Relations d'implémentation et transformations autorisées d'une spécification LOTOS

Guy Leduc

Chercheur Qualifié du Fonds National belge de la Recherche Scientifique (F.N.R.S.)
Université de Liège
Institut d'Electricité Montefiore, B 28
B-4000 Liège 1
Belgique

RESUME. Nous présentons un cadre général permettant d'étudier le processus d'implémentation d'une spécification formelle. Nous considérons qu'il s'agit d'un processus à étapes par lequel une spécification formelle abstraite est transformée successivement afin d'atteindre une spécification finale compilable et bien adaptée à l'environnement cible. Dans ce contexte, une relation d'implémentation est une relation qui doit lier toute implémentation "conforme" à sa spécification formelle abstraite. En d'autres termes, la relation d'implémentation a pour but d'exprimer formellement la notion de conformité. Le cadre général que nous présentons permet de caractériser exactement les transformations autorisées à chaque étape pour une relation d'implémentation donnée. Ce cadre est essentiel lorsque la relation d'implémentation est non transitive. Dans la deuxième partie de cet article, nous présentons plusieurs relations candidates au rôle de relation d'implémentation : conf, red et ext. Ensuite, pour chacune d'elles, nous caractérisons les transformations autorisées correspondantes. Ces résultats nous permettent aussi de proposer une nouvelle relation, appelée conf, pouvant jouer le rôle de relation d'implémentation. Enfin, nous montrons que ces idées sur le processus d'implémentation peuvent être étendues afin d'étudier le lien formel devant exister entre un protocole et un service.*

ABSTRACT. A general framework is presented for studying the implementation process, which is considered as a stepwise process in which an abstract specification is successively transformed to reach a final compilable specification well-adapted to a target environment. In this context, an implementation relation is referred to as a relation which should link any "valid" implementation to its abstract formal specification. In other words, the implementation relation is intended to express formally the notion of validity. The general framework presented here allows the exact characterization of the transformations which are allowed at each step, for a given implementation relation. This framework is essential when the implementation relation is non-transitive. In the second part of the paper, several relations which may play the role of an implementation relation are presented : conf, red and ext. Then, for each one, the corresponding allowed transformations are characterized. These results also lead naturally to a new relation, denoted conf, which may also be used as an implementation relation. Finally, we show that these ideas on the implementation process may be extended to study the formal link which should hold between a protocol and a service.*

MOTS-CLES/KEYWORDS. LOTOS, relation d'implémentation, raffinement, processus d'implémentation, transformation, conformité, spécification, implémentation, abstraction, FDT, algèbre de processus, implementation relation, refinement, implementation process, conformance, specification, implementation, process algebra.

Réseaux et informatique répartie. Volume 1 - n° 1/1991, P. 59 à 86

1. Introduction

Dans cet article, nous commençons par rappeler les éléments et résultats principaux d'un cadre mathématique général publié dans [Led 90, Led 91] et permettant de raisonner de façon précise sur le processus d'implémentation d'une spécification formelle. Ce processus a pour but de transformer progressivement une spécification abstraite afin d'obtenir une spécification d'implémentation. Durant ce processus, diverses activités de synthèse et de vérification se côtoient et se complètent, et plusieurs spécifications intermédiaires sont produites successivement par transformations de la spécification précédente [Cos 88a, Cos 88b, Bog 89]. On comprend que ces transformations doivent respecter certaines règles afin d'assurer que le stade final, souvent appelé *spécification d'implémentation* ou plus simplement *implémentation*, soit conforme à la spécification initiale.

Une des difficultés majeures dans ce contexte est la formalisation de cette notion de conformité, i.e. la nature du lien devant exister entre les stades finals autorisés et le stade initial de ce processus à étapes. C'est cette notion de conformité qui définit implicitement les transformations autorisées. Nous allons rendre cette dépendance plus explicite.

La conformité en tant qu'équivalence

Lorsque l'on utilise des techniques algébriques, on requiert classiquement qu'une relation d'équivalence soit préservée tout au long du processus d'implémentation. Les relations d'équivalence jouent un rôle central dans les langages basés sur des algèbres de processus comme CCS [Mil 89], ACP [BeK 85] or LOTOS [ISO 8807, BoB 87] pour raisonner sur les systèmes et analyser leurs propriétés. Plusieurs notions d'équivalence ont été proposées; ce qui n'est pas surprenant car il existe beaucoup de propriétés qui peuvent être significatives pour l'analyse des systèmes distribués [dNi 87]. Cependant, ces équivalences sont pratiquement toujours basées sur certains critères observationnels, i.e. deux systèmes sont considérés équivalents si, et seulement si, ils sont indistinguables par certains types d'observations extérieures. Ces équivalences permettent de faire abstraction de certains détails internes. Il subsiste néanmoins toute une série de telles équivalences [dNi 87, vGl 90, Led 90] dont les plus connues sont certainement l'équivalence observationnelle [Mil 89] et différentes équivalences de test [dNi 84, BSS 87, Hen 88].

L'approche consistant à préserver une certaine équivalence tout au long des transformations, permet d'assurer que toute implémentation ainsi obtenue se comportera (extérieurement) de la même façon que la spécification elle-même. La structure de la spécification peut bien sûr changer durant le processus de raffinement afin de se rapprocher de la structure d'une implémentation, mais rien ne change extérieurement, i.e. d'autres systèmes ne seraient pas capables de faire la distinction entre les différentes implémentations possibles de la spécification, en communiquant avec elles. En LOTOS, plusieurs styles de spécification [VSv 88] permettent de décrire un même processus de diverses manières. Le processus d'implémentation peut dès lors être considéré comme une succession de transformations de parties de la spécification d'un style vers un autre [vEi 89].

La conformité en tant que relation non nécessairement symétrique

Même si cette vision est intéressante, il en existe une autre qui, selon nous, est plus appropriée et prend en compte le caractère asymétrique du processus d'implémentation. Au lieu de considérer que le stade final doive être équivalent à la spécification, l'idée consiste à définir une relation moins restrictive et habituellement asymétrique. Ces relations sont souvent appelées des *relations d'implémentation*.

Ces relations ont été moins étudiées que les équivalences dans le contexte des techniques algébriques. Il n'existe d'ailleurs pas d'opinion bien établie sur la nature de ces relations d'implémentation; seules quelques tendances existent. Par exemple, il est souvent admis qu'une implémentation puisse être plus déterministe qu'une spécification. Selon cette vision, une relation d'implémentation serait plutôt considérée comme un préordre (i.e. une relation réflexive et transitive). Un préordre, ayant un caractère asymétrique, définit un ordre partiel sur les systèmes. Si ce préordre est bien choisi, il peut être interprété comme une *relation d'implémentation*, i.e. si deux systèmes A et B sont tels que B *est moins (abstrait) que* A selon cet ordre, alors cela signifie que dans un certain sens B *implémente* A, ou B *est une implémentation conforme de* A. Par exemple, un critère pouvant être exprimé formellement par une telle relation est la réduction du non-déterminisme, i.e. B est une implémentation conforme de A si, et seulement si, B est une transformation de A par laquelle certains choix (volontairement) non déterministes de A ont été résolus.

Quelques relations d'implémentation basées sur cette idée ont été définies. Ce sont en général - mais pas toujours - des préordres. En TCSP [BHR 84, Hoa 85], une telle relation avait déjà été introduite comme préordre de l'équivalence des échecs. En CCS [dNH 84], d'autres préordres associés à plusieurs équivalences de test ont été introduites. Finalement, en LOTOS [BSS 87], des préordres de l'équivalence de test, ainsi qu'un relation de conformité, ont été définies. On trouvera dans [dNi 87, Led 90] les principales équivalences et leurs préordres associés, ainsi que de nombreux liens entre des relations, apparemment différentes, basées sur les traces, les refus ou les acceptations, et les divergences.

Le concept de relation d'implémentation a aussi été introduit dans d'autres modèles formels. Avec des spécifications en logique, B est normalement considéré comme une implémentation conforme de A si, et seulement si, $B \Rightarrow A$, i.e. B satisfait plus de propriétés que A [ChM 89]. Avec des machines à états ou des automates E/S, une spécification B peut être considérée comme une implémentation d'une spécification A ssi il existe une application appropriée de B vers A [LyF 81, Lam 83, LaS 84, LyT 87, AbL 88, Mer 89]. Avec des systèmes de transitions modaux [Lar 89] (i.e. une extension des systèmes de transitions étiquetés avec des transitions nécessaires et des transition admissibles), B est considéré comme une implémentation de A ssi il existe une relation de raffinement entre B et A.

