a contradiction. We

have

$$\begin{aligned} S' &= \{\lambda^k(k+a) \mid \frac{k}{p} \in \mathbf{N}\} \\ &= \{r^{(\frac{mk}{p})}(k+a) \mid \frac{k}{p} \in \mathbf{N}\}. \end{aligned}$$

It follows from Theorem 8.24 that the set

$$\left\{\Re(x) - \frac{\Re(a)}{\Im(a)}\Im(x) \mid x \in S'\right\} = \left\{r^{(\frac{mk}{p})}k \mid \frac{k}{p} \in \mathbf{N}\right\}$$

is r-definable, which contradicts Theorem 8.60.

□

Theorem 8.29 *Let $r, p, m \in \mathbf{N}_0$ with $r > 1$, and $\lambda \in \mathbf{C}$ such that $\lambda^p = r^m$. The set*

$$S = \left\{\begin{bmatrix} k \\ \lambda^k \end{bmatrix} \mid k \in \mathbf{N}\right\}$$

is not r-definable.

Proof The proof is by contradiction. Without loss of generality, we assume that there does not exist $j \in \mathbf{N}_0$ such that $j \geq 2$ and $r^{(1/j)} \in \mathbf{N}$ (thanks to Theorem 8.18). Suppose that S is r-definable. According to Theorem 8.24, the following sets are also r-definable:

$$S \cap \mathbf{N}^2 = \left\{\begin{bmatrix} pk \\ r^{mk} \end{bmatrix} \mid k \in \mathbf{N}\right\},$$

$$S' = \left\{\begin{bmatrix} k \\ r^{mk} \end{bmatrix} \mid k \in \mathbf{N}\right\}.$$

Let L be the language of the shortest synchronous encodings in basis r of the vector values in S', expressed over the alphabet $\{0, 1, \ldots, r-1\}^2$. Since S' is r-definable, L is regular. Let $\mathcal{A}$ be a finite-state automaton accepting L. Any $w \in L$ is of the form

$$w = (0,0) \cdot (0,1) \cdot w_1 \cdot w_2,$$

where $w_1 = (0,0)^{mk - \lfloor \log_r k \rfloor - 1}$, $k \in \mathbf{N}$, and w_2 is such that $(0,0) \cdot w_2$ is the shortest encoding of $k\vec{e}_1$ in basis r. For any sufficiently long word w in L, the path of $\mathcal{A}$ that accepts w must encounter an occurrence of a cycle while reading w_1. This cycle can be further iterated, accepting words that do not belong to L. Hence the contradiction. □

The proof of Theorem 8.30 requires two additional results.

Lemma 8.61 *Let $r, m, p \in \mathbf{N}_0$ with $r > 1$, $\lambda \in \mathbf{C}$ such that $\lambda^p = r^m$, and $a \in \mathbf{C}$. The set*

$$\left\{ \begin{bmatrix} \lambda^k j \\ \lambda^k (j+a) \end{bmatrix} \mid j, k \in \mathbf{N} \right\}$$

is r-definable.

Proof We have

$$S = \bigcup_{0 \leq i < p} \left\{ \lambda^i \begin{bmatrix} \lambda^k j \\ \lambda^k (j+a) \end{bmatrix} \mid j, \frac{k}{p} \in \mathbf{N} \right\}.$$

It is thus sufficient to prove that the set

$$S' = \left\{ \begin{bmatrix} \lambda^k j \\ \lambda^k (j+a) \end{bmatrix} \mid j, \frac{k}{p} \in \mathbf{N} \right\}$$

is r-definable. We have

$$S' = \left\{ r^{mk} \begin{bmatrix} j \\ j+a \end{bmatrix} \mid j, k \in \mathbf{N} \right\}.$$

According to Theorem 8.24, the set

$$S'' = \left\{ \begin{bmatrix} j \\ j+a \end{bmatrix} \mid j \in \mathbf{N} \right\}$$

is r-definable. Since $S' = \text{expand}(S'', r^m)$, it follows from the same theorem that S' is r-definable. □

Theorem 8.62 *Let $r, m \in \mathbf{N}_0$ with $r > 1$, and $p, q \in \mathbf{Z}$ with $p \neq 0$. The set*

$$S = \left\{ \begin{bmatrix} r^{mk}(pj+q) \\ j \end{bmatrix} \mid j, k \in \mathbf{N} \right\}$$

is not r-definable.

Proof The proof is by contradiction. Suppose that S is r-definable. From Theorem 8.24, it follows that the set

$$S' = \left\{ \begin{bmatrix} r^{mk}(pj+q) \\ pj+q \end{bmatrix} \mid j, k \in \mathbf{N} \right\}$$

is also r-definable. Let L be the language of the shortest synchronous encodings in basis r of the vector values in S', expressed as a set of pairs (w_1, w_2) of words of same length over the alphabet $\{0, \ldots, r-1\}^*$. Let f be the function

$$f : \mathbf{Z} \to \mathbf{Z} : x \mapsto \frac{x}{V_r(x)}.$$

Intuitively, $f(x)$ is the number obtained by removing all the trailing "0" digits from the encoding of x in basis r. The value of $f(x)$ stays unchanged when x is multiplied by r. It follows that we have

$$(\forall \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in S')(f(x_1) = f(x_2)). \tag{8.10}$$

For any $l \in \mathbf{N}$, let us define

$$y_l = \begin{cases} p(r^l + 1) + q & \text{if } q = 0, \\ p(r^l) + q & \text{if } q \neq 0. \end{cases}$$

Remark that $V_r(y_l)$ stays bounded with respect to l (in other words, the number of trailing "0" digits of y_l encoded in basis r stays bounded when l increases). Let $n \in \mathbf{N}_0$ be such that $r^n > V_r(y_l)$ for every $l \in \mathbf{N}$, and such that n is greater than the length of the shortest synchronous encodings of p and of q in basis r. Let $\mathcal{A}$ be a finite-state automaton accepting L. There exists $l \in \mathbf{N}$ such that the shortest synchronous encoding of y_l in basis r has more than $|\mathcal{A}| + n$ symbols, where $|\mathcal{A}|$ denotes the number of states of $\mathcal{A}$. Let us take $k \in \mathbf{N}$ such that mk is greater than the length of the shortest synchronous encoding of y_l in basis r. We know that the vector value

$$\begin{bmatrix} r^{mk} y_l \\ y_l \end{bmatrix}$$

belongs to S'. Therefore, its shortest synchronous encoding (w_1, w_2) in basis r belongs to L, and is thus accepted by $\mathcal{A}$. This encoding can be decomposed into $(w_1 \cdot w_1' \cdot w_1'', w_2 \cdot w_2' \cdot w_2'')$, with $|w_1'| = |w_2'| = |\mathcal{A}|$ and $|w_1''| = |w_2''| = n$. It follows that w_1' and w_2' only contain the symbol 0. Any subpath of A accepting (w_1', w_2') must contain a cycle that can be iterated one more time. This allows to transform a path accepting (w_1, w_2) into one accepting a different word (u_1, u_2), from which it follows that $(u_1, u_2) \in L$. By construction, w_1 and u_1 differ only by their number of trailing "0" digits, whereas u_2 and w_2 have the same number of trailing "0" digits and encode different integers. Let x_1 and x_2 be the integers encoded by u_1 and u_2. From the previous results, it follows that $f(x_1) = f(y_l)$ and $f(x_2) \neq f(y_l)$, and therefore that $f(x_1) \neq f(x_2)$. This contradicts Equation (8.10). Hence, S is not r-definable. □

We are now ready to prove Theorem 8.30.

Theorem 8.30 *Let $r, p, m \in \mathbf{N}_0$ with $r > 1$, $\lambda \in \mathbf{C}$ such that $\lambda^p = r^m$, and $a \in \mathbf{C}$. The set*

$$S = \left\{ \begin{bmatrix} \lambda^k (j + a) \\ j \end{bmatrix} \mid j, k \in \mathbf{N} \right\}$$

is not r-definable.

Proof Without loss of generality, we assume that p and m are relatively prime, and that there does not exist $j \in \mathbf{N}_0$ such that $j \geq 2$ and $r^{(1/j)} \in \mathbf{N}$ (thanks to Theorem 8.18). The proof is by contradiction. Suppose that S is r-definable. We distinguish two cases.

- *If $a \in \mathbf{Q}$.* Let $p \in \mathbf{N}_0$ be such that $pa \in \mathbf{Z}$. According to Theorem 8.24, the two following sets are r-definable:

$$\left\{ \begin{bmatrix} \lambda^k(pj + pa) \\ j \end{bmatrix} \mid j, k \in \mathbf{N} \right\},$$

$$\left\{ \begin{bmatrix} \lambda^k(pj + pa) \\ j \end{bmatrix} \mid j, k \in \mathbf{N} \right\} \cap \left\{ \begin{bmatrix} k \operatorname{sgn}(j + a) \\ j \end{bmatrix} \mid j, k \in \mathbf{N} \right\}$$
$$= \left\{ \begin{bmatrix} r^{mk}(pj + pa) \\ j \end{bmatrix} \mid j, k \in \mathbf{N} \right\}.$$

  The fact that the second set is r-definable contradicts Theorem 8.62. It follows that S is not r-definable.

- *If $a \notin \mathbf{Q}$.* For any $k_1, k_2, j_1, j_2 \in \mathbf{N}$, we have

$$\lambda^{k_1}(j_1 + a) = \lambda^{k_2}(j_2 + a) \Leftrightarrow \lambda^{k_1} = \lambda^{k_2} \wedge j_1 = j_2.$$

  The set S being r-definable, Theorem 8.24 and Lemma 8.61 imply that the two following sets are also r-definable:

$$\left\{ \begin{bmatrix} \lambda^k(j + a) \\ \lambda^k j \\ j \end{bmatrix} \mid j, k \in \mathbf{N} \right\},$$

$$\left\{ \begin{bmatrix} \lambda^k j \\ j \end{bmatrix} \mid j, k \in \mathbf{N} \right\}.$$

  The fact that the second set is r-definable contradicts Theorem 8.62. It follows that S is not r-definable.

□

Theorem 8.31 *Let $r, p_1, p_2, m_1, m_2 \in \mathbf{N}_0$ with $r > 1$, $\lambda_1, \lambda_2 \in \mathbf{C}$ such that $\lambda_1^{p_1} = r^{m_1}$, $\lambda_2^{p_2} = r^{m_2}$ and $|\lambda_1| \neq |\lambda_2|$. The set*

$$S = \left\{ \begin{bmatrix} \lambda_1^k \\ \lambda_2^k \end{bmatrix} \mid k \in \mathbf{N} \right\}$$

is not r-definable.

Proof Without loss of generality, we can assume that m_i and p_i are relatively prime for $i \in \{1, 2\}$, that $m_1 < m_2$, and that there does not exist $j \in \mathbf{N}_0$ such that $j \geq 2$ and $r^{(1/j)} \in \mathbf{N}$ (thanks to Theorem 8.18). The proof is by contradiction. Suppose that S is r-definable. Theorem 8.24 implies that the set

$$S' = S \cap \mathbf{N}^2 = \left\{ \begin{bmatrix} r^{m_1 k} \\ r^{m_2 k} \end{bmatrix} \mid k \in \mathbf{N} \right\}$$

is r-definable as well. Let L be the language of the shortest encodings in basis r of the vectors in S, expressed over the alphabet $\{0, 1, \ldots, r-1\}^2$. This language is of the form

$$L = \{(0,0) \cdot (0,1) \cdot (0,0)^{k(m_2 - m_1)-1} \cdot (1,0) \cdot (0,0)^{km_1} \mid k \in \mathbf{N}\}.$$

Since L is not regular, S' is not r-definable. It follows that S is not r-definable either. □

Lemma 8.36 *Let $n, r \in \mathbf{N}_0$ with $n > 1, r > 1$, $\lambda \in \mathbf{C}$ such that $\lambda \neq 1$, $p \in \mathbf{N}_0$, $m \in \mathbf{N}$ such that $\lambda^p = r^m$, $q \in \mathbf{N}$ with $1 < q \leq n$, $V \in \mathbf{C}^{q \times n}$ of rank q, and $\vec{b} \in \mathbf{Z}^n$. There exists a r-definable set $S \subseteq \mathbf{Z}^n$ such that the set*

$$S' = \{J^k_{q,\lambda}\vec{x} + \sum_{0 \leq i < k} J^i_{q,\lambda}\vec{b'} \mid \vec{x} \in VS \wedge k \in \mathbf{N}\},$$

where $\vec{b'} = V\vec{b}$, is not r-definable.

Proof Let us project S' onto the two vector components that have the highest index. We obtain

$$S'' = \left\{ \begin{bmatrix} \lambda^k & k\lambda^{k-1} \\ 0 & \lambda^k \end{bmatrix} \vec{x} + \sum_{0 \leq i < k} \begin{bmatrix} \lambda^i & i\lambda^{i-1} \\ 0 & \lambda^i \end{bmatrix} \vec{b''} \mid \vec{x} \in V'S \wedge k \in \mathbf{N} \right\},$$

where $V' \in \mathbf{C}^{2 \times n}$ is composed of the two last lines of V (and is therefore of rank 2), and $\vec{b''} = V'\vec{b}$. It is sufficient to prove that there exists a r-definable set $S \subseteq \mathbf{Z}^n$ such that the corresponding S'' is not r-definable. Let $\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \vec{b''}$. We distinguish four different situations.