Contenu de l'article

Dans le cadre théorique que nous présentons, nous considérons une relation d'implémentation générique, appelée *relation d'implémentation de référence* qui est supposée exprimer formellement cette notion de conformité.

L'article se focalise alors sur le problème suivant :

> Etant donné une relation d'implémentation de référence, quelles sont les restrictions qu'elle impose sur les transformations autorisées entre deux stades intermédiaires de spécification dans le processus d'implémentation ? En d'autres termes, comment pouvons-nous caractériser les *degrés de liberté* de l'implémenteur à chaque étape en conséquence du choix de la relation d'implémentation ?

Dans la deuxième partie de cet article, nous présentons plusieurs relations candidates au rôle de relation d'implémentation, à savoir *conf*, *red* et *ext*. Ensuite, pour chacune d'elles nous caractérisons les transformations autorisées correspondantes.

Ces résultats nous amènent à proposer une nouvelle relation, *conf**, pouvant jouer le rôle de relation d'implémentation.

Nous montrons enfin que ces idées peuvent être étendues afin d'étudier le lien formel devant exister entre un service et un protocole.

Nous avons essayé de faire un minimum d'hypothèses sur la relation d'implémentation de référence. Ainsi, nous supposons qu'elle est réflexive, mais pas nécessairement symétrique, ni transitive. La réflexivité est justifiée par le fait que la spécification est une implémentation conforme d'elle-même. La symétrie n'est sûrement pas requise étant donné que la spécification et l'implémentation ne sont pas interchangeable en général. La transitivité est plus discutable mais ne doit pas être requise selon nous. Ce point sera discuté en détail dans la section 2.

Il est important de noter que ces relations d'implémentation sont ici étudiées per se, et que nous n'avons pas l'intention de définir des règles de transformation qui pourraient être appliquées dans le processus d'implémentation afin d'automatiser partiellement l'obtention d'une implémentation conforme à partir d'une spécification. Nos résultats ont cependant pour but d'aider à comprendre la nature de la relation qui doit exister, et donc être vérifiée, entre deux étapes intermédiaires en fonction de la relation d'implémentation de référence choisie. Les résultats ont été étendus afin de permettre des transformations locales, i.e. des transformations dans un certain contexte LOTOS. Ceci conduit à considérer la monotonicité des opérateurs LOTOS vis-à-vis de ces relations, ou encore le caractère précongruent de ces relations.

2. Concept de relation d'implémentation

Dans cette section, nous raisonnons de façon très générale sur le concept de relation d'implémentation et sur son rôle. Un telle relation, appelée *imp* dans la suite, a pour but d'exprimer les conditions qu'une implémentation doit remplir pour être considérée conforme à la spécification. Nous ne proposerons pas d'exemples de relation *imp* dans cette section, mais nous présenterons un cadre générique reposant sur *imp* .

Une spécification est une description non ambiguë d'un objet à un niveau relativement élevé d'abstraction. Très souvent, l'objectif n'est pas d'implémenter cet objet tel quel; dans certains cas, c'est tout simplement impossible en raison du niveau d'abstraction trop élevé ou du caractère non constructif de la spécification ou du modèle sémantique sous-jacent. Ceci justifie en pratique le besoin d'utiliser un processus à étapes partant d'une spécification abstraite et se terminant à l'implémentation.

Avant d'aller plus loin, remarquons que, comme indiqué dans [BrS 86], le terme *implémentation* ne signifie pas ici une implémentation physique mais plutôt une spécification à un niveau d'abstraction très bas, i.e. un modèle d'une implémentation physique. Ceci rend possible l'inclusion d'une notion d'implémentation dans une théorie formelle. Le lien entre la réalité physique et son modèle reste bien sûr informel par nature.

Les transformations entre différents niveaux de spécification doivent préserver certaines caractéristiques tout en en changeant d'autres. Ainsi, une spécification moins abstraite doit respecter dans un certain sens une spécification plus abstraite, mais ceci ne signifie pas nécessairement qu'elle doivent avoir exactement le même comportement. Par exemple, la fonctionnalité peut être réduite ou étendue, le non-déterminisme peut être réduit partiellement, …

Cette notion de conformité d'une implémentation vis-à-vis d'une spécification n'est pas exprimée par la spécification elle-même. Ceci n'est pas surprenant et probablement souhaitable puisque cela permet la définition de plusieurs notions de ce type selon l'usage que l'on envisage. La même approche a conduit à la définition de plusieurs relations d'équivalence entre spécifications. En conséquence, une relation formelle, *imp*, doit être définie afin de décider si une implémentation donnée est conforme à une spécification. Le couple "spécification + relation d'implémentation" définit implicitement l'ensemble (infini) des implémentations conformes à la spécification. Ici encore le choix d'une relation d'équivalence est similaire, en ce sens que le couple "spécification + équivalence" définit implicitement tout un ensemble de spécifications (équivalentes). Nous considérons toutefois que la notion de relation d'implémentation a une nature plus fondamentale car elle induit naturellement, quelle qu'elle soit (cf. définition 2.1), une équivalence.

Quelques propriétés de *imp* peuvent être établies a priori.

Imp doit être réflexive car la spécification est une implémentation conforme d'elle-même. En conséquence, nous considérerons dans le reste de cet article que *imp* est réflexive. Si *Id* désigne la relation d'identité, qui est la plus petite relation réflexive, nous avons $Id \subseteq imp$.

De plus, il n'est pas nécessaire que *imp* soit symétrique puisque l'implémentation et la spécification ne sont pas interchangeables en général.

Un point plus délicat est la transitivité de *imp*, i.e. doit-on exiger qu'une implémentation conforme d'une implémentation conforme soit toujours une implémentation conforme ? Si *imp* n'est pas transitive, une implémentation conforme ne peut pas être utilisée comme une spécification intermédiaire; ce qui n'est pas son rôle de toute façon. Nous reviendrons sur ce point plus loin.

Nous allons montrer comment *imp* induit naturellement une équivalence, et comment *imp* restreint la liberté de l'implémenteur à chaque étape de transformation. Ce dernier point sera formalisé par le biais d'une autre relation, appelée *imp-restr*, qui doit être vérifiée entre deux étapes intermédiaires. Notons déjà que *imp-restr* sera naturellement transitive, quelle que soit *imp* .

Ces résultats seront étendus aux transformations locales, i.e. dans tout contexte LOTOS; ce qui conduit à étudier la monotonicité des opérateurs LOTOS vis-à-vis de *imp-restr* (et non *imp*), ou encore, le caractère précongruent de *imp-restr* dans les contextes LOTOS.

Nous commencerons la présentation du cadre théorique par la relation d'équivalence induite par *imp* et appelée *imp–eq*.

Définition 2.1

S_1 *imp-eq* S_2 *ssi* $\{I \mid I \text{ imp } S_1\} = \{I \mid I \text{ imp } S_2\}$

où $\{I \mid I \text{ imp } S\}$ désigne l'ensemble des I qui sont des implémentations conformes de S selon la relation *imp*.

Intuitivement, deux spécifications sont équivalentes si, et seulement si, elles déterminent exactement le même ensemble d'implémentations conformes au sens de *imp*.

Il est évident que *imp-eq* est réflexive, symétrique et transitive. *Imp-eq* est donc une équivalence quelle que soit *imp*.

Si *imp* est considérée comme relation de référence, cette équivalence joue un rôle fondamental en ce sens qu'aucune distinction ne doit être faite entre deux spécifications autorisant les mêmes implémentations conformes. L'équivalence est déduite naturellement de la relation d'implémentation. Le contraire n'est pas toujours possible.

On pourrait penser que *imp-eq* est l'équivalence définie par $\text{imp} \cap \text{imp}^{-1}$, i.e. deux spécifications sont équivalentes si, et seulement si, chacune d'elles est une implémentation conforme de l'autre. En fait, $\text{imp} \cap \text{imp}^{-1}$ n'est pas nécessairement une équivalence, et nous avons le résultat suivant.

Proposition 2.2 [Led 91]

$\text{imp-eq} \subseteq \text{imp} \cap \text{imp}^{-1}$

Nous introduisons maintenant une autre relation, appelée *imp-restr*, qui joue un rôle central dans le processus de transformations à étapes.

L'idée de base est de considérer que chaque étape de transformation ne peut que réduire (ou préserver) l'ensemble des implémentations conformes à une spécification. Le stade final étant l'implémentation voulue qui est dès lors conforme par construction.

Définition 2.3

S_2 *imp-restr* S_1 *ssi* $\{I \mid I \text{ imp } S_2\} \subseteq \{I \mid I \text{ imp } S_1\}$

Le rôle de cette nouvelle relation sera plus clair dans la suite. Remarquons déjà que si nous remplaçons la spécification S_1 par une autre spécification S_2 (plus proche d'une implémentation) de sorte que S_2 *imp-restr* S_1, nous garantissons que les implémentations conformes restantes sont conformes à S_1.

Propositions 2.4 [Led 91]

(i) *imp-restr* est un préordre (i.e. une relation réflexive et transitive)

(ii) $\text{imp-restr} \supseteq \text{imp-eq}$

(iii) $\text{imp-restr} \cap \text{imp-restr}^{-1} = \text{imp-eq}$

Nous pouvons maintenant revenir à la question de la transitivité de *imp*, i.e. *imp* doit-elle être transitive ? Nous pensons que les arguments en faveur de la transitivité de *imp* proviennent de la confusion entre *imp* et *imp-restr*. Souvenons-nous que *imp* exprime le lien formel entre deux objets : une spécification et une implémentation; la transitivité n'a en fait pas de sens à ce niveau puisqu'aucune notion de composition n'intervient. Au contraire, *imp-restr* est intrinsèquement liée au découpage du processus d'implémentation en étapes, puisque B *imp-restr* A signifie que B est plus proche d'une implémentation que A, ou que B autorise moins d'implémentations conformes que A. C'est pour cette raison qu'il n'est pas surprenant que *imp-restr* soit naturellement transitive quelle que soit *imp*.

La relation transitive *imp-restr* résulte du choix de *imp*, qui ne doit pas nécessairement être elle-même transitive. Ces deux relations seront reliées davantage par les propositions suivantes.

Propositions 2.5 [Led 91]

(i) *imp-restr* $\subseteq$ *imp*

(ii) *imp* o *imp-restr* = *imp*

Le symbole "o" indique la composition de deux relations.

Si *imp* est transitive, nous obtenons les résultats plus forts suivants.

Propositions 2.6 [Led 91]

(i) *imp* est transitive $\Rightarrow$ *imp-eq* = *imp* $\cap$ *imp*$^{-1}$

(ii) *imp* est transitive $\Leftrightarrow$ *imp-restr* = *imp*

Dans ce cas, deux spécifications sont équivalentes si, et seulement si, chacune d'elles est une implémentation conforme de l'autre.

Ainsi, quand *imp* est transitive, *imp-eq* peut être définie plus simplement comme suit :

$$S_1 \; \textit{imp-eq} \; S_2 \quad ssi \quad S_1 \; \textit{imp} \; S_2 \; \wedge \; S_2 \; \textit{imp} \; S_1.$$

De plus, dans ce cas, il n'y a pas de distinction entre *imp* et *imp-restr*. Ceci s'explique par le fait que si la relation d'implémentation est transitive, elle caractérise également les transformations autorisées à chaque étape.

3. Transformations autorisées par une relation d'implémentation

Dans cette section, nous considérons l'activité de synthèse d'une implémentation à partir d'une spécification. Comme nous l'avons dit, cette activité est supposée être une succession de transformations proposées par l'implémenteur de façon plus ou moins empirique, mais qui doivent préserver certaines propriétés formalisées par une relation convenable.

Considérons qu'une transformation proposée par l'implémenteur respecte une certaine relation *R*. Cela signifie que si S et S' désignent les spécifications avant et après transformation, nous obtenons S' *R* S. Nous pouvons imaginer plusieurs relations *R* exprimant des critères pratiques différents : certaines peuvent permettre la réduction du non déterminisme, d'autre une extension contrôlée de la fonctionnalité, …

Toutefois, si nous exigeons que chaque transformation ne puisse que réduire ou conserver l'ensemble des implémentations conformes, le choix de $\underline{imp}$ impose des contraintes sur les relations $\underline{R}$ autorisées, et par la même sur les transformations autorisées.

Définitions 3.1

(i) Une transformation de *S* en *S'* est une $\underline{R}$-transformation ssi *S'* $\underline{R}$ *S*.

(ii) Une $\underline{R}$-transformation est autorisée par $\underline{imp}$ ssi

$\forall S, S'$, *on a* $S' \ \underline{R} \ S \Rightarrow \{I \mid I \ \underline{imp} \ S'\} \subseteq \{ I \mid I \ \underline{imp} \ S\}$,

ou encore, ssi $\underline{R} \subseteq \underline{imp\text{-}restr}$ par définition de $\underline{imp\text{-}restr}$.

Intuitivement, $\underline{R}$ peut être utilisée pour caractériser certaines transformations autorisées sur *S* (appelées des $\underline{R}$-transformations) si, et seulement si, la spécification obtenue *S'* n'admet pas de nouvelles implémentations conformes (figure 3.1).

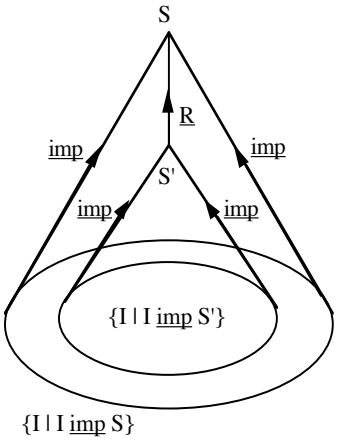

Figure 3.1 : Une transformation

La définition 3.1 fait apparaître que $\underline{imp\text{-}restr}$ et les relations plus fortes que $\underline{imp\text{-}restr}$ jouent un rôle important dans le processus d'implémentation. Nous allons en étudier quelques propriétés.

La proposition suivante montre une autre façon de caractériser les $\underline{R}$-transformations autorisées.

Proposition 3.2 [Led 91]

$\underline{R} \subseteq \underline{imp\text{-}restr} \Leftrightarrow \underline{imp} \circ \underline{R} \subseteq \underline{imp}$

Si nous nous restreignons aux relations réflexives plus fortes que $\underline{imp\text{-}restr}$, nous obtenons les résultats suivants.

Proposition 3.3 [Led 91]

$\underline{Id} \subseteq \underline{R} \Rightarrow (\underline{R} \subseteq \underline{imp\text{-}restr} \Leftrightarrow \underline{imp} \circ \underline{R} = \underline{imp})$

Corollaire 3.4 [Led 91]

$\underline{imp\text{-}restr}$ est la plus faible relation $\underline{R}$ telle que $\underline{imp} \circ \underline{R} = \underline{imp}$

Proposition 3.5 [Led 91]

$\underline{imp}$ est transitive $\Leftrightarrow (\underline{Id} \subseteq \underline{R} \Rightarrow (\underline{R} \subseteq \underline{imp} \Leftrightarrow \underline{imp} \circ \underline{R} = \underline{imp}))$

A partir de la définition 3.1 (ii), les propositions 3.2 et 3.3 peuvent être exprimées comme suit.

Proposition 3.6
Une $\underline{R}$-transformation est autorisée par $\underline{imp}$ ssi $\underline{imp}\ o\ \underline{R} \subseteq \underline{imp}$.

Si $\underline{R}$ est réflexive, nous obtenons le résultat plus fort suivant.

Proposition 3.7
Si $\underline{R}$ est une relation réflexive, alors
une $\underline{R}$-transformation est autorisée par $\underline{imp}$ ssi $\underline{imp}\ o\ \underline{R} = \underline{imp}$.

4. Transformations locales

Dans la section 3, nous avons implicitement supposé que nous appliquions les transformations à la spécification tout entière. En pratique toutefois, il est plus réaliste de considérer que les transformations successives porterons sur des parties de la spécification : un processus particulier sera transformé alors que le reste de la spécification sera inchangé.

Pour traité ce problème en LOTOS, nous utiliserons le concept de contexte et nous étudierons les contraintes supplémentaires à prendre en compte pour appliquer des transformations dans certains contextes LOTOS. Avant de donner une définition précise de la notion de contexte, notons qu'un contexte est construit comme une expression de comportement LOTOS, à l'aide des opérateurs du langage. La table 4.1 donne la sémantique opérationnelle des principaux opérateurs LOTOS : le séquencement (;), le choix ([]), le parallélisme (|[γ]|) et l'abstraction (hide).