- *If $|\lambda| = 1$ and $b_2 = 0$.* We have

  $$S'' = \left\{ \begin{bmatrix} \lambda^k x_1 + k\lambda^{k-1} x_2 + \frac{\lambda^k - 1}{\lambda - 1} b_1 \\ \lambda^k x_2 \end{bmatrix} \mid \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in V'S \wedge k \in \mathbf{N} \right\}.$$

  Let $\vec{v} \in \mathbf{Z}^n$ be such that the second component of $V'\vec{v}$ is different from zero (such a $\vec{v}$ always exists, otherwise the rank of V' would be less than 2). Choosing $S = \{j\vec{v} \mid j \in \mathbf{N}\}$ yields

  $$S'' = \left\{ \begin{bmatrix} \lambda^k j v'_1 + kj\lambda^{k-1} v'_2 + \frac{\lambda^k - 1}{\lambda - 1} b_1 \\ \lambda^k j v'_2 \end{bmatrix} \mid j, k \in \mathbf{N} \right\},$$

with $\begin{bmatrix} v'_1 \\ v'_2 \end{bmatrix} = V'\vec{v}$. If S'' is r-definable, then by Theorem 8.24 the following sets are also r-definable:

$$\left\{ \begin{bmatrix} kj\lambda^{k-1}v'_2 + \frac{\lambda^k}{\lambda-1}b_1 \\ \lambda^k j \end{bmatrix} \mid j,k \in \mathbf{N} \right\},$$

$$\left\{ \lambda^k \begin{bmatrix} kj + \frac{\lambda}{\lambda-1}\frac{b_1}{v'_2} \\ j \end{bmatrix} \mid j,k \in \mathbf{N} \right\},$$

$$\left\{ \lambda^k \begin{bmatrix} kj + j\frac{\lambda}{\lambda-1}\frac{b_1}{v'_2} + \frac{\lambda}{\lambda-1}\frac{b_1}{v'_2} \\ j \end{bmatrix} \mid j,k \in \mathbf{N} \right\},$$

$$\left\{ \lambda^k \begin{bmatrix} j(k + \frac{\lambda}{\lambda-1}\frac{b_1}{v'_2}) + \frac{\lambda}{\lambda-1}\frac{b_1}{v'_2} \\ j \end{bmatrix} \mid j,k \in \mathbf{N} \right\}.$$

By Theorem 8.26, this last set is not r-definable. It follows that S'' and S' are not r-definable.

- *If* $|\lambda| = 1$ *and* $b_2 \neq 0$. Let us take $S = \{j\vec{b} \mid j \in \mathbf{N}\}$. We obtain

$$S'' = \left\{ \begin{bmatrix} \lambda^k jb_1 + kj\lambda^{k-1}b_2 + \frac{\lambda^k-1}{\lambda-1}b_1 + \frac{(k-1)\lambda^k - k\lambda^{k-1}+1}{(\lambda-1)^2}b_2 \\ \lambda^k jb_2 + \frac{\lambda^k-1}{\lambda-1}b_2 \end{bmatrix} \mid j,k \in \mathbf{N} \right\}$$

If S'' is r-definable, then by Theorem 8.24 the following sets are also r-definable:

$$\left\{ \begin{bmatrix} \lambda^k jb_1 + kj\lambda^{k-1}b_2 + \frac{\lambda^k}{\lambda-1}b_1 + \frac{k\lambda^{k-1}}{\lambda-1}b_2 - \frac{\lambda^k}{(\lambda-1)^2}b_2 \\ \lambda^k jb_2 + \frac{\lambda^k}{\lambda-1}b_2 \end{bmatrix} \mid j,k \in \mathbf{N} \right\},$$

$$\left\{ \lambda^k \begin{bmatrix} jb_1 + \frac{b_2}{\lambda}jk + \frac{1}{\lambda-1}b_1 + \frac{k}{\lambda(\lambda-1)}b_2 - \frac{1}{(\lambda-1)^2}b_2 \\ jb_2 + \frac{1}{\lambda-1}b_2 \end{bmatrix} \mid j,k \in \mathbf{N} \right\},$$

$$\left\{ \lambda^k \begin{bmatrix} jk + \lambda\frac{b_1}{b_2}j + \frac{\lambda}{\lambda-1}\frac{b_1}{b_2} + \frac{k}{\lambda-1} - \frac{\lambda}{(\lambda-1)^2} \\ j + \frac{1}{\lambda-1} \end{bmatrix} \mid j,k \in \mathbf{N} \right\},$$

$$\left\{ \lambda^k \begin{bmatrix} (j + \frac{1}{\lambda-1})(k + \lambda\frac{b_1}{b_2}) - \frac{\lambda}{(\lambda-1)^2} \\ j + \frac{1}{\lambda-1} \end{bmatrix} \mid j,k \in \mathbf{N} \right\}.$$

Since we have $\frac{1}{\lambda-1} \notin \mathbf{R} \setminus \mathbf{Q}$, it follows from Theorem 8.26 that the last set is not r-definable. Therefore, S'' and S' are not r-definable.

- *If* $|\lambda| > 1$ *and* $b_2 = 0$. We have

$$S'' = \left\{ \begin{bmatrix} \lambda^k x_1 + k\lambda^{k-1}x_2 + \frac{\lambda^k-1}{\lambda-1}b_1 \\ \lambda^k x_2 \end{bmatrix} \mid \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in V'S \wedge k \in \mathbf{N} \right\}.$$

Let $\vec{v} \in \mathbf{Z}^n$ be such that the second component of $V'\vec{v}$ is different from zero (such a $\vec{v}$ always exists, otherwise the rank of V' would be less than 2). Choosing $S = \{\vec{v} \mid j \in \mathbf{N}\}$ yields

$$S'' = \left\{ \left[\begin{array}{c} \lambda^k v'_1 + k\lambda^{k-1} v'_2 + \frac{\lambda^k - 1}{\lambda - 1} b_1 \\ \lambda^k v'_2 \end{array} \right] \mid k \in \mathbf{N} \right\},$$

with $\left[\begin{array}{c} v'_1 \\ v'_2 \end{array} \right] = V'\vec{v}$. If S'' is r-definable, then by Theorem 8.24 the following sets are also r-definable:

$$\left\{ \lambda^k \left[\begin{array}{c} k v'_2 + \lambda v'_1 + \frac{1}{\lambda - 1} b_1 \\ v'_2 \end{array} \right] \mid k \in \mathbf{N} \right\},$$

$$\{\lambda^k k \mid k \in \mathbf{N}\}.$$

According to Theorem 8.28, the last set is not r-definable. It follows that S'' and S' are not r-definable.

- *If* $|\lambda| > 1$ *and* $b_2 \neq 0$. Let us take $S = \{\vec{b}\}$. We obtain

$$S'' = \left\{ \left[\begin{array}{c} \lambda^k b_1 + \lambda^{k-1} k b_2 + \frac{\lambda^k - 1}{\lambda - 1} b_1 + \frac{(k-1)\lambda^k - k\lambda^{k-1} + 1}{(\lambda - 1)^2} b_2 \\ \lambda^k b_2 + \frac{\lambda^k - 1}{\lambda - 1} b_2 \end{array} \right] \mid k \in \mathbf{N} \right\}.$$

If S'' is r-definable, then by Theorem 8.24 the following sets are also r-definable:

$$\left\{ \lambda^k \left[\begin{array}{c} b_1 + \frac{k}{\lambda} b_2 + \frac{1}{\lambda - 1} b_1 + \frac{k}{\lambda(\lambda - 1)} b_2 - \frac{1}{(\lambda - 1)^2} b_2 \\ b_2 + \frac{1}{\lambda - 1} b_2 \end{array} \right] \mid k \in \mathbf{N} \right\},$$

$$\{\lambda^k k \mid k \in \mathbf{N}\}.$$

(Remark that $|\lambda| > 1$ implies $1 + \frac{1}{\lambda - 1} \neq 0$.) According to Theorem 8.28, the last set is not r-definable. It follows that S'' and S' are not r-definable.

□

Lemma 8.37 *Let* $n, r \in \mathbf{N}_0$ *with* $n > 1, r > 1$, $q \in \mathbf{N}$ *with* $1 < q \leq n$, $V \in \mathbf{Q}^{q \times n}$ *of rank* q, *and* $\vec{b} \in \mathbf{Z}^n$. *There exists a* r*-definable set* $S \subseteq \mathbf{Z}^n$ *such that the set*

$$S' = \{J^k_{q,1}\vec{x} + \sum_{0 \leq i < k} J^i_{q,1}\vec{b'} \mid \vec{x} \in VS \wedge k \in \mathbf{N}\},$$

where $\vec{b'} = V\vec{b}$, *is not* r*-definable.*

Proof Let us project S' onto the two vector components that have the highest index. We obtain

$$S'' = \left\{ \begin{bmatrix} 1 & k \\ 0 & 1 \end{bmatrix} \vec{x} + \sum_{0 \leq i < k} \begin{bmatrix} 1 & i \\ 0 & 1 \end{bmatrix} \vec{b}'' \mid \vec{x} \in V'S \wedge k \in \mathbf{N} \right\},$$

where $V' \in \mathbf{Q}^{2 \times n}$ is composed of the two last rows of V (and is therefore of rank 2), and $\vec{b}'' = V'\vec{b}$. It is sufficient to prove that there exists a r-definable $S \subseteq \mathbf{Z}^n$ such that the corresponding S'' is not r-definable. Let $\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \vec{b}''$. We distinguish two different situations.

- *If* $b_2 = 0$. We have

  $$S'' = \left\{ \begin{bmatrix} x_1 + kx_2 + kb_1 \\ x_2 \end{bmatrix} \mid \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in V'S \wedge k \in \mathbf{N} \right\}.$$

  Let $\vec{v} \in \mathbf{Z}^n$ be such that the second component of $V'\vec{v}$ is different from zero (such a $\vec{v}$ always exists, otherwise the rank of V' would be less than 2). Choosing $S = \{j\vec{v} \mid j \in \mathbf{N}\}$ yields

  $$S'' = \left\{ \begin{bmatrix} jv'_1 + jkv'_2 + kb_1 \\ jv'_2 \end{bmatrix} \mid j, k \in \mathbf{N} \right\},$$

  with $\begin{bmatrix} v'_1 \\ v'_2 \end{bmatrix} = V'\vec{v}$. If S'' is r-definable, then by Theorem 8.24 the following set is also r-definable:

  $$\left\{ \begin{bmatrix} jk + \frac{b_1}{v'_2}k \\ j \end{bmatrix} \mid j, k \in \mathbf{N} \right\}.$$

  Since $\frac{b_1}{v'_2} \in \mathbf{Q}$ (because $\vec{v} \in \mathbf{Z}^n$ and $V' \in \mathbf{Q}^{2 \times n}$), Theorem 8.26 implies that this set is not r-definable. It follows that S'' and S' are not r-definable.

- *If* $b_2 \neq 0$. We have

  $$S'' = \left\{ \begin{bmatrix} x_1 + kx_2 + kb_1 + \frac{1}{2}k(k-1)b_2 \\ x_1 + kb_2 \end{bmatrix} \mid \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in VS' \wedge k \in \mathbf{N} \right\}.$$

  Let $S = \{\vec{0}\}$. We obtain

  $$S'' = \left\{ \begin{bmatrix} kb_1 + \frac{1}{2}k(k-1)b_2 \\ kb_2 \end{bmatrix} \mid k \in \mathbf{N} \right\}.$$

  If S'' is r-definable, then by Theorem 8.24 the following sets are also r-definable:

  $$\left\{ \begin{bmatrix} \frac{1}{2}k(k-1)b_2 \\ k \end{bmatrix} \mid k \in \mathbf{N} \right\},$$

$$\{k(k-1) \mid k \in \mathbf{N}\}.$$

By Theorem 8.25, this last set is not r-definable. It follows that S'' and S' are not r-definable. □

Lemma 8.38 *Let $n \in \mathbf{N}_0$ and $A \in \mathbf{Z}^{n \times n}$. There exists a nonsingular matrix $U \in \mathbf{C}^{n \times n}$ transforming A into its Jordan form A_J, and such that every row of U^{-1} at the same position as a line of a Jordan block $J_{q,\lambda}$ in A_J contains only rational components provided that λ is rational.*

Proof In order for U to transform A into A_J, we must have $A_J = U^{-1}AU$. Let J be a Jordan block in A_J associated to a rational eigenvalue. Without loss of generality, we may assume that J is the first block of A_J. We have $A_J U^{-1} = U^{-1}A$, which can be decomposed into

$$\begin{bmatrix} J & 0 \\ 0 & X \end{bmatrix} \begin{bmatrix} U_1 & U_2 \\ U_3 & U_4 \end{bmatrix} = \begin{bmatrix} U_1 & U_2 \\ U_3 & U_4 \end{bmatrix} A,$$

where $U_1, \ldots, U_4$ are parts of U^{-1} of appropriate sizes. This linear system can be split into the two equations

$$J\,[U_1; U_2] = [U_1; U_2]\,A \tag{8.11}$$

and

$$X\,[U_3; U_4] = [U_3; U_4]\,A.$$

If U exists, replacing $[U_1; U_2]$ by any solution of (8.11) whose lines are linearly independent from each other and from the lines of $[U_3; U_4]$ yields a matrix transforming A into A_J. Since all the coefficients of Equation (8.11) belong to $\mathbf{Q}$, it is always possible to find a suitable rational solution. □

8.4 Creation of Multicycle Meta-Transitions

The problem addressed in this section is to design the algorithms that are needed in order to associate multicycle meta-transitions to systems using integer variables. As it has been shown in Section 3.4.2, the creation of multicycle meta-transitions is governed by a computable function MULTI-META-SET that takes as arguments a finite number of linear operations, and returns a finite number of memory functions corresponding to multicycle meta-transitions that can be associated to the cycles labeled by those linear operations.