$a; B \xrightarrow{a} B$	$i; B \xrightarrow{i} B$
$\dfrac{B_1 \xrightarrow{a} B_1'}{B_1 \,[]\, B_2 \xrightarrow{a} B_1'}$	$\dfrac{B_2 \xrightarrow{a} B_2'}{B_1 \,[]\, B_2 \xrightarrow{a} B_2'}$
$\dfrac{B_1 \xrightarrow{a} B_1' \quad (a \notin \gamma)}{B_1 \,\|[\gamma]\|\, B_2 \xrightarrow{a} B_1' \,\|[\gamma]\|\, B_2}$	$\dfrac{B_2 \xrightarrow{a} B_2' \quad (a \notin \gamma)}{B_1 \,\|[\gamma]\|\, B_2 \xrightarrow{a} B_1 \,\|[\gamma]\|\, B_2'}$
$\dfrac{B_1 \xrightarrow{a} B_1',\ B_2 \xrightarrow{a} B_2' \quad (a \in \gamma)}{B_1 \,\|[\gamma]\|\, B_2 \xrightarrow{a} B_1' \,\|[\gamma]\|\, B_2'}$	
$\dfrac{B \xrightarrow{a} B' \quad (a \notin \gamma)}{hide\ \gamma\ in\ B \xrightarrow{a} hide\ \gamma\ in\ B'}$	$\dfrac{B \xrightarrow{a} B' \quad (a \in \gamma)}{hide\ \gamma\ in\ B \xrightarrow{i} hide\ \gamma\ in\ B'}$

Table 4.1 : principaux opérateurs de LOTOS

Supposons que notre spécification ait la structure suivante, où γ est une liste de points d'interaction :

S := (hide γ in (S_1 |[γ]| Q))

Supposons que nous décidions de transformer S_1 en un processus S_2, nous obtenons ainsi une spécification *S' := (hide γ in (S_2 |[γ]| Q))*. Cette transformation est décrite à la figure 4.2

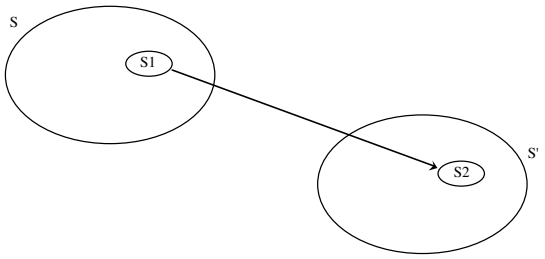

Figure 4.2 : Transformations locales

Est-il suffisant que S_2 *imp-restr* S_1 pour que *S' imp-restr S ?*

Nous allons voir que la réponse est négative et qu'il est nécessaire de respecter localement des règles plus strictes caractérisées par une relation plus forte que *imp-restr*.

Cette relation sera présentée après un bref rappel de la notion de contexte et de précongruence.

Définition 4.1

Un *contexte C [.]* est une expression de comportement contenant un paramètre formel '[.]', appelé un trou.

Une expression de comportement *C [B]* est obtenue à partir de *C [.]* en remplaçant chaque occurrence du trou '[.]' par *B*.

Par exemple, si *C [.] = hide a in (A || [.])*, alors *C [B] = hide a in (A || B)*.

Définition 4.2

La plus faible précongruence plus forte que *R* est appelée *cR* et est définie comme suit :

∀ P, Q, on a P cR Q ssi ∀ contexte C [.], on a C [P] R C [Q].

Il découle immédiatement de la définition 4.2 que la relation à préserver localement pour une relation d'implémentation donnée *imp* est la relation *cimp-restr* définie ci-dessous.

Définition 4.3

La plus faible précongruence plus forte que *imp-restr* est appelée *cimp-restr* et est définie comme suit :

∀ P, Q, on a P cimp-restr Q ssi ∀ contexte C [.], on a C [P] imp-restr C [Q].

Nous terminerons cette section par une dernière définition.

Définition 4.4
Soit $\underline{R}$ une relation réflexive,
une $\underline{R}$-transformation est autorisée *localement* par $\underline{imp}$ ssi
$\forall P, Q$, on a $P \underline{R} Q \Rightarrow \forall C[.], C[P] \underline{imp\text{-}restr} C[Q]$.

Informellement, une $\underline{R}$-transformation est autorisée *localement* par $\underline{imp}$, si, et seulement si, lorsqu'on limite les remplacements d'un quelconque sous-processus Q d'une spécification $C[Q]$ par un quelconque sous-processus P tel que $P \underline{R} Q$, alors la nouvelle spécification $C[P]$ est bien une spécification plus concrète que $C[Q]$, i.e. $C[P] \underline{imp\text{-}restr} C[Q]$.

Notons que c'est la relation $\underline{imp\text{-}restr}$, et non $\underline{imp}$, qui doit être utilisée dans cette définition comme nous l'avons étudié plus haut.

A partir des résultats obtenus dans la section 3, les propositions suivantes permettent de donner des définitions plus utilisables que 4.4. lorsque $\underline{R}$ a certaines propriétés telles que la réflexivité ou la précongruence.

Proposition 4.5 [Led 91]
Soit $\underline{R}$ une relation réflexive,
une $\underline{R}$-transformation est autorisée localement par $\underline{imp}$ ssi $R \subseteq \underline{cimp\text{-}restr}$

Proposition 4.6 [Led 91]
Soit $\underline{R}$ une précongruence,
une $\underline{R}$-transformation est autorisée localement par $\underline{imp}$ ssi $R \subseteq \underline{imp\text{-}restr}$.

Proposition 4.7 [Led 91]
Soit $\underline{R}$ une précongruence,
une $\underline{R}$-transformation est autorisée localement par $\underline{imp}$ ssi $\underline{imp} \; o \; \underline{R} = \underline{imp}$

5. Définitions et propriétés de quelques relations

En LOTOS, quelques relations candidates au rôle de relation d'implémentation, $\underline{imp}$, ont été proposées dans [BrS 86, BSS 87]. Nous les rappelons brièvement ci-dessous en utilisant un formalisme de traces et refus similaire à celui de TCSP [Hoa 85].

Notations 5.1
L est un alphabet d'actions observables, et i est l'action interne (i.e. non observable).
$P -a\rightarrow P'$ signifie que le processus P peut effectuer l'action a et, une fois cette action faite, se comporter ultérieurement comme le processus P'.
$P -i^k\rightarrow P'$ signifie que le processus P peut effectuer la séquence de k actions internes et, une fois cette séquence faite, se comporter ultérieurement comme le processus P'.
$P -a.b\rightarrow P'$ signifie $\exists P''$, *tel que* $P -a\rightarrow P'' \; \wedge \; P'' -b\rightarrow P'$.
$P =a\Rightarrow P'$ où $a \in L$, signifie $\exists k_0, k_1 \in N$, tels que $P -i^{k_0}.a.i^{k_1}\rightarrow P'$

$P{=}a{\Rightarrow}$ où $a \in L$, signifie que $\exists\ P'$, tel que $P{=}a{\Rightarrow}P'$, i.e. P peut accepter l'action a.

$P{=}a{\neq}>$ où $a \in L$, signifie $\neg\ (P{=}a{\Rightarrow})$, i.e. P ne peut pas accepter (ou doit refuser) l'action a.

$P{=}\sigma{\Rightarrow}P'$ signifie que le processus P peut effectuer la séquence d'actions observables σ et, une fois cette séquence faite, se comporter ultérieurement comme le processus P'. Plus précisément,

si $\sigma = a_1..a_n$ où $a_1, \ldots a_n \in L$:

$$\exists\, k_0, \ldots k_n \in N, \text{ tel que } P \xrightarrow{} i^{k_0}.a_1.i^{k_1}.a_2 \ldots a_n.i^{k_n} \rightarrow P'$$

$P{=}\sigma{\Rightarrow}$ signifie que $\exists\ P'$, tel que $P{=}\sigma{\Rightarrow}P'$

$P\ after\ \sigma = \{P' \mid P{=}\sigma{\Rightarrow}P'\}$,

i.e. l'ensemble de toutes les expressions de comportement (ou états) accessibles à partir de P par la séquence σ.

$Tr\ (P)$ est l'ensemble des traces de P, i.e. $\{\sigma \mid P{=}\sigma{\Rightarrow}\}$; $Tr\ (P)$ est un sous-ensemble de L^*.

$Ref\ (P, \sigma)$ est l'ensemble des ensembles de refus de P après la trace σ, i.e.

$Ref\ (P, \sigma) = \{X \mid \exists\, P' \in P\ after\ \sigma, tel\ que\ P'{=}a{\neq}>,\ \forall\, a \in X\};$

$Ref\ (P, \sigma)$ est un ensemble d'ensembles et un sous-ensemble de $\wp\ (L)$, le "power set" de L, i.e. l'ensemble des sous-ensembles de L. Un ensemble $X \subseteq L$ appartient à $Ref\ (P, \sigma)$ si, et seulement si, P peut effectuer la trace σ et, une fois cette séquence faite, refuser toute action de l'ensemble X. On notera que la définition de $Ref\ (P, \sigma)$ est telle que si $X \in Ref\ (P, \sigma)$ et si $Y \subseteq X$, alors $Y \in Ref\ (P, \sigma)$.

Quelques interprétations possibles de la notion de conformité ont été présentées et formalisées dans [BrS 86, BSS 87] par trois relations de base : *conf*, *red* et *ext* , et deux relations associées : *cred* et *cext*. Quelques relations d'équivalence ont aussi été proposées : *te* et *tc*. Nous en donnons les définitions précises ci-dessous.

Définitions 5.2

Soient les deux processus P_1 and P_2 .

P_1 *conf* P_2 *ssi* $\forall\ \sigma \in Tr\ (P_2)$, *on a* $Ref\ (P_1, \sigma) \subseteq Ref\ (P_2, \sigma)$

ou de façon équivalente,

ssi $\forall\ \sigma \in Tr\ (P_2) \cap Tr\ (P_1)$, *on a* $Ref\ (P_1, \sigma) \subseteq Ref\ (P_2, \sigma)$

puisque si $\sigma \in Tr\ (P_2) - Tr\ (P_1)$, alors $Ref\ (P_1, \sigma) = \varnothing$

Intuitivement, P_1 *conf* P_2 si, et seulement si, placé dans tout environnement dont les traces sont limitées à celles de P_2, P_1 ne peut pas être bloqué quand P_2 ne peut pas l'être. Autrement dit, P_1 se bloque moins souvent que P_2 dans un tel environnement. Cette relation a été choisie pour modéliser formellement la notion de conformité aux tests dans [Bri 88], et a été appelée pour cette raison la relation de *conformité*.

P_1 *red* P_2 ssi (i) $Tr\ (P_1) \subseteq Tr\ (P_2)$, et

(ii) P_1 *conf* P_2

Intuitivement, si P_1 *red* P_2, P_1 a moins de traces que P_2, et même dans un environnement dont les traces sont limitées à celles

de P_1, P_1 se bloque moins souvent que P_2. $\underline{Red}$ est la relation de *réduction*.

$P_1 \underline{ext} P_2$ ssi (i) $Tr(P_1) \supseteq Tr(P_2)$, et
(ii) $P_1 \underline{conf} P_2$

Intuitivement, si $P_1 \underline{ext} P_2$, P_1 a plus de traces que P_2, et dans un environnement dont les traces sont limitées à celles de P_2, il se bloque moins souvent que P_2. $\underline{Ext}$ est la relation d'*extension*.

$P_1 \underline{cred} P_2$ ssi (i) $P_1 \underline{red} P_2$
(ii) $Stable(P_2) \Rightarrow Stable(P_1)$

où un processus est stable s'il ne peut effectuer aucune action interne dans l'état initial; plus formellement, $Stable(P)$ ssi $\neg\ \exists P'$, *tel que* $P \xrightarrow{i} P'$;

$\underline{cred}$ est la plus faible précongruence plus forte que $\underline{red}$ [BrS 86]. [1]

$\underline{cext} = \underline{ext} \cap \underline{cred}$

$\underline{cext}$ est la plus faible précongruence plus forte que $\underline{ext}$ [BrS 86]. [2]

$\underline{te} = \underline{red} \cap \underline{red}^{-1} = \underline{ext} \cap \underline{ext}^{-1}$ C'est l'équivalence de test.

$\underline{tc} = \underline{cred} \cap \underline{cred}^{-1} = \underline{cext} \cap \underline{cext}^{-1}$

$\underline{tc}$ est la plus faible congruence plus forte que $\underline{te}$ [BrS 86]. [3]

Propositions 5.3

(i) $\underline{conf} \supset \underline{red}$ [BrS 86]
(ii) $\underline{conf} \supset \underline{ext}$ [BrS 86]
(iii) $\underline{conf} = \underline{red} \circ \underline{ext}$ [BrS 86, Led 90]
(iv) $\underline{conf} \circ \underline{ext} = \underline{conf} = \underline{red} \circ \underline{conf}$ [Led 90]

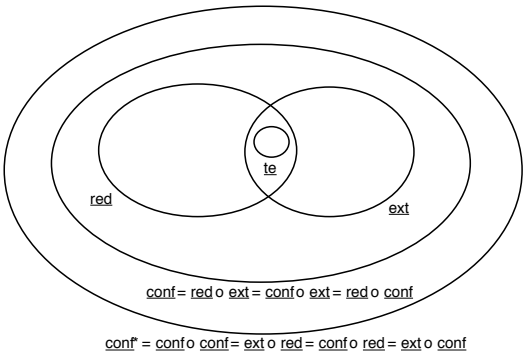

Figure 5.1 : Relations entre $\underline{te}$, $\underline{red}$, $\underline{ext}$, $\underline{conf}$, $\underline{conf}^*$

---

[1] à condition d'exclure les contextes d'abstraction (opérateurs "hide") créant des divergences. Une divergence est une séquence infinie d'action internes. Ce problème est traité en détail dans [Led 90].

[2] cf note 1

[3] cf note 1

Propositions 5.4 [Led 91]

(i) $\underline{conf} \subset \underline{ext}\ o\ \underline{red}$

(ii) $\underline{ext}\ o\ \underline{conf} = \underline{ext}\ o\ \underline{red}$

(iii) $\underline{conf}\ o\ \underline{conf} = \underline{conf}\ o\ \underline{red}$

(iv) $\underline{conf}\ o\ \underline{red} = \underline{ext}\ o\ \underline{red}$

(v) $\underline{conf}^* = \underline{conf}\ o\ \underline{conf}$

où $\underline{conf}^*$ est la fermeture transitive de $\underline{conf}$ (informellement, $\underline{conf}\ o\ \underline{conf}\ o\ \underline{conf}\ o\ \ldots$).

(vi) $\underline{conf}^* = \underline{ext}\ o\ \underline{red}$

La proposition 5.4 (v) signifie que $\underline{conf}\ o\ \underline{conf}$ est la fermeture transitive de $\underline{conf}$, qui est la plus forte relation transitive plus faible que $\underline{conf}$.

Tous ces résultats sont résumés à la figure 5.1.

La figure 5.2 montre comment $\underline{cred}$, $\underline{cext}$ et $\underline{tc}$ sont liées à ces relations.

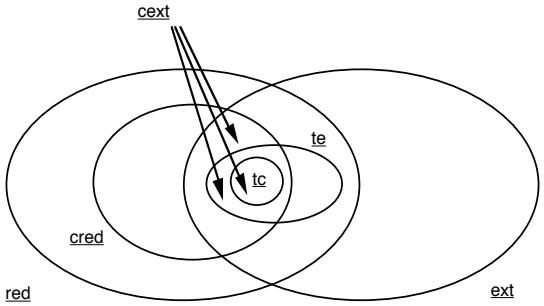

Figure 5.2 : Relations entre $\underline{tc}$, $\underline{te}$, $\underline{cred}$, $\underline{red}$, $\underline{cext}$, $\underline{ext}$

6. *Conf* comme relation d'implémentation de référence

Conf a été proposée pour formaliser la notion de conformité aux tests dans [Bri 88]. Puisque les tests de conformité ont avant tout comme objectif de tester des implémentations conformes, *conf* est un bon candidat comme relation d'implémentation de référence. De plus, *conf* est un exemple intéressant de relation non transitive.

Cette section étudie le processus d'implémentation lorsque la relation générique *imp* a été instanciée par *conf*. Les conséquences et les perspectives de ce choix sont également analysées.

Conf est réflexive et satisfait donc la seule condition préalable, mais *conf* n'est pas transitive [BrS 86].

Une première question intéressante en rapport avec *conf* est la nature du préordre *conf-restr* et de l'équivalence *conf-eq* associées à *conf*. Nous allons d'abord instancier quelques résultats des sections 2 et 3, et nous proposerons ensuite d'autres définitions de *conf-restr* et *conf-eq*.

Propositions 6.1 [Led 91]

(i) $\underline{conf\text{-}eq} \subset \underline{conf\text{-}restr} \subset \underline{conf}$

(ii) $\underline{conf\text{-}eq} \subset \underline{conf} \cap \underline{conf}^{-1}$

(iii) $\underline{conf\text{-}eq} \supset \underline{te}$
i.e., $\underline{conf\text{-}eq}$ est plus faible que l'équivalence de test

(iv) $\underline{conf\text{-}eq} \cap \underline{trace\text{-}eq} = \underline{conf} \cap \underline{conf}^{-1} \cap \underline{trace\text{-}eq} = \underline{te}$
ou de façon équivalente,
$\forall P, Q$, on a $P \ \underline{conf\text{-}eq} \ Q \wedge (Tr(P) = Tr(Q))$
$\Leftrightarrow \ P \ \underline{conf} \ Q \wedge Q \ \underline{conf} \ P \wedge (Tr(P) = Tr(Q))$
$\Leftrightarrow \ P \ \underline{te} \ Q$
Pour des processus ayant les mêmes traces, $\underline{conf\text{-}eq}$ et $\underline{te}$ sont identiques.