The problem that consists of deciding whether the closure of a finite set of linear operations preserves the r-definable nature of sets in a given basis $r > 0$ is very tough. To the best of our knowledge, it is presently not known whether this

problem is decidable or not. A weaker problem, equivalent to deciding whether the closure of a finite set of linear operations preserves the Presburger-definable nature of sets, has been successfully solved by Hauschildt [Hau90]. The proof of this result is constructive, and can be turned into an algorithm for computing an NDD representing the image of a set of vector values represented as an NDD by the closure of a finite set of linear operations that preserves the Presburger-definable nature of sets. Although very elegant, Hauschildt's result is described in 150 pages, and its presentation is far beyond the scope of this thesis. A smaller but more intricate decision procedure has also been developed independently by Lambert [Lam94].

The implementations of MULTI-META-SET that we provide here simply return the set of all the cycle meta-transitions that can be created from the given set of linear operations. The algorithms developed with respect to a given basis $r > 1$ and to any basis are given in Figures 8.16 and 8.17. The algorithms for computing the image of a set of vector values represented as an NDD by cycle meta-transitions can be found in Section 8.3.6.

8.5 Model Checking

This section is aimed at providing algorithms for applying to linear operations the functions ITERABLE and MULTI-ITERABLE required by the model-checking algorithms introduced in Chapter 4. In the present context, the purpose of ITERABLE is to determine, given a linear operation θ, a representation of the set of integer vector values to which it is known that θ can be applied infinitely many times.

We only consider linear operations that satisfy the conditions expressed by Theorem 8.40, i.e., the linear operations for which it has been established that their closure preserves the definable nature of sets of vector values. The motivation of this restriction is twofold. First, it simplifies the computations, by allowing to exploit the results established in Section 8.3. Second, it does not influence the model-checking algorithms, since they only apply ITERABLE to operations that can be associated to cycle meta-transitions.

Let $n \in \mathbf{N}$ be a dimension, $r \in \mathbf{N}$ with $r > 1$ be a basis, and $\theta = (P\vec{x} \leq \vec{q} \rightarrow \vec{x} := A\vec{x} + \vec{b})$, where $m \in \mathbf{N}$, $P \in \mathbf{Z}^{m \times n}$, $\vec{q} \in \mathbf{Z}^m$, $A \in \mathbf{Z}^{n \times n}$ and $\vec{b} \in \mathbf{Z}^n$ be a linear operation whose transformation matrix A satisfies the hypotheses of Theorem 8.40. For every vector value $\vec{v} \in \mathbf{Z}^n$, the guardless linear operation $\theta' = (\vec{x} := A\vec{x} + \vec{b})$ can be applied infinitely many times to $\vec{v}$, producing the set of values $(\theta')^*(\vec{v})$. The problem is thus reduced to computing the set of all the $\vec{v}$ for which $P(\theta')^k(\vec{v}) \leq \vec{q}$ for every $k \in \mathbf{N}$.

By hypothesis, there exist $m \in \mathbf{N}$ and $p \in \mathbf{N}_0$ such that all the eigenvalues of A^p belong to $\{0, r^m\}$. We distinguish two situations.

function MULTI-META-SET-BASIS(basis r, dimension n, **set of** linear operations S) : **set of** functions;

1: **var** T : **set of** functions;
2: **begin**
3: $T := \{(P\vec{x} \leq \vec{q} \rightarrow \vec{x} := A\vec{x} + \vec{b})^* \mid (P\vec{x} \leq \vec{q} \rightarrow \vec{x} := A\vec{x} + \vec{b}) \in S \wedge \text{META-BASIS?}(r, n, (\vec{x} := A\vec{x} + \vec{b})) = \mathbf{T}\}$;
4: **return** T
5: **end**.

Figure 8.16: Creation of multicycle meta-transitions in a given basis.

function MULTI-META-SET-PRESBURGER(dimension n, **set of** linear operations S) : **set of** functions;

1: **var** T : **set of** functions;
2: **begin**
3: $T := \{(P\vec{x} \leq \vec{q} \rightarrow \vec{x} := A\vec{x} + \vec{b})^* \mid (P\vec{x} \leq \vec{q} \rightarrow \vec{x} := A\vec{x} + \vec{b}) \in S \wedge \text{META-PRESBURGER?}(n, (\vec{x} := A\vec{x} + \vec{b})) = \mathbf{T}\}$;
4: **return** T
5: **end**.

Figure 8.17: Creation of multicycle meta-transitions in any basis.

- *If* $m = 0$. For any $\vec{v} \in \mathbf{Z}^n$, we have

$$(\theta')^*(\vec{v}) = \bigcup_{k \in \mathbf{N}} (\theta')^{kp} \left(\bigcup_{0 \leq j < p} (\theta')^j(\vec{v}) \right).$$

According to the proof of Theorem 8.43, this expression can be rewritten as

$$\begin{aligned}(\theta')^*(\vec{v}) &= \bigcup_{0 \leq j < p} (\theta')^j(\vec{v}) \cup \bigcup_{k \in \mathbf{N}} \left(A^p \left(\bigcup_{0 \leq j < p} (\theta')^j(\vec{v}) \right) + kA^p\vec{b'} + \vec{b'} \right) \\ &= \bigcup_{0 \leq j < p} \left((\theta')^j(\vec{v}) \cup \bigcup_{k \in \mathbf{N}} \left(A^p(\theta')^j(\vec{v}) + kA^p\vec{b'} + \vec{b'} \right) \right),\end{aligned}$$

with $\vec{b'} = \sum_{0 \leq i < p} A^i\vec{b}$. We thus have

$$\text{ITERABLE}(\theta) = \bigcap_{0 \leq j < p} \{\vec{v} \in \mathbf{Z}^n \mid P(\theta')^j(\vec{v}) \leq \vec{q} \wedge \varphi(A^p(\theta')^j(\vec{v}))\},$$

where φ is the predicate

$$\varphi : \mathbf{Z}^n \to \{\mathbf{T}, \mathbf{F}\} : \vec{x} \mapsto (\forall k \in \mathbf{N})(P(\vec{x} + kA^p\vec{b'} + \vec{b'}) \leq \vec{q}).$$

The previous formula belongs to the first-order theory $\langle \mathbf{Z}, \leq, + \rangle$ and can therefore be translated into an algorithm for constructing an NDD representing the set $\{\vec{x} \in \mathbf{Z}^n \mid \varphi(\vec{x})\}$ (and thus for constructing an NDD representing the set ITERABLE(θ)).

It is however possible to simplify the expression of φ. Since the region $\{\vec{x} \in \mathbf{Z}^n \mid P\vec{x} \leq \vec{q}\}$ is convex, we have for every $x \in \mathbf{Z}^n$

$$\varphi(\vec{x}) \equiv \begin{cases} P(\vec{x} + \vec{b'}) \leq \vec{q} & \text{if } PA^p\vec{b'} \leq \vec{0}, \\ \mathbf{F} & \text{if } PA^p\vec{b'} \not\leq \vec{0}. \end{cases}$$

- *If* $m > 0$. For any $\vec{v} \in \mathbf{Z}^n$, we have

$$(\theta')^*(\vec{v}) = \bigcup_{k \in \mathbf{N}} (\theta')^{kp} \left(\bigcup_{0 \leq j < p} (\theta')^j(\vec{v}) \right).$$

According to the proof of Theorem 8.43, this expression can be rewritten as

$$\begin{aligned}(\theta')^*(\vec{v}) &= \bigcup_{0 \leq j < p} (\theta')^j(\vec{v}) \cup \bigcup_{k \in \mathbf{N}} \left(\frac{1}{r^m - 1} \left[r^{mk} \left((r^m - 1)A^p \Big(\bigcup_{0 \leq j < p} (\theta')^j(\vec{v}) \Big) \right.\right.\right. \\ &\qquad \left.\left.\left. + A^p\vec{b'} \right) - A^p\vec{b'} \right] + \vec{b'} \right) \\ &= \bigcup_{0 \leq j < p} \left((\theta')^j(\vec{v}) \cup \bigcup_{k \in \mathbf{N}} \left(\frac{1}{r^m - 1} \left[r^{mk} \left((r^m - 1)A^p(\theta')^j(\vec{v}) \right.\right.\right.\right. \\ &\qquad \left.\left.\left.\left. + A^p\vec{b'} \right) - A^p\vec{b'} \right] + \vec{b'} \right) \right),\end{aligned}$$

with $\vec{b'} = \sum_{0 \leq i < p} A^i \vec{b}$. We thus have

$$\text{ITERABLE}(\theta) = \bigcap_{0 \leq j < p} \{\vec{v} \in \mathbf{Z}^n \mid P(\theta')^j(\vec{v}) \leq \vec{q} \wedge \varphi'(A^p(\theta')^j(\vec{v}))\},$$

where φ' is the predicate

$$\varphi' : \mathbf{Z}^n \to \{\mathbf{T}, \mathbf{F}\} : \vec{x} \mapsto (\forall k \in \mathbf{N})\left(P\left(\frac{1}{r^m - 1}\left[r^{mk}\left((r^m - 1)\vec{x} + A^p\vec{b'}\right) - A^p\vec{b'}\right] + \vec{b'}\right) \leq \vec{q}\right).$$

Like in the previous case, it is possible to simplify the expression of φ'. Since the region $\{\vec{x} \in \mathbf{Z}^n \mid P\vec{x} \leq \vec{q}\}$ is convex, we have for every $x \in \mathbf{Z}^n$

$$\begin{aligned}\varphi'(\vec{x}) &\equiv (\forall k' \in \mathbf{N}_0)\left(P\left(\frac{1}{r^m - 1}\left[k'\left((r^m - 1)\vec{x} + A^p\vec{b'}\right) - A^p\vec{b'}\right] + \vec{b'}\right) \leq \vec{q}\right) \\ &\equiv P(\vec{x} + \vec{b'}) \leq \vec{q} \wedge P\left(\vec{x} + \frac{1}{r^m - 1}A^p\vec{b'}\right) \leq \vec{0}.\end{aligned}$$

Algorithms formalizing the computation of an NDD representing ITERABLE(θ) with respect to a given basis $r > 1$ and to any basis are given in Figures 8.18 and 8.19.

Theorem 8.39 *Let $n \in \mathbf{N}$ be a dimension, $r \in \mathbf{N}$ with $r > 1$ be a basis, and $\theta = (P\vec{x} \leq \vec{q} \to \vec{x} := A\vec{x} + \vec{b})$, where $m \in \mathbf{N}$, $P \in \mathbf{Z}^{m \times n}$, $\vec{q} \in \mathbf{Z}^m$, $A \in \mathbf{Z}^{n \times n}$ and $\vec{b} \in \mathbf{Z}^n$, be a linear operation such that META-BASIS?$(r, n, A) = \mathbf{T}$. ITERABLE-BASIS(r, n, θ) is an NDD representing in basis r the set of all the vector values $\vec{v} \in \mathbf{Z}^n$ to which θ can be applied an infinite number of times.*

Proof The algorithm in Figure 8.18 is a direct implementation of the computation method discussed in this section. □

Theorem 8.40 *Let $n \in \mathbf{N}$ be a dimension and $\theta = (P\vec{x} \leq \vec{q} \to \vec{x} := A\vec{x} + \vec{b})$, where $m \in \mathbf{N}$, $P \in \mathbf{Z}^{m \times n}$, $\vec{q} \in \mathbf{Z}^m$, $A \in \mathbf{Z}^{n \times n}$ and $\vec{b} \in \mathbf{Z}^n$, be a linear operation such that META-PRESBURGER?$(n, A) = \mathbf{T}$. In any basis $r > 1$, ITERABLE-PRESBURGER(n, θ) is an NDD representing the set of all the vector values $\vec{v} \in \mathbf{Z}^n$ to which θ can be applied an infinite number of times.*

Proof The algorithm in Figure 8.19 is a direct implementation of the computation method discussed in this section. □

It is worth noticing that all the sets computed during the execution of the algorithms in Figures 8.18 and 8.19 are *closed convex polyhedra*, i.e., sets whose elements are the solutions of a linear system of inequations. A possible optimization of these

function ITERABLE-BASIS(basis r, dimension n,
linear operation $(P\vec{x} \leq \vec{q} \rightarrow \vec{x} := A\vec{x} + \vec{b})$) : NDD

1: **var** m, p : integers;
2: $\vec{b'}$: integer vector;
3: $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$: NDDs;
4: **begin**
5: $(\mathbf{T}, m, p) :=$ DEFINABLE-CLOSURE?(r, n, A);
6: $\vec{b'} := \sum_{0 \leq i < p} A^i \vec{b}$;
7: $\mathcal{A}_1 :=$ NDD$(\bigcap_{0 \leq j < p} \{\vec{v} \in \mathbf{Z}^n \mid P(A^j \vec{v} + \sum_{0 \leq i < j} A^i \vec{b}) \leq \vec{q}\})$;
8: $\mathcal{A}_2 :=$ NDD$(\bigcap_{0 \leq j < p} \{\vec{v} \in \mathbf{Z}^n \mid P(A^p A^j \vec{v} + A^p \sum_{0 \leq i < j} A^i \vec{b} + \vec{b'}) \leq \vec{q}\})$;
9: **if** $m = 0$ **then**
10: **if** $PA^p\vec{b'} \not\leq \vec{0}$ **then return** NDD$(\emptyset)$;
11: **else return** INTERSECTION$(\mathcal{A}_1, \mathcal{A}_2)$
12: **else**
13: **begin**
14: $\mathcal{A}_3 :=$ NDD$(\bigcap_{0 \leq j < p} \{\vec{v} \in \mathbf{Z}^n \mid PA^p((r^m - 1)(A^j \vec{v} + \sum_{0 \leq i < j} A^i \vec{b}) + \vec{b'}) \leq \vec{0}\})$;
15: **return** INTERSECTION(INTERSECTION$(\mathcal{A}_1, \mathcal{A}_2), \mathcal{A}_3$)
16: **end**
17: **end**.