(v) $\underline{ext} \subset \underline{conf\text{-}restr}$

(vi) $\neg (\underline{red} \subseteq \underline{conf\text{-}restr})$

Tous ces résultats sont résumés à la figure 6.1 La partie hachurée est exactement l'équivalence de test. Dans [Led 90], des exemples ont été trouvés afin de prouver que les inclusions sont strictes, c'est-à-dire qu'aucune zone de la figure n'est vide.

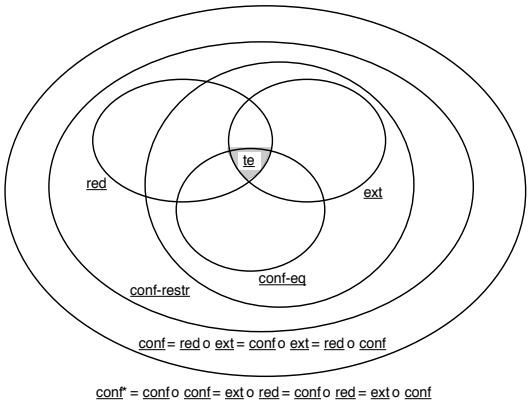

Figure 6.1 : Liens entre les relations principales

Nous donnons maintenant des définitions plus classiques de $\underline{conf\text{-}restr}$ et $\underline{conf\text{-}eq}$.

Propositions 6.2 [Led 90]

(i) $P \ \underline{conf\text{-}restr} \ Q$ *ssi*
(a) $P \ \underline{conf} \ Q$, *et*
(b) $\forall \sigma \in Tr(Q) - Tr(P)$, *on a* $L \in Ref(Q, \sigma)$

(ii) $P \ \underline{conf\text{-}eq} \ Q$ *ssi*
(a) $P \ \underline{conf} \ Q \wedge Q \ \underline{conf} \ P$, *et*
(i.e. $\forall \sigma \in Tr(P) \cap Tr(Q)$, *on a* $Ref(P, \sigma) = Ref(Q, \sigma)$)
(b) $\forall \sigma \in Tr(P) - Tr(Q)$, *on a* $L \in Ref(P, \sigma)$, *et*
(c) $\forall \sigma \in Tr(Q) - Tr(P)$, *on a* $L \in Ref(Q, \sigma)$

On voit mieux apparaître en quoi *conf-restr* est plus restrictive que *conf* : pour que P *conf-restr* Q, il faut que Q puisse accéder à un état terminal après toute trace n'appartenant pas à P.

Transformations autorisées par la relation d'implémentation conf

La proposition 6.1 (v) établit que *ext* ⊆ *conf-restr*, ce qui signifie que les *ext* - transformations sont autorisées par *conf*. Plus précisément, on peut étendre (au sens de *ext*) une spécification; après une telle transformation, les implémentations conformes sont toujours moins nombreuses.

Il en est tout autrement avec la réduction : la proposition 6.1 (vi) établit que *red* ⊆ *conf-restr* n'est pas vérifiée. En conséquence, les *red*-transformations ne sont pas autorisées par *conf* en général (malgré que *red* ⊆ *conf*), c'est-à-dire que la réduction (au sens de *red*) d'une spécification *S* peut en général conduire à une spécification *S'* qui accepte certaines implémentations conformes n'étant pas conformes à S. L'exemple simple suivant permet d'illustrer le problème (figure 6.2) :

Soient *S = i; a ; stop [] b; c; stop,*
S' = a; stop, *S' red S*
I = a; stop [] b; stop, *I conf S',* mais ¬ *(I conf S).*

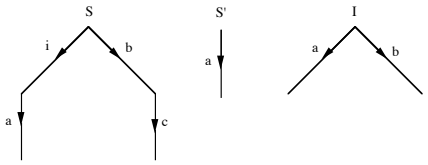

Figure 6.2 : ¬ (red ⊆ conf-restr)

On notera toutefois que *red* ∩ *ext* ⊂ *conf-restr*, ce qui signifie que toute réduction du non-déterminisme qui préserve les traces est une transformation autorisée par *conf*.

On notera enfin que, *cred* ⊆ *conf-restr* n'est pas vérifiée non plus.

En conclusion, si *conf* est choisie comme relation d'implémentation de référence, toute transformation doit respecter *conf-restr*, ce qui permet d'étendre la spécification (au sens de *ext*) ou de réduire le non-déterminisme (au sens de *red* ∩ *ext* , mais pas au sens de *red* seule).

Transformations locales

Dans la section 3, nous avons introduit la relation *cimp-restr* afin de caractériser les transformations locales autorisées. Nous allons maintenant étudier de plus près la relation *cconf-restr* : la plus faible précongruence plus forte que *conf-restr*.

Propositions 6.3 [Led 90]

(i) La plus faible précongruence[4] plus forte que *conf-restr* est *cext*,
i.e. *P cext Q ssi ∀ contexte C [.], on a C [P] conf-restr C [Q].*

(ii) La plus faible congruence[5] plus forte que *conf-eq* est *tc*,

---

[4] cf note 1

[5] cf note 1

i.e. *P* $\underline{tc}$ *Q ssi* $\forall$ *contexte C [.], on a C [P]* $\underline{conf\text{-}eq}$ *C [Q].*

La proposition 6.3 garantit que si une partie quelconque *Q* d'une spécification *C [Q]* est remplacée par un processus *P* tel que *P* $\underline{cext}$ *Q*, nous obtenons une spécification *C [P]* telle que *C [P]* $\underline{conf\text{-}restr}$ *C [Q]*; ce qui signifie que les implémentations conformes à *C [P]* sont aussi conformes à *C [Q]*.

7. *Red ou ext* comme relations d'implémentation de référence

Si une relation transitive comme $\underline{red}$ ou $\underline{ext}$ est choisie comme relation d'implémentation de référence, le cadre théorique que nous avons présenté se simplifie considérablement.

Considérons d'abord $\underline{red}$.

$\underline{Red}$ est réflexive et transitive; ce qui permet, par application des propositions 2.6, 2.7, 3.5 et 3.6 d'obtenir les résultats suivants.

Propositions 7.1

(i) $\underline{red\text{-}restr} = \underline{red}$

(ii) $\underline{red\text{-}eq} = \underline{red} \cap \underline{red}^{-1} = \underline{te}$

(iii) $\underline{Id} \subseteq \underline{R} \subseteq \underline{red} \Rightarrow \underline{red} \circ \underline{R} = \underline{red}$

(iv) une $\underline{R}$-transformation est autorisée par $\underline{red}$ ssi $\underline{R} \subseteq \underline{red}$

Proposition 7.2 [BrS 86]

La plus faible précongruence[6] plus forte que $\underline{red}$ est $\underline{cred}$,
i.e. *P* $\underline{cred}$ *Q* ssi $\forall$ *contexte C [.], on a C [P]* $\underline{red}$ *C [Q].*

La proposition 7.2 garantit que si une partie quelconque *Q* d'une spécification *C [Q]* est remplacée par un processus P tel que *P* $\underline{cred}$ *Q*, nous obtenons une spécification *C [P]* telle que *C [P]* $\underline{red}$ *C [Q]*, ce qui signifie que toute réduction de *C [P]* est aussi une réduction de *C [Q]*.

Le cas de la relation $\underline{ext}$ est en tout point similaire.

$\underline{Ext}$ est réflexive et transitive; ce qui permet, par application des propositions 2.6, 2.7, 3.5 et 3.6 d'obtenir les résultats suivants.

Propositions 7.3

(i) $\underline{ext\text{-}restr} = \underline{ext}$

(ii) $\underline{ext\text{-}eq} = \underline{ext} \cap \underline{ext}^{-1} = \underline{te}$

(iii) $\underline{Id} \subseteq \underline{R} \subseteq \underline{ext} \Rightarrow \underline{ext} \circ \underline{R} = \underline{ext}$

(iv) une $\underline{R}$-transformation est autorisée par $\underline{ext}$ ssi $\underline{R} \subseteq \underline{ext}$

Proposition 7.4 [BrS 86]

La plus faible précongruence[7] plus forte que $\underline{ext}$ est $\underline{cext}$,

---

[6] cf note 1

[7] cf note 1

i.e. $P \underline{cext} Q$ ssi $\forall$ *contexte C [.], on a C [P]* $\underline{ext}$ *C [Q]*.

La proposition 7.4 garantit que si une partie quelconque Q d'une spécification *C [Q]* est remplacée par un processus P tel que $P \underline{cext} Q$, nous obtenons une spécification *C [P]* telle que *C [P]* $\underline{ext}$ *C [Q]*, ce qui signifie que toute extension de *C [P]* est aussi une extension de *C [Q]*.