Figure 8.18: Set of vector values to which a linear operation can be applied infinitely many times (in a given basis).

---

function ITERABLE-PRESBURGER(dimension n,
linear operation $(P\vec{x} \leq \vec{q} \rightarrow \vec{x} := A\vec{x} + \vec{b})$) : NDD

1: **begin**
2: **var** p : integers;
3: $\vec{b}'$: integer vector;
4: $\mathcal{A}_1, \mathcal{A}_2$: NDDs;
5: **begin**
6: $(\mathbf{T}, 0, p) :=$ DEFINABLE-CLOSURE?$(1, n, A)$;
7: $\vec{b}' := \sum_{0 \leq i < p} A^i \vec{b}$;
8: **if** $PA^p\vec{b}' \not\leq \vec{0}$ **then return** NDD$(\emptyset)$;
9: $\mathcal{A}_1 :=$ NDD$(\bigcap_{0 \leq j < p} \{\vec{v} \in \mathbf{Z}^n \mid P(A^j\vec{v} + \sum_{0 \leq i < j} A^i\vec{b}) \leq \vec{q}\})$;
10: $\mathcal{A}_2 :=$ NDD$(\bigcap_{0 \leq j < p} \{\vec{v} \in \mathbf{Z}^n \mid P(A^pA^j\vec{v} + A^p \sum_{0 \leq i < j} A^i\vec{b} + \vec{b}') \leq \vec{q}\})$;
11: **return** INTERSECTION$(\mathcal{A}_1, \mathcal{A}_2)$
12: **end**.

Figure 8.19: Set of vector values to which a linear operation can be applied infinitely many times (in any basis).

function MULTI-ITERABLE-BASIS(basis r, dimension n,
set of linear operations $\{\theta_1, \theta_2, \ldots, \theta_q\}$) : NDD

1: **begin**
2: **return** $\bigcup_{1 \leq i \leq q}$ ITERABLE-BASIS(r, n, θ_i)
3: **end**.

Figure 8.20: Set of vector values to which a finite set of linear operations can be applied infinitely many times (in a given basis).

function MULTI-ITERABLE-PRESBURGER(dimension n,
set of linear operations $\{\theta_1, \theta_2, \ldots, \theta_q\}$) : NDD

1: **begin**
2: **return** $\bigcup_{1 \leq i \leq q}$ ITERABLE-PRESBURGER(n, θ_i)
3: **end**.

Figure 8.21: Set of vector values to which a finite set of linear operations can be applied infinitely many times (in any basis).

algorithms would consist of manipulating sets of vector values with the help of programs specifically designed for handling closed convex polyhedra. The description of such a program can be found in [LV92].

Since our implementation of MULTI-META-SET for systems with integer variables simply returns cycle meta-transitions, algorithms for computing the set of vector values to which it is known that a finite set of linear operations can be applied infinitely many times can easily be obtained from the ones in Figures 8.18 and 8.19. The resulting algorithms are given in Figures 8.20 and 8.21.

Theorem 8.41 *Let $n \in \mathbf{N}$ be a dimension, $r \in \mathbf{N}$ with $r > 1$ be a basis, and $\theta_1, \theta_2, \ldots, \theta_q$ ($q \in \mathbf{N}$) be linear operations such that META-BASIS?$(r, n, A_i) = \mathbf{T}$ for every $i \in \{1, 2, \ldots, q\}$, where A_i is the transformation matrix of θ_i. MULTI-ITERABLE-BASIS$(r, n, \{\theta_1, \theta_2, \ldots, \theta_q\})$ is an NDD representing in basis r a set $S \subseteq \mathbf{Z}^n$ such that for every $\vec{v} \in S$, the set of linear operations $\{\theta_1, \theta_2, \ldots, \theta_q\}$ can be applied an infinite number of times to $\vec{v}$.*

Proof Immediate. □

function NDD-FINITE?(basis r, dimension n, NDD $\mathcal{A}$) : $\{\mathbf{T}, \mathbf{F}\}$;

1: **begin**
2: $\mathcal{A}$:= DIFFERENCE($\mathcal{A}$, $\bigcup_{a \in \{0, r-1\}^n} a \cdot a \cdot (\{0, 1, \ldots, r-1\}^n)^*$);
3: **return** FINITE?($\mathcal{A}$)
4: **end**

Figure 8.22: Test of finiteness of a set represented as an NDD.

Theorem 8.42 *Let $n \in \mathbf{N}$ be a dimension and $\theta_1, \theta_2, \ldots, \theta_q$ ($q \in \mathbf{N}$) be linear operations such that META-PRESBURGER?$(n, A_i) = \mathbf{T}$ for every $i \in \{1, 2, \ldots, q\}$, where A_i is the transformation matrix of θ_i. For any $r > 1$, MULTI-ITERABLE-PRESBURGER$(n, \{\theta_1, \theta_2, \ldots, \theta_q\})$ is an NDD representing a set $S \subseteq \mathbf{Z}^n$ such that for every $\vec{v} \in S$, the set of linear operations $\{\theta_1, \theta_2, \ldots, \theta_q\}$ can be applied an infinite number of times to $\vec{v}$.*

Proof Immediate. □

8.6 Termination

The goal of this section is to give algorithms for computing the truth value of the predicates required by Sections 5.1 to 5.5 in the context of ISMAs. Specifically, we implement the predicates FINITE?, whose purpose is to decide the finiteness of a set of vector values represented as an NDD, and PRECEDES?, which checks whether two linear operations θ_1 and θ_2 are such that $\theta_1 \lhd \theta_2$. We address each problem separately.

8.6.1 Finiteness of Sets of Vector Contents

Deciding the finiteness of a set of vector contents $S \subseteq \mathbf{Z}^n$ ($n \in \mathbf{N}$) represented as an NDD $\mathcal{A}$ is easy. Since each element of S has exactly one shortest encoding, this can be done by checking whether the language of the shortest encodings of the elements of S is finite or not. This language can be expressed as the difference between $L(\mathcal{A})$ and the language of all the encodings of vectors of $\mathbf{Z}^n$ in which the sign digit is repeated. Testing the finiteness of the language accepted by a finite-state automaton can be done by the algorithm of Figure 7.41. The resulting algorithm for deciding the emptiness of a set of vector values represented as an NDD is given in Figure 8.22.

Theorem 8.43 *Let $n \in \mathbf{N}$ be a dimension, $r \in \mathbf{N}$ with $r > 1$ be a basis, and $\mathcal{A}$ be an NDD representing the set $S \subseteq \mathbf{Z}^n$ in basis r. NDD-FINITE?$(r, n, \mathcal{A}) = \mathbf{T}$ if and only if S is finite.*

Proof Immediate. □

8.6.2 Precedence Relation

The problem addressed here consists of deciding whether two linear operations θ_1 and θ_2 are such that $\theta_1 \triangleleft \theta_2$, i.e., whether $(\theta_2; \theta_1)(S) \subseteq (\theta_1; \theta_2)(S)$ for every subset S of $\mathbf{Z}^n$.

Let $n \in \mathbf{N}$ be a dimension, $r \in \mathbf{N}$ with $r > 1$ be a basis, and let

$$\begin{aligned} \theta_1 &= (P_1\vec{x} \leq \vec{q}_1 \rightarrow \vec{x} := A_1\vec{x} + \vec{b}_1); \\ \theta_2 &= (P_2\vec{x} \leq \vec{q}_2 \rightarrow \vec{x} := A_2\vec{x} + \vec{b}_2), \end{aligned}$$

with $m_1, m_2 \in \mathbf{N}$, $P_1 \in \mathbf{Z}^{m_1 \times n}$, $P_2 \in \mathbf{Z}^{m_2 \times n}$, $\vec{q}_1 \in \mathbf{Z}^{m_1}$, $\vec{q}_2 \in \mathbf{Z}^{m_2}$, $A_1, A_2 \in \mathbf{Z}^{n \times n}$ and $\vec{b}_1, \vec{b}_2 \in \mathbf{Z}^n$. We have

$$\begin{aligned} (\theta_1; \theta_2) &= (P\vec{x} \leq \vec{q} \rightarrow \vec{x} := A\vec{x} + \vec{b}); \\ (\theta_2; \theta_1) &= (P'\vec{x} \leq \vec{q}' \rightarrow \vec{x} := A'\vec{x} + \vec{b}'), \end{aligned}$$

with $P = \begin{bmatrix} P_1 \\ P_2A_1 \end{bmatrix}$, $\vec{q} = \begin{bmatrix} \vec{q}_1 \\ \vec{q}_2 - P_2\vec{b}_1 \end{bmatrix}$, $A = A_2A_1$, $\vec{b} = A_2\vec{b}_1 + \vec{b}_2$, $P' = \begin{bmatrix} P_2 \\ P_1A_2 \end{bmatrix}$, $\vec{q}' = \begin{bmatrix} \vec{q}_2 \\ \vec{q}_1 - P_1\vec{b}_2 \end{bmatrix}$, $A' = A_1A_2$, and $\vec{b}' = A_1\vec{b}_2 + \vec{b}_1$. Therefore,

$$\begin{aligned} \theta_1 \triangleleft \theta_2 &\equiv (\forall S \subseteq \mathbf{Z}^n)((\theta_2; \theta_1)(S) \subseteq (\theta_1; \theta_2)(S)) \\ &\equiv (\forall \vec{x} \in \mathbf{Z}^n)(P'\vec{x} \leq \vec{q}' \Rightarrow (P\vec{x} \leq \vec{q} \wedge A\vec{x} + \vec{b} = A'\vec{x} + \vec{b}')). \end{aligned}$$

It follows that we have $\theta_1 \triangleleft \theta_2$ if and only if the set

$$\{\vec{x} \in \mathbf{Z}^n \mid P'\vec{x} \leq \vec{q}'\}$$

is included in the set

$$\{\vec{x} \in \mathbf{Z}^n \mid P\vec{x} \leq \vec{q} \wedge A\vec{x} + \vec{b} = A'\vec{x} + \vec{b}'\}.$$

This can easily be checked thanks to the results of Chapter 6. An algorithm formalizing the decision procedure is given in Figure 8.23.

Theorem 8.44 *Let $n \in \mathbf{N}$ be a dimension and θ_1, θ_2 be two linear operations over $\mathbf{Z}^n$. LINEAR-PRECEDES?$(n, \theta_1, \theta_2) = \mathbf{T}$ if and only if θ_1 and θ_2 are such that $\theta_1 \triangleleft \theta_2$.*

function LINEAR-PRECEDES?(dimension n, linear operations θ_1, θ_2) : $\{\mathbf{T}, \mathbf{F}\}$

1: **var** $P_1, P_2, P, P', A_1, A_2, A, A'$: integer matrices;
2: $\vec{q}_1, \vec{q}_2, \vec{q}, \vec{q}', \vec{b}_1, \vec{b}_2, \vec{b}, \vec{b}'$: integer vectors;
3: $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$: NDDs;
4: **begin**
5: $(P_1\vec{x} \leq \vec{q}_1 \rightarrow \vec{x} := A_1\vec{x} + \vec{b}_1) := \theta_1$;
6: $(P_2\vec{x} \leq \vec{q}_2 \rightarrow \vec{x} := A_2\vec{x} + \vec{b}_2) := \theta_2$;
7: $P := \begin{bmatrix} P_1 \\ P_2A_1 \end{bmatrix}$;
8: $\vec{q} := \begin{bmatrix} \vec{q}_1 \\ \vec{q}_2 - P_2\vec{b}_1 \end{bmatrix}$;
9: $A := A_2A_1$;
10: $\vec{b} := A_2\vec{b}_1 + \vec{b}_2$;
11: $P' := \begin{bmatrix} P_2 \\ P_1A_2 \end{bmatrix}$;
12: $\vec{q}' := \begin{bmatrix} \vec{q}_2 \\ \vec{q}_1 - P_1\vec{b}_2 \end{bmatrix}$;
13: $A' := A_1A_2$;
14: $\vec{b}' := A_1\vec{b}_2 + \vec{b}_1$;
15: $\mathcal{A}_1 := \text{NDD}(\{\vec{x} \in \mathbf{Z}^n \mid P\vec{x} \leq \vec{q}\})$;
16: $\mathcal{A}_2 := \text{NDD}(\{\vec{x} \in \mathbf{Z}^n \mid P'\vec{x} \leq \vec{q}'\})$;
17: $\mathcal{A}_3 := \text{NDD}(\{\vec{x} \in \mathbf{Z}^n \mid A\vec{x} + \vec{b} = A'\vec{x} + \vec{b}'\})$;
18: **return** INCLUDED?($\mathcal{A}_2$, INTERSECTION($\mathcal{A}_1, \mathcal{A}_3$))
19: **end**.