8. *Conf** comme relation d'implémentation de référence

Les sections 6 et 7 mettent bien en évidence que le choix d'une relation d'implémentation de référence à des conséquences importantes sur les transformations autorisées. Ainsi, l'extension ne peut être appliquée que si $\underline{imp} = \underline{conf}$ ou $\underline{imp} = \underline{ext}$. De même, la réduction ne peut être appliquée que si $\underline{imp} = \underline{red}$. La réduction du non-déterminisme sans réduction de traces peut être appliquée si $\underline{imp} = \underline{conf}$, $\underline{imp} = \underline{ext}$ ou $\underline{imp} = \underline{red}$.

Il peut être intéressant de se poser le problème dans le sens opposé, à savoir de fixer d'abord les transformations que l'on voudrait pouvoir effectuer, et ensuite de voir quelle relation d'implémentation de référence pourrait être choisie dans ce cas.

Ces considérations nous ont conduit à proposer $\underline{conf}$* comme relation d'implémentation. En effet, si, lors du processus d'implémentation, nous voulons pouvoir appliquer à plusieurs reprises des transformations de réduction et d'extension dans n'importe quel ordre, nous devons prévoir une relation d'implémentation de référence plus tolérante que toutes les précédentes. En fait, nous allons montrer que dans ce cas, il convient de choisir $\underline{imp} = \underline{conf}^*$.

La proposition suivante établit que $\underline{conf}^*$ est la plus forte relation transitive plus faible que $\underline{red}$ et $\underline{ext}$; il semble donc intéressant de la considérer, car c'est la relation d'implémentation la plus forte permettant d'effectuer des réductions ($\underline{red}$) et des extensions ($\underline{ext}$) de façon quelconque.

Proposition 8.1

$\underline{Conf}^*$ est la plus forte relation transitive plus faible que $\underline{red}$ et $\underline{ext}$

Preuve

Désignons par $\underline{T}$ une relation transitive quelconque plus faible que $\underline{red}$ et $\underline{ext}$.

Par la proposition 3.5 avec $\underline{imp} = \underline{T}$, nous obtenons $\underline{T} \circ \underline{red} = \underline{T} = \underline{T} \circ \underline{ext}$. (*)

Il suit que $\underline{T} \circ \underline{conf}^*$	$= \underline{T} \circ \underline{ext} \circ \underline{red}$	proposition 5.4 (vi)
	$= \underline{T} \circ \underline{red}$	car $\underline{T} \circ \underline{ext} = \underline{T}$, par (*)
	$= \underline{T}$	car $\underline{T} \circ \underline{red} = \underline{T}$, par (*) (**)
Et finalement $\underline{conf}^*$	$\subseteq \underline{T} \circ \underline{conf}^*$	$\underline{T}$ est réflexive car plus faible que $\underline{red}$ et $\underline{ext}$
	$= \underline{T}$	par (**).

❒

Des propositions 2.6, 2.7, 3.5 and 3.6, nous obtenons directement les résultats suivants.

Propositions 8.2

(i) $\underline{conf}^{*}\text{-}\underline{eq} = \underline{conf}^{*} \cap \underline{conf}^{*\,-1}$

(ii) $\underline{conf}^{*}\text{-}\underline{restr} = \underline{conf}^{*}$

(iii) $\underline{Id} \subseteq \underline{R} \subseteq \underline{conf}^{*} \Rightarrow \underline{conf}^{*}\ o\ \underline{R} = \underline{conf}^{*}$

(iv) une $\underline{R}$-transformation est autorisée par $\underline{conf}^{*}$ ssi $\underline{R} \subseteq \underline{conf}^{*}$

Avec $\underline{conf}^{*}$ comme relation d'implémentation de référence, toutes les relations que nous avons présentées jusqu'à présent sont des transformations autorisées.

Nous allons tenter de caractériser $\underline{conf}^{*}\text{-}\underline{eq}$ par rapport aux autres équivalences.

Proposition 8.3

$\underline{conf}^{*}\text{-}\underline{eq} \supset \underline{conf\text{-}eq}$

Preuve

$\underline{conf}^{*}\text{-}\underline{eq}$	$= \underline{conf}^{*} \cap \underline{conf}^{*\,-1}$	par définition
	$\supseteq \underline{conf} \cap \underline{conf}^{-1}$	car $\underline{conf} \subseteq \underline{conf}^{*}$
	$\supset \underline{conf\text{-}eq}$	par la proposition 6.1 (ii)

❒

Exemple

P = i; a; stop [] b; c; stop

Q = i; a; stop [] b; d; stop

$P\ \underline{conf}^{*}\text{-}\underline{eq}\ Q$: il suffit de prouver que $P\ \underline{conf}^{*}\ Q$ et $Q\ \underline{conf}^{*}\ P$, sachant que $\underline{conf}^{*} = \underline{ext}\ o\ \underline{red}$,

On voit facilement que $P\ \underline{ext}\ a; stop\ \underline{red}\ Q$ et $Q\ \underline{ext}\ a; stop\ \underline{red}\ P$

Mais $P\ \underline{conf\text{-}eq}\ Q$ n'est pas vérifiée, il suffit de considérer $I = P$, $I\ \underline{conf}\ P$, $\neg\ (I\ \underline{conf}\ Q)$

❒

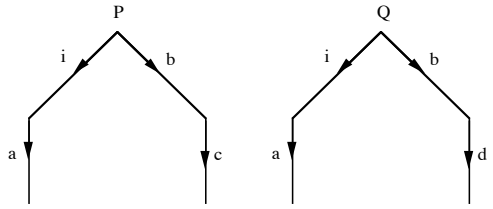

Figure 8.1 : $P\ \underline{conf}^{*}\text{-}\underline{eq}\ Q, \neg\ (P\ \underline{conf\text{-}eq}\ Q)$

Une définition de $\underline{conf}^{*}$ plus manipulable que la proposition 5.4 (vi) semble difficile à trouver. La proposition 8.5 est une tentative dans cette voie. Les propositions 8.4 établissent des conditions nécessaires pour que $P\ \underline{conf}^{*}\ Q$ soit vérifiée.

Propositions 8.4

(i) $P\ \underline{conf}^{*}\ Q \Rightarrow Ref\,(P, \varepsilon) \subseteq Ref\,(Q, \varepsilon)$

(ii) $P \underline{conf}^* Q \Rightarrow L - (out(P, \varepsilon) \cap out(Q, \varepsilon)) \in Ref(Q, \varepsilon)$

(iii) $P \underline{conf}^* Q \Rightarrow \{ a \mid \exists X \in Ref(Q, \varepsilon), tel\ que\ \{a\} \cup X \notin Ref(Q, \varepsilon)\} \subseteq out(P, \varepsilon)$

L'ensemble $\{ a \mid \exists X \in Ref(Q, \varepsilon), tel\ que\ \{a\} \cup X \notin Ref(Q, \varepsilon)\}$ est l'ensemble des actions que Q ne peut jamais refuser dans l'état initial.

Preuves

(i) $P \underline{conf}^* Q \Rightarrow \exists X, tel\ que\ P \underline{ext}\ X \underline{red}\ Q$, car $\underline{conf}^* = \underline{ext}\ o\ \underline{red}$

Il suit directement que $\forall \sigma \in Tr(X)$, on a

$Ref(P, \sigma) \subseteq Ref(X, \sigma) \subseteq Ref(Q, \sigma)$

et en particulier $Ref(P, \varepsilon) \subseteq Ref(Q, \varepsilon)$, car $\varepsilon \in Tr(X)$.

(ii) On sait que $\exists X, tel\ que\ P \underline{ext}\ X \underline{red}\ Q$,

ce qui implique $Tr(X) \subseteq Tr(P) \cap Tr(Q)$

ou encore $out(X, \varepsilon) \subseteq out(P, \varepsilon) \cap out(Q, \varepsilon)$

D'où $L - (out(P, \varepsilon) \cap out(Q, \varepsilon)) \subseteq L - out(X, \varepsilon)$. (*)

Or, on sait que $L - out(X, \varepsilon) \in Ref(X, \varepsilon)$,

ce qui implique $L - (out(P, \varepsilon) \cap out(Q, \varepsilon)) \in Ref(X, \varepsilon)$,

par (*) et la fermeture inclusive des ensembles de refus

Et enfin $L - (out(P, \varepsilon) \cap out(Q, \varepsilon)) \in Ref(Q, \varepsilon)$,

puisque $Ref(X, \varepsilon) \subseteq Ref(Q, \varepsilon)$.