Figure 8.23: Precedence test for linear operations.

Proof The algorithm in Figure 8.23 is the direct implementation of the computation method discussed in this section. □

One again here, the sets computed during the execution of the algorithm in Figure 8.23 are closed convex polyhedra, and specific manipulation routines may be used.

8.7 Loop Optimization

This section is aimed at providing implementations of the predicate EXISTS-LOOP-EQUIV? and of the function LOOP-EQUIV-OP required by the loop optimization technique introduced in Section 5.6. In the present context, the predicate EXISTS-LOOP-EQUIV? takes as arguments two linear operations θ_1 and θ_2 corresponding respectively to the label of the body of a loop to which a cycle meta-transition can be associated and to the label of the exit transition of that loop. It returns **T** if a memory function f equivalent to $(\theta_1^+;\theta_2)$ can be determined. The purpose of the function LOOP-EQUIV-OP is to compute f given θ_1 and θ_2. There is no need for EXISTS-LOOP-EQUIV? to be complete, i.e., to return **T** for every pair (θ_1,θ_2) such that there exists a computable memory function equivalent to $(\theta_1^+;\theta_2)$. Instead, we simply require EXISTS-LOOP-EQUIV? to be easily computable, and to be such that for every pair (θ_1,θ_2) such that EXISTS-LOOP-EQUIV?$(\theta_1,\theta_2) = \mathbf{T}$, LOOP-EQUIV-OP$(\theta_1,\theta_2)$ returns a memory function equivalent to $(\theta_1^+;\theta_2)$.

Let $n \in \mathbf{N}$ be a dimension, $r \in \mathbf{N}$ with $r > 1$ be a basis, and $\theta_1 = (P_1\vec{x} \leq \vec{q_1} \rightarrow \vec{x} := A_1\vec{x} + \vec{b_1})$, $\theta_2 = (P_2\vec{x} \leq \vec{q_2} \rightarrow \vec{x} := A_2\vec{x} + \vec{b_2})$, where $m_1, m_2 \in \mathbf{N}$, $P_1 \in \mathbf{Z}^{m_1 \times n}$, $P_2 \in \mathbf{Z}^{m_2 \times n}$, $\vec{q_1} \in \mathbf{Z}^{m_1}$, $\vec{q_2} \in \mathbf{Z}^{m_2}$, $A_1, A_2 \in \mathbf{Z}^{n \times n}$ and $\vec{b_1}, \vec{b_2} \in \mathbf{Z}^n$, be linear operations. The loop optimization method that we propose is based on the following ideas. First, since it must be possible to associate a meta-transition to the cycle whose body is labeled by θ_1, we require that θ_1 satisfies the hypotheses of Corollary 8.41, i.e., that there exist $m \in \mathbf{N}$ and $p \in \mathbf{N}_0$ such that A_1^p is diagonalizable and has all its eigenvalues in $\{0, r^m\}$.

A consequence of the proof of Theorem 8.43 is that for every vector value $\vec{v} \in \mathbf{Z}^n$, the set $\theta_1^+(\{\vec{v}\})$ can be expressed as the union of p sets, each of them being composed of colinear elements. If there exists a linear operation equivalent to $(\theta_1^+;\theta_2)$, then the image by $(\theta_1^+;\theta_2)$ of a single vector value $\vec{v} \in \mathbf{Z}^n$ cannot contain more than one vector value. A simple way of ensuring that is to require $p = 1$ and $m = 0$, which intuitively means that all the elements of $\theta_1^+(\{\vec{v}\})$ must be colinear and uniformly spaced. We also require that the guard of θ_2 can never be satisfied by more than one of those elements.

The latter requirement can be formalized by following the approach of [BW94]. Assume that A_1 is diagonalizable that all its eigenvalues belong to $\{0, 1\}$. Such a transformation matrix is said to be *idempotent*; it is such that $A_1^2 = A_1$. Let $k \in \mathbf{N}$,

with $k \geq 2$. For every vector value $\vec{v} \in \mathbf{Z}^n$ to which θ_1 can be applied k times, we have $\theta_1^k(\vec{v}) = A_1\vec{v} + (k-1)A_1\vec{b}_1 + \vec{b}_1$. The pairs $(\vec{v}, k) \in \mathbf{Z}^n \times \mathbf{Z}$ such that $k \geq 2$, θ_1 can be applied k times to $\vec{v}$, and $\theta_1^k(\vec{v})$ satisfies the guard of θ_2 are therefore those that satisfy the predicate

$$\begin{aligned}\psi(\vec{v},k) &\equiv P_1\vec{v} \leq \vec{q}_1 \wedge P_1\theta_1(\vec{v}) \leq \vec{q}_1 \wedge \ldots \wedge P_1\theta_1^{k-1}(\vec{v}) \leq \vec{q}_1 \\ &\qquad \wedge P_2\theta_1^k(\vec{v}) \leq \vec{q}_2 \wedge k \geq 2 \\ &\equiv P_1\vec{v} \leq \vec{q}_1 \wedge P_1(A_1\vec{v} + \vec{b}_1) \leq \vec{q}_1 \wedge \ldots \\ &\qquad \wedge P_1(A_1\vec{v} + (k-2)A_1\vec{b}_1 + \vec{b}_1) \leq \vec{q}_1 \\ &\qquad \wedge P_2(A_1\vec{v} + (k-1)A_1\vec{b}_1 + \vec{b}_1) \leq \vec{q}_2 \wedge k \geq 2.\end{aligned}$$

Since the region $\{\vec{x} \in \mathbf{Z}^n \mid P_1\vec{x} \leq \vec{q}_1\}$ is convex, this formula can be rewritten as

$$\begin{aligned}\psi(\vec{v},k) &\equiv P_1\vec{v} \leq \vec{q}_1 \wedge P_1(A_1\vec{v} + \vec{b}_1) \leq \vec{q}_1 \wedge P_1(A_1\vec{v} + (k-2)A_1\vec{b}_1 + \vec{b}_1) \leq \vec{q}_1 \\ &\qquad \wedge P_2(A_1\vec{v} + (k-1)A_1\vec{b}_1 + \vec{b}_1) \leq \vec{q}_2 \wedge k \geq 2 \\ &\equiv P\begin{bmatrix} \vec{v} \\ k \end{bmatrix} \leq \vec{q},\end{aligned}$$

where $P = \begin{bmatrix} P_1 & 0 \\ P_1A_1 & 0 \\ P_1A_1 & P_1A_1\vec{b}_1 \\ P_2A_1 & P_2A_1\vec{b}_1 \end{bmatrix}$ and $\vec{q} = \begin{bmatrix} \vec{q}_1 \\ \vec{q}_1 - P_1\vec{b}_1 \\ \vec{q}_1 + 2P_1A_1\vec{b}_1 - P_1\vec{b}_1 \\ \vec{q}_2 + P_2A_1\vec{b}_1 - P_2\vec{b}_1 \end{bmatrix}$.

Assume now that the linear system of inequations defined by P and $\vec{q}$ contains at least one equation $\vec{\alpha}_1.\vec{v} + \alpha_k k = \beta$, with $\vec{\alpha}_1 \in \mathbf{Z}^n$, $\alpha_k \in \mathbf{Z}_0$ and $\beta \in \mathbf{Z}$ ($\mathbf{Z}_0$ denotes the set of nonzero integers.) This means that the number of iterations k of θ_1 is determined by the vector value $\vec{v}$, i.e., that the loop which is labeled by θ_1 and whose exit transition is labeled by θ_2 is deterministic. In this situation, k can be expressed as the linear function

$$k : \mathbf{Z}^n \to \mathbf{Z} : \vec{v} \mapsto \frac{1}{\alpha_k}(\beta - \vec{\alpha}_1.\vec{v}).$$

If all the coefficients in this linear function are integers, then for every $\vec{v}, \vec{v}\,' \in \mathbf{Z}^n$ such that $\vec{v}\,' = \theta^{k(\vec{v})}(\vec{v})$, we have

$$P\begin{bmatrix} \vec{v} \\ k(\vec{v}) \end{bmatrix} \leq \vec{q} \wedge \vec{v}\,' = A_1\vec{v} + (k(\vec{v}) - 1)A_1\vec{b}_1 + \vec{b}_1.$$

For every $\vec{v}, \vec{v}\,' \in \mathbf{Z}^n$ such that $\vec{v}\,' \in (\theta_1, \theta_1^+, \theta_2)(\vec{v})$, we thus have

$$P\begin{bmatrix} \vec{v} \\ k(\vec{v}) \end{bmatrix} \leq \vec{q} \wedge \vec{v}\,' = A_2(A_1\vec{v} + (k(\vec{v}) - 1)A_1\vec{b}_1 + \vec{b}_1) + \vec{b}_2.$$

Replacing $k(\vec{v})$ by its value, we obtain that the transformation $(\theta_1; \theta_1^+; \theta_2)$ is equivalent to the linear operation $\theta' = (P'\vec{x} \leq \vec{q}\,' \to \vec{x} := A'\vec{x} + \vec{b}')$, where

- $P' = P_L - \frac{1}{\alpha^k}(\vec{p}_R\vec{\alpha}_1^{\mathcal{T}})$,
- $\vec{q}' = \vec{q} - \frac{\beta}{\alpha_k}\vec{p}_R$,
- $A' = A_2A_1 - \frac{1}{\alpha^k}(A_2A_1\vec{b}_1\vec{\alpha}_1^{\mathcal{T}})$, and
- $\vec{b}' = (\frac{\beta}{\alpha_k} - 1)A_2A_1\vec{b}_1 + A_2\vec{b}_1 + \vec{b}_2$.

In these expressions, $\mathcal{T}$ denotes transposition, and $\vec{p}_R$ and P_L denote respectively the rightmost column of P, and the matrix composed of all the other columns of P.

In summary, the transformation $(\theta_1^+;\theta_2)$ is equivalent to the memory function

$$f : 2^{\mathbf{Z}^n} \to 2^{\mathbf{Z}^n} : S \mapsto \theta_2(\theta_1(S)) \cup \theta'(S).$$

The only difficulty in the computation of θ' is to check whether the system of linear inequations defined by P and $\vec{q}$ contains an equation, and to determine the coefficients of this equation if one exists. This can be done straightforwardly if the proper data structures are used for handling systems of linear inequations. Simple solutions to that problem, which are not described in this thesis, can be found in [CH78, Hal93, LV92]. If there are more than one suitable equation in the system defined by P and $\vec{q}$, then the equation that one considers can be chosen arbitrarily.

An algorithm formalizing the computation of a linear operation equivalent to the transformation $(\theta_1;\theta_1^+;\theta_2)$ is given in Figure 8.24.

Theorem 8.45 *Let $n \geq 0$ be a dimension and θ_1, θ_2 be two linear operations over subsets of $\mathbf{Z}^n$. If LINEAR-EQUIV$(n, \theta_1, \theta_2) = (\mathbf{T}, \theta)$, then θ is a linear operation equivalent to $(\theta_1;\theta_1^+;\theta_2)$.*

Proof The algorithm in Figure 8.24 directly implements the computation developed in this section. □

Algorithms implementing the predicate EXISTS-LOOP-EQUIV? and the function LOOP-EQUIV-OP are given in Figures 8.25 and 8.26.

function LINEAR-EQUIV(dimension n, linear operations $(P_1\vec{x} \le \vec{q}_1 \to \vec{x} := A_1\vec{x} + \vec{b}_1)$, $(P_2\vec{x} \le \vec{q}_2 \to \vec{x} := A_2\vec{x} + \vec{b}_2)$) : ($\{\mathbf{T}, \mathbf{F}\}$, linear operation)

var P, P', A', P_L : integer matrices;
  $\vec{\alpha}_1, \vec{q}, \vec{q}', \vec{b}', \vec{p}_R$: integer vectors;
  t : boolean;
  p, α_k, β : integers;
begin
  $(t, 0, p) :=$ DEFINABLE-CLOSURE?$(1, n, A_1)$;
  if $p \neq 1$ **then return** $(\mathbf{F}, \perp)$;
  $$P := \begin{bmatrix} P_1 & 0 \\ P_1A_1 & 0 \\ P_1A_1 & P_1A_1\vec{b}_1 \\ P_2A_1 & P_2A_1\vec{b}_1 \end{bmatrix};$$
  $$\vec{q} := \begin{bmatrix} \vec{q}_1 \\ \vec{q}_1 - P_1\vec{b}_1 \\ \vec{q}_1 + 2P_1A_1\vec{b}_1 - P_1\vec{b}_1 \\ \vec{q}_2 + P_2A_1\vec{b}_1 - P_2\vec{b}_1 \end{bmatrix};$$
  if $(\exists \vec{\alpha}_1 \in \mathbf{Z}^n, \alpha_k, \beta \in \mathbf{Z})$ **such that** $(\vec{\alpha}_1.\vec{v} + \alpha_k k = \beta) \in P \begin{bmatrix} \vec{v} \\ k \end{bmatrix} \le \vec{q}$ $\wedge \frac{1}{\alpha_k}\vec{\alpha}_1 \in \mathbf{Z}^n \wedge \frac{\beta}{\alpha_k} \in \mathbf{Z}$ **then**
    begin
      $[P_L; \vec{p}_R] := P$;
      $P' := P_L - \frac{1}{\alpha^k}(\vec{p}_R\vec{\alpha}_1^{\mathcal{T}})$;
      $\vec{q}' := \vec{q} - \frac{\beta}{\alpha_k}\vec{p}_R$;
      $A' := A_2A_1 - \frac{1}{\alpha^k}(A_2A_1\vec{b}_1\vec{\alpha}_1^{\mathcal{T}})$;
      $\vec{b}' = (\frac{\beta}{\alpha_k} - 1)A_2A_1\vec{b}_1 + A_2\vec{b}_1 + \vec{b}_2$;
      return $(\mathbf{T}, (P'\vec{x} \le \vec{q}' \to \vec{x} := A'\vec{x} + \vec{b}'))$
    end
  else return $(\mathbf{F}, \perp)$
end.

Figure 8.24: Computation of a linear operation equivalent to $(\theta_1; \theta_1^+; \theta_2)$.

```
function EXISTS-LOOP-EQUIV?(dimension n, linear operations θ1, θ2) : {T, F}
1:    var θ : linear operation;
2:        t : boolean;
3:    begin
4:        (t, θ) := LINEAR-EQUIV(n, θ1, θ2);
5:        return t
6:    end.
```

Figure 8.25: Predicate EXISTS-LOOP-EQUIV? for linear operations.

```
function LOOP-EQUIV-OP(dimension n, linear operations θ1, θ2) : function
1:    var θ : linear operation;
2:    begin
3:        (T, θ) := LINEAR-EQUIV(n, θ1, θ2);
4:        return (θ1; θ2) ∪ θ
5:    end.
```

Figure 8.26: Function LOOP-EQUIV-OP for linear operations.

Chapter 9

Conclusions

9.1 Summary

The central theme of this thesis is a new approach for performing the state-space exploration of systems with an infinite state space. The exact class of infinite-state systems considered are those that can be modeled as a state machine with a finite control and an infinite data domain. No restrictions are imposed on this data domain, except the requirement that its structure and algebraic properties make the computation of a given set of operations possible.

The proposed approach for exploring infinite state spaces extends the classical state-space search technique for finite-state systems. It relies on two central ideas: carrying out the state-space exploration with possibly infinite sets of states rather than with individual states, and the concept of *meta-transition*. A meta-transition is a generalization of the notion of transition with which it is possible to deduce the reachability of an infinite set of states from the reachability of a finite set of states. Two important types of meta-transitions were considered, those corresponding respectively to cycles and to finite sets of cycles present in the control graph of the system. Once meta-transitions have been identified, an infinite-state system can undergo a state-space search consisting of a simple generalization of classical state-space exploration algorithms.

Computing a finite representation with decidable membership of the set of reachable states of an infinite-state system has been the main focus of this work. This gives a solution to the reachability problem (is a given state reachable) as well as to some of its variants and to other problems that are easily solved given a representation of the reachable states, e.g., boundedness and absence of deadlocks. Furthermore, the verification of *temporal properties*, and more specifically of properties than can be expressed in Linear-time Temporal Logic (LTL) or as Büchi automata, has been addressed. A partial decision procedure has been obtained for this undecidable problem, in the form of an extension of the infinite state-space exploration algorithms

that have been developed.

In general, though the reachability problem for infinite-state systems is undecidable, it is possible to give sufficient syntactic conditions under which the exploration of the state space is guaranteed to terminate. Examples of such conditions are given, and it is shown that the LTL model-checking problem becomes decidable for infinite-state systems satisfying some of these conditions. In addition, we have studied an optimization technique that allows the control graph of systems to be modified in a way that preserves their behavior, but that makes satisfying the sufficient termination conditions more likely.

Since the infinite state-space exploration algorithms that we have introduced proceed by manipulating sets of states that may be infinite, they need a representation system for such sets. We have introduced a general technique for obtaining suitable representation systems in a large number of domains. This technique consists of encoding memory contents as words over some finite alphabet, and of representing sets as finite automata accepting the encodings of the elements of these sets. The advantage of this approach over other representations is that set operations translate naturally into simple operations over automata.

This general technique has been particularized to two important classes of infinite-state systems. The first is the one of systems that use a finite number of unbounded FIFO channels, on which send and receive operations are performed. The representation system adapted for such systems is the *Queue Decision Diagram*, or QDD. Another task was to show that all the operations required by the state-space exploration, the model-checking, and the termination study of systems using unbounded FIFO channels can be performed with QDDs. Among other results that were obtained during the design of the algorithms, it has been proved constructively that the iteration of any sequence of send and receive operations involving only one channel preserves the recognizable nature of sets of channel contents. Those results have been generalized to sequences involving more than one channel, in the form of an exact decision procedure for the preservation of recognizability, and of an algorithm for computing the effect of iterating a sequence of operations. Another result that has been obtained is that *lossy* systems can easily be analyzed by simply adding a new type of meta-transition.

The second class of infinite-state systems that has been studied is the one containing systems using unbounded integer variables on which linear operations are performed. The representation system developed for such systems is the *Number Decision Diagram*, or NDD. It has been shown that all the operations required by state-space exploration, model-checking, and the termination study of systems using unbounded integer variables can be performed with NDDs. An interesting result obtained in this context is a decision procedure for determining whether the closure of a guardless linear operation preserves the recognizability of sets of integer vector values. In order to develop this decision procedure, an original extension of the

concept of definability to vectors with complex components has been introduced. Algorithms have also been provided for applying the closure of a linear operation to sets represented as NDDs, as well as for optimizing the control graphs of systems using unbounded integer variables.

9.2 Related Work

A methodologically related approach is the symbolic model checking of finite-state systems [CMB91, BCM+92, McM93]. It consists of representing symbolically sets of states as well as the transition relation between these states, and of expressing the set of reachable states of the system as the solution of a fixpoint equation. The symbolic representation system that is mostly used is the Binary Decision Diagram (BDD) [Bry92]. The main advantage of this approach over enumerative state-space exploration [Hol88, Hol90, HK90, Hol91, DDHY92, FGM+92] is that the sets of states are represented and manipulated implicitly rather than explicitly. This may reduce dramatically the total cost of the exploration. The main limit of symbolic model-checking using BDDs is that it can only be applied to systems with a finite state space.

In this thesis, we have extended the scope of symbolic state-space exploration by allowing to compute the reachability of an infinite number of states in a finite amount of time. The idea of capturing the state-space periodicity that results from repeated executions of the same operations is not new. In [KM69], Karp and Miller show that this approach makes it possible to decide the boundedness problem for Petri nets. Sketchily, their decision procedure consists of computing an upper approximation of the set of reachable markings of a Petri net, by performing a state-space exploration in which every sequence of transitions that can be repeatedly followed an infinite number of times produces an upper bound of the set of markings that are reached during these repetitions. The symbolic representation system that is used is rather simple and consists of replacing in the description of markings each unbounded component by the special value ω. The set of states returned by the algorithm of Karp and Miller does not exactly correspond to the set of reachable markings of the Petri net, but allows to decide the boundedness of each place.

Other authors have investigated the possibility of computing exactly the set of reachable states of an infinite-state system by considering the effect of repeated executions of the same operations. Lubachevsky [Lub84] uses mathematical induction as a tool for establishing the reachability of infinite sets of states, in the field of systems composed of a large number of identical processing elements. The method consists of performing a depth-first search in the state space of the system, in which one detects in exploration paths particular sequences of transitions that are repeated

more than a given amount of times. When such a sequence is detected, one attempts to compute in one step the effect of its repetition and then resumes the search (without any guarantee of termination). This approach has similarities with the dynamic state-space exploration algorithm presented in Section 3.5. However, Lubachevsky does not describe algorithms for detecting repeated transitions or for computing their effect, and does not provide a representation system for sets of states.

Similar ideas also appear in a paper by Valmari [Val89], which describes an extended state-space exploration algorithm that is able to compute the effect of infinitely repeating sequences of operations. Specifically, the sequences that can be iterated are those in which the value of exactly one integer variable grows one by one, and the value of all the other variables stays unchanged. The representation system for sets of states consists of formulas expressed in a dedicated formalism equivalent to a restricted subset of Presburger arithmetic. The results that we have obtained in Chapter 8 thus strictly extend those of Valmari.

The first definition of the concept of meta-transition appears in [Boi93], in which it is shown that adding cycle meta-transitions to a system may speed up its state-space exploration. The class of systems that is considered is the one of state machines associated with a finite set of integer variables. The reachability analysis of such a system is carried out by performing a depth-first search in which a cycle analysis takes place whenever the same control location appears twice in an exploration path. A cycle meta-transition is then created each time one finds a cycle labeled by a sequence of operations that can be iterated. The main limits of this technique are that only a simple sufficient condition is given for detecting iterable sequences of transitions (amounting to require an idempotent transformation matrix), and that the representation system used for sets of states is only able to represent finite unions of convex sets.

The same state-space exploration technique appears in [BW94] together with an improved representation system for sets of states. This representation system, which consists of associating a set of *periodicity vectors* to the set of integer solutions of a linear system of inequations, is closed over the set of all the operations needed by the state-space exploration with cycle meta-transitions. Its main drawback is the difficulty of deciding the inclusion of a set of states into another, which is unfortunately essential for detecting the convergence of state-space exploration. It can easily be shown that the representation system introduced in [BW94] is inclusively less expressive that the NDDs. Since the algorithms of Chapter 7 allow to apply the closure of all the sequences whose transformation matrix is idempotent, the results presented in this thesis supersede those of [BW94].

As already mentioned, the most widely used representation system in the context of symbolic exploration is the *Binary Decision Diagram (BDD)* [Bry92]. The idea consists of encoding the elements of a set as fixed-length words of bits. The set is then represented by a canonical decision diagram – isomorphic to a directed acyclic

graph – that recognizes the encodings of all the elements of the set. This simple and elegant representation has efficient implementations, and can easily be applied to a large class of domains. It does however suffer from an important drawback: BDDs only allow to represent finite sets. As a consequence, symbolic exploration with BDDs is limited to the analysis of models with a finite state space. Nonetheless, BDDs have similarities with the finite-state representations introduced in this thesis. Representing a set as a BDD actually consists of constructing a minimal finite-state machine that accepts the encodings of the elements of the set, with the restriction that the length of those encodings is fixed. The finite-state representations proposed in Chapter 6 can thus be seen as generalizations of the concept of BDD. By using the minimization operation, these representations can easily be converted into BDDs if the sets that they represent are finite.

Systems whose infinite nature results from the use of unbounded FIFO channels have been studied for a long time [BZ83, MF85, Pac86]. A restricted class of such systems that has received much attention is the one of *lossy systems*, which are systems whose FIFO channels are unreliable and may nondeterministically lose messages. It has been shown by Abdulla and Jonsson [AJ93, AJ94] that several interesting verification problems are decidable for this class of systems, namely the restricted reachability problem, the problem consisting of deciding safety properties expressed as a set of regular traces, and the eventuality problem. These results are strictly more powerful than the ones presented in Section 7.5, in which we have only shown that the main results of Chapter 7 can be adapted with little difficulty to lossy systems by simply adding a new type of meta-transition. It is however possible to solve the restricted reachability problem using the meta-transition based state-space exploration method proposed in this thesis. The only required modifications are to perform the search backwards (from the state whose reachability is to be determined to the initial state) rather than forwards, and to create for each control location a special meta-transition that nondeterministically inserts arbitrary symbols into the queue contents. Thanks to a result due to Higman [Hig52], the search then always terminates. The result of the search is the set of predecessors of the state of interest, and this state is reachable if and only if that set contains the initial state.

In fairly recent work, Finkel [Fin90, Fin94], Cécé and Iyer [CFI96] have also demonstrated that an infinite state space does not always prevent one from being able to decide interesting properties of systems using unbounded FIFO queues. Precisely, these authors consider several restricted classes of such systems, and establish the decidability or the undecidability of different important problems over these classes. In particular, they show that there are families of systems such as those with *insertion errors* for which a finite-state representation of their set of reachable states always exists and can always be computed. The approach followed in this thesis is significantly different from the one of Finkel and al. Rather than isolating elegant but very restricted classes of systems for which some simple properties

can always be decided, we have developed partial algorithmic solutions for deciding reachability properties of full-fledged systems. Nevertheless, we have provided in Chapter 5 sufficient static conditions that characterize a subclass of systems using FIFO channels for which a finite representation of their set of reachable states can always be computed. These conditions are quite different from the ones described in in [Fin90, Fin94, CFI96], which is far from being surprising. Indeed, our conditions are derived from the state-space exploration algorithm rather than the other way around. Moreover, the sufficient conditions presented in Chapter 5 have been developed independently from a particular data domain and are thus also applicable to systems different from those using FIFO channels.

There are other ways than ours of obtaining useful partial decision procedures for interesting properties of systems using FIFO channels. In [JJ93], Jard and Jéron address the boundedness problem, and give a partial solution based on the detection of sequences of queue operations that can be followed an infinite number of times. Their approach differs from the one promoted in Chapter 3 in that they consider sequences of queue operations that can always be applied infinitely many times to at least one initial queue-set content, rather than sequences whose closure can always be computed with respect to some symbolic representation system. Actually, the technique of Jard and Jéron can be seen as a generalization to systems using FIFO channels of the Karp and Miller solution to the boundedness problem [KM69]. The condition given in [JJ93] that allows to determine whether a given sequence of queue operations can be applied an infinite number of times is actually equivalent to the one that has been obtained in Sections 7.6.1 and 7.6.2, except that the latter condition allows an easy and efficient computation of the set of queue-set contents from which the sequence can be followed an infinite number of times. The scope of the algorithm of Jard and Jéron has been extended by Burkhart, Jéron and Quemener [QJ95, QJ96, BQ96], which use this algorithm for building a finite representation of the state space of a system using unbounded FIFO queues. Their representation consists of a graph grammar, i.e., a set of transformation rules that finitely describes an infinite graph, and differs from ours in that it represents, in addition to the reachable states, the reachability relation between these states. It is shown in [QJ95, QJ96] that the branching-time temporal logic CTL can be decided for restricted classes of systems, using a simple extension of the algorithm of Jard and Jéron. This result is generalized to the μ-calculus in [BQ96].

The notion of QDD and some algorithms for performing elementary operations on QDDs first appeared in [BG96b]. In that paper, the sequences of queue operations from which one is able to create meta-transitions are limited to three very restricted subclasses, the purpose of this restriction being to simplify the algorithms for computing the effect of meta-transitions. These results are improved in [BGWW97], in which a full decision procedure for the sequences of queue operations whose closure can be computed is presented. Most of the results appearing in Chapter 7 are

actually detailed descriptions of results announced in [BGWW97].

Several extensions of the QDDs have been developed in these recent years. A generalization of QDDs that broadens their expressiveness is proposed in [BH97]. This generalization consists of associating with the QDD a set of integer variables constrained by formulas of Presburger arithmetic, and of restricting the form of the state-transition graph of the QDD. Although attractive, this extended representation system has the disadvantage of not being closed under all usual operations, which limits its applicability. An alternative to QDDs is also proposed in [FM96] in the form of a representation system that is less expressive but easier to manipulate. BDDs have also been used for representing finite but large sets of queue-set contents. An elegant encoding scheme that facilitates the computation of queue operations over finite sets represented as BDDs is described in [GL96].

Systems using integer variables have been a subject of intense study. A excellent survey of decidability results for such systems is presented in [EN94]. Recent developments in this field include the design of efficiently manageable representation systems for sets of integer vector values. Such a representation system has been developed by Pugh [Pug92a, Pug92b, Pug94] and has been implemented in a tool called the *Omega Test*. It proceeds by representing sets as formulas of Presburger arithmetic, on which some carefully selected simplifications are made in order to keep their size as low as possible. In spite of the very high theoretical lower bound on the complexity of deciding Presburger arithmetic, the Omega tool allows to manipulate Presburger-definable sets with a cost that in practice is quite low. The idea of using finite-state automata as a practical representation of sets of vector values appeared in [WB95], in which it was shown that integer programming can be solved in its known optimal lower bound with this technique. The same idea is also present in the work of Boudet and Comon [BC96], who have given algorithms for computing efficiently the minimal NDD representing the set of solutions of a system of equations and of inequations. An extension of finite-state representations to sets of real vectors has been proposed in [BBR97]. This extension simply consists of encoding real vectors as infinite words over a finite alphabet, and of representing sets as finite-state automata on infinite words.

A class of infinite-state systems that has not been studied in this thesis is the one of *pushdown systems*. These systems belong however to a natural class to consider in order to obtain decidability results, and there are indeed already a number of results on that topic [MS85, HS91, HJM94, BS95, Wal96]. Furthermore, it is known [Cau92] that the set of reachable states of a pushdown system can always be represented by a finite-state automaton, and that a finite-state representation of this set can always be effectively computed. In [BEM97, FWW97], a symbolic representation system similar to QDDs is used in order to compute the set of reachable states of a pushdown system. This method makes it possible to perform linear-time model checking, and has been generalized to branching-time temporal logic in [FWW97].

9.3 Future Work

The primary purpose of the algorithms presented in this thesis is to prove that the functions that they implement are actually computable. Although most of them can readily be translated into actual code, this translation is by no means always a straightforward task and deserves further work. Due to the lack of an actual implementation, the actual cost of analyzing systems with the approach promoted here is still unknown, even though small tests carried out with prototypes of early versions of this work have given encouraging results.

Another subject of potential research is to evaluate the benefit of symbolic methods for analyzing systems that have a large but finite state space. For instance, in the case of systems using FIFO channels whose capacity is large but bounded, it is our understanding that using symbolic state-space exploration with QDDs rather than traditional state-space exploration might reduce dramatically the cost of the analysis for a large class of systems.

The representation system introduced in Chapter 6 is very general and can be used in a large number of domains. A direction that we did not follow but that seems promising could be to combine two different domains into an heterogenous representation system for sets. For instance, a memory content of a system using FIFO channels as well as integer variables could be encoded as the concatenation of the encodings of both parts of the content. The challenge would then be to combine the algorithms developed for individual domains into ones suited for combined representations.

The reachability analysis performed by the technique that have been introduced does not rely on approximations, i.e., it computes exactly the set of reachable states of the system. This approach has the disadvantage that termination is not guaranteed for sufficiently expressive systems. Another direction for future research could be to introduce in the framework of symbolic state-space exploration operators that force convergence (at the cost of introducing approximations). One could for instance consider transformations analogous to the *widening operators* used in abstract interpretation [CC77, CC92, JN95].

Finally, we stress the fact that symbolic representations for possibly infinite sets of values have applications well beyond verification issues. Domains such as *temporal databases* could indeed benefit from the types of representation systems that we have developed [KSW90].

Bibliography

[ACD90] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proceedings of the 5th Symposium on Logic in Computer Science*, pages 414–425, Philadelphia, June 1990.

[ACJT96] P. A. Abdulla, K. Cerāns, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science*, pages 313–321, New Brunswick, New Jersey, July 1996. IEEE Computer Society Press.

[AHH93] R. Alur, T. A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. In *Proceedings of the 15th Annual Real-Time Systems Symposium*, pages 2–11. IEEE Computer Society Press, 1993.

[AJ93] P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. In *Proceedings of the 8th IEEE International Symposium on Logic in Computer Science*, pages 160–171, 1993.

[AJ94] P. A. Abdulla and B. Jonsson. Undecidable verification problems for programs with unreliable channels. In *Proceedings of ICALP'94*, volume 820 of *Lecture Notes in Computer Science*, pages 316–327. Springer-Verlag, 1994.

[AJ96] P. A. Abdulla and B. Jonsson. Undecidable verification problems for programs with unreliable channels. *Information and Computation*, 130(1):71–90, October 1996.

[AU72] A. Aho and J. D. Ullman. *The Theory of Parsing, Translation, and Compiling.* Prentice-Hall, 1972.

[BBR97] B. Boigelot, L. Bronne, and S. Rassart. An improved reachability analysis method for strongly linear hybrid systems. In *Proceedings of the 9th International Conference on Computer-Aided Verification*, number 1254 in Lecture Notes in Computer Science, pages 167–177, Haifa, Israel, June 1997. Springer-Verlag.

[BC96] A. Boudet and H. Comon. Diophantine equations, Presburger arithmetic and finite automata. In *Proceedings of CAAP'96*, number 1059 in Lecture Notes in Computer Science, pages 30–43. Springer-Verlag, 1996.

[BCM+92] J. Burch, E. M. Clarke, K. McMillan, D. Dill, and L. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model checking. In *Proceedings of CONCUR '97*, number 1243 in Lecture Notes in Computer Science, pages 135–150. Springer-Verlag, 1997.

[BFH91] A. Bouajjani, J.-C. Fernandez, and N. Halbwachs. Minimal model generation. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 3:85–91, 1991.

[BG96a] B. Boigelot and P. Godefroid. Model checking in practice: An analysis of the ACCESS.bus protocol using SPIN. In *Proc. Formal Methods Europe*, volume 1051 of *Lecture Notes in Computer Science*, pages 456–478, Oxford, UK, March 1996. Springer-Verlag.

[BG96b] B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using QDDs. In *Proc. Computer Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 1–12, New-Brunswick, New-Jersey, July 1996. Springer-Verlag.

[BGWW97] B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDDs. Submitted for publication, 1997.

[BH97] A. Bouajjani and P. Habermehl. Symbolic reachability analysis of FIFO channel systems with nonregular sets of configurations. In *Proceedings of ICALP'97*, number 1256 in Lecture Notes in Computer Science, pages 560–570, Bologna, Italy, July 1997. Springer-Verlag.

[BHMV94] V. Bruyère, G. Hansel, C. Michaux, and R. Villemaire. Logic and p-recognizable sets of integers. *Bulletin of the Belgian Mathematical Society*, 1(2):191–238, March 1994.

[Bod59] E. Bodewig. *Matrix Calculus*. Elsevier North-Holland, Amsterdam, second edition edition, 1959.

[Boi93] B. Boigelot. Développement d'une technique de vérification de systèmes parallèles combinant l'utilisation d'un invariant et l'exploration de l'espace d'états. Travail de fin d'études, Université de Liège, 1993.

[BQ96] O. Burkhart and Y.-M. Quemener. Model-checking of infinite graphs defined by graph grammars. Technical Report 995, IRISA, April 1996.

[Bru85] V. Bruyère. Entiers et automates finis. Mémoire de fin d'études, Université de Mons, 1985.

[Bry92] R. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[BS95] O. Burkart and B. Steffen. Composition, decomposition and model checking of pushdown processes. *Nordic Journal of Computing*, 2(2):89–125, 1995.

[Büc60] J. R. Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift Math. Logik und Grundlagen der Mathematik*, 6:66–92, 1960.

[Büc62] J. R. Büchi. On a decision method in restricted second order arithmetic. In *Proceedings of the International Congress on Logic, Method, and Philosophy of Science*, pages 1–12, Stanford, California, 1962. Stanford University Press.

[BW94] B. Boigelot and P. Wolper. Symbolic verification with periodic sets. In *Proc. 6rd Workshop on Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 55–67, Stanford, June 1994. Springer-Verlag.

[BZ83] D. Brand and P. Zafiropoulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, 1983.

[Cau92] D. Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106:61–86, 1992.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Fourth ACM Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, January 1977. ACM Press, New York.

[CC92] P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(2–3):103–179, 1992.

[CCI88] CCITT. Specification and description language SDL. In *Recommendation Z.100. Blue Book X.1–X-5*. ITU General Secretariat, Geneva, 1988.

[CE81] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. Technical Report TR-12-81, Center for Research in Computing Technology, Harvard University, 1981.

[CFI96] G. Cécé, A. Finkel, and S. P. Iyer. Unreliable channels are easier to verify than perfect channels. *Information and Computation*, 124(1):20–31, October 1996.

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the 5th ACM Symposium on Principles of Programming Languages*, pages 84–96, 1978.

[Chu36] A. Church. A note on the entscheidungsproblem. *Journal of Symbolic Logic*, 1:40–41, 101–102, 1936.

[CM89] K. Chandy and J. Misra. *Parallel Program Design*. Addison-Wesley, Austin, Texas, May 1989.

[CMB91] O. Coudert, J.-C. Madre, and C. Berthet. Verifying temporal properties of sequential machines without building their state diagrams. In E. M. Clarke and R. P. Kurshan, editors, *Proceedings of the Workshop on Computer-Aided Verification (CAV90)*, volume 3 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, New York, 1991. American Mathematical Society, Springer-Verlag.

[Cob69] A. Cobham. On the base-dependence of sets of numbers recognizable by finite automata. *Mathematical Systems Theory*, 3:186–192, 1969.

[CVWY92] C. Courcoubetis, M. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. *Formal Methods in System Design*, 1:275–288, 1992.

[DAAC89] M. Diaz, J. P. Ansart, P. Azema, and V. Chari. *The Formal Description Technique Estelle*. North-Holland, 1989.

[DDHY92] D. Dill, A. Drexler, A. Hu, and C. Yang. Protocol verification as a hardware design aid. In *1992 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 522–525, Cambridge, MA, October 1992. IEEE Computer Society.

[Dij76] E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, Englewood Cliffs, New-Jersey, 1976.

[Eme90] E. A. Emerson. *Temporal and Modal Logic*, volume B of *Handbook of Theoretical Computer Science*, chapter 16, pages 996–1072. Elsevier, 1990.

[EN94] J. Esparza and M. Nielsen. Decidability issues for Petri nets – a survey. *Bulletin of the EATCS*, 52:245–262, 1994.

[Esp97] J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34(2):85–107, 1997.

[FGM+92] F. Fernandez, H. Garavel, L. Mounier, A. Rasse, C. Rodriguez, and J. Sifakis. A toolbox for the verification of LOTOS programs. In *Proceedings of the 14th International Conference on Software Engineering (ICSE'14)*, Melbourne, Australia, May 1992. ACM.

[Fin90] A. Finkel. Reduction and covering of infinite reachability trees. *Information and Computation*, 89(2):144–179, 1990.

[Fin94] A. Finkel. Decidability of the termination problem for completely specified protocols. *Distributed Computing*, 7(3):129–135, 1994.

[FM96] A. Finkel and O. Marcé. Verification of infinite regular communicating automata. Internal report, ENS Cachan, France, 1996.

[FR79] J. Ferrante and C. W. Rackoff. *The Computational Complexity of Logical Theories*, volume 718 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin-Heidelberg-New York, 1979.

[FR87] A. Finkel and L. Rosier. A survey of FIFO nets. Technical Report 632, University of Montréal, Canada, Département d'Informatique et de Recherche Opérationnelle, October 1987.

[Fra68] J. N. Franklin. *Matrix Theory*. Prentice-Hall Series in Applied Mathematics. Prentice-Hall, 1968.

[FWW97] A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. In *Proceedings of Infinity'97*, 1997.

[GL96] P. Godefroid and D. E. Long. Symbolic protocol verification with queue BDDs. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, pages 198–206, New-Brunswick, New-Jersey, July 1996.

[God96] P. Godefroid. *Partial-order methods for the verification of concurrent systems – An approach to the state-explosion problem*, volume 1032 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[GPVW95] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proceedings of the 15th Workshop on Protocol Specification, Testing, and Verification*, Warsaw, June 1995. North-Holland.

[Gri93] E. Gribomont. Concurrency without toil: A systematic method for parallel program design. *Science of Computer Programming*, 21:1–56, 1993.

[GW93] P. Godefroid and P. Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. *Formal Methods in System Design*, 2(2):149–164, April 1993.

[Hal93] N. Halbwachs. Delay analysis in synchronous programs. In *Proceedings of the 5th Workshop on Computer-Aided Verification*, volume 697 of *Lecture Notes in Computer Science*, pages 333–346, Elounda, Crete, June 1993. Springer-Verlag.

[Har65] M. A. Harrison. *Introduction to switching and automata theory*. McGraw-Hill, New-York, 1965.

[Hau90] D. Hauschildt. *Semilinearity of the Reachability Set is Decidable for Petri Nets*. PhD thesis, Universität Hamburg, May 1990.

[Hen96] T. A. Henzinger. The theory of hybrid automata. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science*, pages 278–292, New Brunswick, New Jersey, July 1996.

[HH94] T. A. Henzinger and P.-H. Ho. Model-checking strategies for linear hybrid systems. Technical Report CSD-TR-94-1437, Cornell University, 1994. Presented at the 7th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (Austin, TX).

[Hig52] G. Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, 2:326–336, 1952.

[HJM94] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial-time algorithm for deciding equivalence of normed context-free processes. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 623–631, Santa Fe, New Mexico, November 1994. IEEE Computer Society Press.

[HK90] Z. Har'El and R. P. Kurshan. Software for analytical development of communication protocols. *AT&T Technical Journal*, 69(1):44–59, 1990.

[HKPV95] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? In *Proceedings of the 27th Annual Symposium on Theory of Computing*, pages 373–382. ACM Press, 1995.

[HNSY94] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994. Special issue for LICS 92.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12:576–583, 1969.

[Hol88] G. Holtzmann. An improved protocol reachability analysis technique. *Software, Practice, and Experience*, 18(2):137–161, 1988.

[Hol90] G. Holtzmann. Algorithms for automated protocol validation. *AT&T Technical Journal, Special Issue on Protocol Specification, Testing, and Verification*, 69(1):32–44, 1990.

[Hol91] G. Holtzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1991.

[Hop71] J. E. Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. *Theory of Machines and Computation*, pages 189–196, 1971.

[HS91] H. Hüttel and C. Stirling. Actions speak louder than words: Proving bisimilarity for context-free processes. In *Proceedings of the 6th Annual IEEE Symposium on Logic in Computer Science*, pages 376–386, Amsterdam, The Netherlands, July 1991. IEEE Computer Society Press.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[IR90] K. Ireland and M. Rosen. *A Classical Introduction to Modern Number Theory*, volume 84 of *Graduate Texts in Mathematics*. Springer-Verlag, second edition, 1990.

[JJ93] T. Jéron and C. Jard. Testing for unboundedness of FIFO channels. *Theoretical Computer Science*, 113:93–117, 1993.

[JN95] N. D. Jones and F. Nielson. *Abstract Interpretation: a semantics-based tool for program analysis*, volume 4 of *Handbook of Logic in Computer Science*, chapter 5, pages 527–636. Clarendon Press, 1995.

[KL93] R. P. Kurshan and L. Lamport. Verification of a multiplier: 64 bits and beyond. In *Proceedings of the 5th International Conference on Computer Aided Verification*, volume 697 of *Lecture Notes in Computer Science*, pages 166–179, Elounda, Greece, 1993. Springer-Verlag.

[KM69] R. M. Karp and R. E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3:147–195, 1969.

[KR78] B. W. Kernighan and D. M. Ritchie. *The C Programming Language*. Prentice-Hall Software Series. Prentice-Hall, Englewood Cliffs, NJ, 1978.

[KSW90] F. Kabanza, J.-M. Stevenne, and P. Wolper. Handling infinite temporal data. In *Proceedings of the 9th ACM Symposium on Principles of Database Systems*, pages 392–403, Nashville, Tennessee, 1990.

[Lam94] J.-L. Lambert. Vector addition systems and semi-linearity. Internal Report, Université de Paris-Nord, 1994.

[Lub84] B. D. Lubachevsky. An approach to automating the verification of compact parallel coordination programs. I. *Acta Informatica*, 21:125–169, 1984.

[LV92] H. Le Verge. A note on Chernikova's algorithm. Research Report 1662, INRIA, Le Chesnay, France, April 1992.

[Mac63] R. MacNaughton. Review of [Büc60]. *Journal of Symbolic Logic*, 28:100–102, 1963.

[Mat94] A. Matos. Periodic sets of integers. *Theoretical Computer Science*, 127:287–312, 1994.

[McC65] E. J. McCluskey. *Introduction to the theory of switching circuits*. McGraw-Hill, New-York, 1965.

[McM93] K. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[MF85] G. Memmi and A. Finkel. An introduction to FIFO nets – monogenous nets: A subclass of FIFO nets. *Theoretical Computer Science*, 35, 1985.

[Mor68] R. Morris. Scatter storage techniques. *Communications of the ACM*, 11(1):38–44, 1968.

[MP86] C. Michaux and F. Point. Les ensembles k-reconnaissables sont définissables dans $\langle \mathbf{N}, +, V_k \rangle$. *Comptes Rendus de l'Académie des Sciences de Paris*, 303:939–942, 1986.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.

[MS85] D. E. Muller and P. E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science*, 37(1):51–75, May 1985.

[Muc91] A. Muchnik. Definable criterion for definability in Presburger arithmetic and its application. Institute of New Technologies, 1991.

[Mul63] D. Muller. Infinite sequences and finite machines. In *Proceedings of the 4th IEEE Symposium on Switching Circuit Theory and Logical Design*, pages 3–16, New-York, 1963.

[MV93] C. Michaux and R. Villemaire. Cobham theorem seen through Büchi theorem. In *Proceedings of ICALP'93*, volume 700 of *Lecture Notes in Computer Science*, pages 325–334. Springer-Verlag, 1993.

[MV96] C. Michaux and R. Villemaire. Presburger arithmetic and recognizability of sets of natural numbers by automata: New proofs of Cobham's and Semenov's theorems. *Annals of Pure and Applied Logic*, 77(3):251–277, February 1996.

[Neu96] P. G. Neumann. Illustrative notes to the public in the use of computer systems and related technology. *ACM SIGSOFT Software Engineering Notes*, 21(1):16–31, January 1996. Quarterly updates available at `ftp://ftp.csl.sri.com/pub/illustrative.PS`.

[Neu97] P. G. Neumann. Computer security in aviation: Vulnerabilities, threats, and risks. In *International Conference on Aviation Safety in the 21st Century*. White House Commission on Safety and Security and George Washington University, January 1997.

[Opp78] D. C. Oppen. A $2^{2^{2^{pn}}}$ upper bound on the complexity of Presburger arithmetic. *Journal of Computer and System Sciences*, 16:323–332, 1978.

[Pac86] J. Pachl. Protocol description and analysis based on a state transition model with channel expressions. In *Proceedings of the 6th international workshop on Protocol Specification, Testing, and Validation*, IFIP'86, Montreal, Quebec, 1986. North-Holland.

[Péc86] J.-P. Pécuchet. On the complementation of Büchi automata. *Theoretical Computer Science*, 47:95–98, 1986.

[Per90] D. Perrin. *Finite Automata*, volume B of *Handbook of Theoretical Computer Science*, chapter 1. Elsevier, 1990.

[Pet62] C. Petri. Kommunikation mit Automaten. Technical report, University of Bonn, 1962.

[Pet81] J. Peterson. *Petri net theory and the modeling of systems*. Prentice-Hall, 1981.

[Pre29] M. Presburger. Über die Volständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du Premier Congrès des Mathématiciens des Pays Slaves*, pages 92–101, Warsaw, Poland, 1929.

[PT87] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal of Computing*, 16(6):973–989, 1987.

[Pug92a] W. Pugh. The Omega Test: A fast and practical integer programming algorithm for dependence analysis. *Communications of the ACM*, pages 102–114, August 1992.

[Pug92b] W. Pugh. A practical algorithm for exact array dependence analysis. *Communications of the ACM*, 35(8):102–114, August 1992.

[Pug94] W. Pugh. Counting solutions to Presburger formulas: How and why. *SIGPLAN*, 94-6/94:121–134, 1994.

[PY97] A. Parashkevov and J. Yantchev. Space efficient reachability analysis through use of pseudo-root states. In *Proceedings of the 3rd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'97)*, volume 1217 of *Lecture Notes in Computer Science*, pages 50–64, Enschede, The Netherlands, April 1997. Springer Verlag.

[QJ95] Y.-M. Quemener and T. Jéron. Model-checking of CTL on infinite Kripke structures defined by simple graph grammars. Technical Report 2563, INRIA, June 1995.

[QJ96] Y.-M. Quemener and T. Jéron. Finitely representing infinite reachability graphs of CFSMs with graph grammars. Technical Report 994, IRISA, March 1996.

[Rei85] W. Reisig. *Petri nets*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1985.

[Saf88] S. Safra. On the complexity of ω-automata. In *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, pages 319–327, 1988.

[Sem77] A. L. Semenov. Presburgerness of predicates regular in two number systems. *Siberian Mathematical Journal*, 18:289–299, 1977.

[ST79] I. Stewart and D. Tall. *Algebraic Number Theory*. Chapman and Hall Mathematics Series. John Wiley & Sons, New-York, 1979.

[SVW87] A. Sisla, M. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.

[Tar83] R. E. Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, 1983.

[Tho90] W. Thomas. *Automata on Infinite Objects*, volume B of *Handbook of Theoretical Computer Science*, chapter 4, pages 133–191. Elsevier, 1990.

[Tur36] A. M. Turing. On computable numbers with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society (Series 2)*, 42:230–265, 1936.

[Val89] A. Valmari. State-space generation with induction. In *Proceedings of the Scandinavian Conference on Artificial Intelligence '89*, pages 99–115, June 1989.

[Val91] A. Valmari. Stubborn sets for reduced state space generation. In *Advances in Petri Nets*, volume 483 of *Lecture Notes in Computer Science*, pages 491–515. Springer-Verlag, 1991.

[Vil92] R. Villemaire. The theory of $\langle \mathbf{N}, +, V_k, V_l \rangle$ is undecidable. *Theoretical Computer Science*, 106:337–349, 1992.

[vLS79] A. van Lamsweerde and M. Sintzoff. Formal derivation of strongly correct concurrent programs. *Acta Informatica*, 12:1–31, 1979.

[VW86] M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 322–331, Cambridge, UK, June 1986.

[VW94] M. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994.

[Wal96] I. Walukiewicz. Pushdown processes: Games and model checking. In *Proceedings of the 8th Workshop on Computer-Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 62–74, New Brunswick, New-Jersey, July/August 1996. Springer-Verlag.

[WB95] P. Wolper and B. Boigelot. An automata-theoretic approach to Presburger arithmetic constraints. In *Proceedings of Static Analysis Symposium*, volume 983 of *Lecture Notes in Computer Science*, pages 21–32, Glasgow, September 1995. Springer-Verlag.

[Win84] G. Winskel. Categories of models for concurrency. In *Seminar on Concurrency*, number 197 in Lecture Notes in Computer Science, pages 246–267. Springer-Verlag, July 1984.

[Wir71] N. Wirth. The programming language Pascal. *Acta Informatica*, 1(1):35–63, 1971.

[WL93] P. Wolper and D. Leroy. Reliable hashing without collision detection. In *Proceedings of Int. Workshop on Computer-Aided Verification*, number 697 in Lecture Notes in Computer Science, Elounda, Crete, June 1993. Springer-Verlag.

[Wol83] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1-2):72–99, January–February 1983.

[Wol86] P. Wolper. Expressing interesting properties of programs in propositional temporal logic. In *Proceedings of the 13th ACM Symposium on Principles of Programming Languages*, pages 184–193, January 1986.