(iii) Par contradiction supposons que $P \underline{conf}^* Q$, et l'existence d'une action a telle que

$a \notin out(P, \varepsilon)$, et (*)

$\exists X \in Ref(Q, \varepsilon), tel\ que\ \{a\} \cup X \notin Ref(Q, \varepsilon)$ (**)

Par (*), on obtient $\forall X' \in Ref(P, \varepsilon)$, on a $\{a\} \cup X' \in Ref(P, \varepsilon)$ (***)

Considérons donc un tel $X' \in Ref(P, \varepsilon)$, on en déduit que $X' \in Ref(Q, \varepsilon)$, par la proposition 8.4 (i)

Ce qui implique, par (**), que $\{a\} \cup X' \notin Ref(Q, \varepsilon)$

Or on sait, par (***), que $\{a\} \cup X' \in Ref(P, \varepsilon)$

Ces deux derniers résultats sont en contradiction par la proposition 8.4 (i).

❐

La proposition suivante est une condition nécessaire et suffisante pour que $P \underline{conf}^* Q$; elle est basée sur l'existence d'un X tel que $P \underline{ext}\ X \underline{red}\ Q$. Ce X est caractérisé modulo l'équivalence de test $\underline{te}$, en en donnant les traces et les refus après chaque trace.[8]

---

[8] Cette caractérisation d'un processus par ses traces et ses refus a d'abord été proposée en CSP par Hoare dans [Hoa 85] et en LOTOS par Brinksma dans [Bri 87] où ce modèle est appelé "Failure Tree". Un modèle semblable mais étendu est également développé dans [Led 90]. Il faut noter que les traces et les refus satisfont certaines propriétés générales présentées dans les travaux cités (e.g. tout préfixe d'une trace est une trace; si $X \in Ref(P, \sigma)$ et si $X' \subseteq X$, alors $X' \in Ref(P, \sigma)$). Ceci nous oblige, lorsque nous travaillons dans ce modèle, à vérifier que les ensembles de traces et de refus sont bien formés sous peine de traiter des processus irréels. Ces vérifications sont effectuées implicitement dans cet article.

Proposition 8.5

$P \underline{conf}^{*} Q$ ssi $\neg(lub\ (<X_i>)\ \underline{te}\ stop) \vee (stop\ \underline{red}\ Q)$

où $<X_i> = <X_0, X_1, \ldots>$ est la suite définie comme suit :

$Tr\ (X_0) = Tr\ (P) \cap Tr\ (Q)$

$Tr\ (X_{i+1}) = \{\varepsilon\} \cup (Tr\ (X_i) \cap \{\sigma \mid \forall \sigma' \leq \sigma, on\ a\ Ref\ (X_i, \sigma') \subseteq Ref\ (Q, \sigma')\})$

$\forall \sigma \in Tr\ (X_i), on\ a\ Ref\ (X_i, \sigma) =$

$\{A_1 \cup A_2 \mid A_1 \in Ref\ (P, \sigma) \wedge A_2 \subseteq out\ (P, \sigma) - out\ (X_i, \sigma)\}$

et où la "lub" (least upper bound) de $<X_i>$ est définie par rapport à l'ordre partiel $\underline{ext}$, i.e.

(a) $\forall i, X_i\ \underline{ext}\ lub\ (<X_i>)$

(b) $(\forall i, X_i\ \underline{ext}\ Q) \Rightarrow lub\ (<X_i>)\ \underline{ext}\ Q$

ou encore

(a) $Tr\ (lub\ (<X_i>)) = \bigcap_i Tr\ (X_i)$

(b) $\forall \sigma \in Tr\ (lub\ (<X_i>)), on\ a$

$Ref\ (lub\ (<X_i>), \sigma) = \bigcup_i Ref\ (X_i, \sigma)$

ou encore

(a) $Tr\ (lub\ (<X_i>)) = \bigcap_i Tr\ (X_i)$

(b) $\forall \sigma \in Tr\ (lub\ (<X_i>)), on\ a\ Ref\ (lub\ (<X_i>), \sigma) =$

$\{A_1 \cup A_2 \mid A_1 \in Ref\ (P, \sigma)$

$\wedge\ A_2 \subseteq out\ (P, \sigma) - out\ (lub\ (<X_i>), \sigma)\}$

Preuve

La méthode consiste à tester l'existence d'un processus X tel que $P\ \underline{ext}\ X\ \underline{red}\ Q$.

On voit directement que $P\ \underline{ext}\ X_0$, et que $\forall i, X_i\ \underline{ext}\ X_{i+1}$

Cette suite admet une lub (least upper bound), selon la relation d'ordre $\underline{ext}$.

De plus, on voit directement que $lub\ (<X_i>)$ a la caractéristique suivante :

$Tr\ (lub\ (<X_i>)) \neq \{\varepsilon\} \Rightarrow \forall \sigma \in Tr\ (lub\ (<X_i>)),$

$on\ a\ Ref\ (lub\ (<X_i>), \sigma) \subseteq Ref\ (Q, \sigma).$

$Tr\ (lub\ (<X_i>)) = \{\varepsilon\} \Rightarrow Ref\ (lub\ (<X_i>), \varepsilon) = \wp\ (L)$

Ce qui peut s'écrire $Tr\ (lub\ (<X_i>)) \neq \{\varepsilon\} \Rightarrow lub\ (<X_i>)\ \underline{conf}\ Q$

$Tr\ (lub\ (<X_i>)) = \{\varepsilon\} \Rightarrow lub\ (<X_i>)\ \underline{te}\ stop$

Comme, d'autre part, $Tr\ (lub\ (<X_i>)) \subseteq Tr\ (Q)$, puisque $Tr\ (X_0) \subseteq Tr\ (Q)$, on obtient

$lub\ (<X_i>)\ \underline{red}\ Q \quad \vee \quad lub\ (<X_i>)\ \underline{te}\ stop$ (*)

($\Leftarrow$)

(a) Si $\neg\ (lub\ (<X_i>)\ \underline{te}\ stop)$, nous pouvons réécrire (*) comme suit :

$lub\ (<X_i>)\ \underline{red}\ Q$.

Dès lors, il existe un X tel que $P\ \underline{ext}\ X\ \underline{red}\ Q$, en l'occurrence

$X = lub\ (<X_i>)$.

(b) Si $stop\ \underline{red}\ Q$, c'est encore plus simple puisqu'il suffit de prendre $X = stop$.

($\Rightarrow$)

Supposons l'existence d'un X tel que $P\ \underline{ext}\ X\ \underline{red}\ Q$,

et supposons que $(lub\ (<X_i>)\ \underline{te}\ stop) \wedge \neg\ (stop\ \underline{red}\ Q)$

Nous allons montrer que nous obtenons une contradiction.

$\{\varepsilon\} = Tr\ (lub\ (<X_i>)) \subseteq Tr\ (X)$, puisque X existe

$\subseteq Tr(P) \cap Tr(Q)$, puisque P $\underline{ext}$ X $\underline{red}$ Q
$= Tr(X_0)$, par définition de X_0

En conséquence, $\exists j$, *tel que* $Tr(X_{j+1}) \subseteq Tr(X) \subseteq Tr(X_j)$,
puisque les X_i forment une séquence de processus à ensemble de traces décroissants.
Nous savons que $\forall \sigma \in Tr(X)$, on a $Ref(X, \sigma) \subseteq Ref(Q, \sigma)$,
puisque X $\underline{red}$ Q.
Grâce au lemme 8.6 ci-après, nous obtenons
$\forall \sigma \in Tr(X)$, $\forall i \leq j$, *on a* $Ref(X_i, \sigma) \subseteq Ref(X, \sigma) \subseteq Ref(Q, \sigma)$.
En particulier, $\forall \sigma \in Tr(X)$, on a $Ref(X_j, \sigma) \subseteq Ref(Q, \sigma)$.
En conséquence, par définition de $Tr(X_{j+1})$, on obtient $Tr(X) \subseteq Tr(X_{j+1})$,
Ce qui implique $Tr(X) = Tr(X_{j+1})$,
puisque par hypothèse $Tr(X_{j+1}) \subseteq Tr(X)$.
Il est aisé de montrer que cela permet d'étendre le lemme 8.6 pour $i = j+1$.
Nous en déduisons alors que
$\forall \sigma \in Tr(X_{j+1})$, *on a* $Ref(X_{j+1}, \sigma) \subseteq Ref(Q, \sigma)$, (*)
avec $Tr(X_{j+1}) = Tr(X)$.
Il s'en suit que $Tr(lub(<X_i>)) = Tr(X_{j+1})$
par (*) et la définition de $Tr(X_k)$, $\forall k \geq j+1$
$= Tr(X)$.
Or $Tr(lub(<X_i>)) = \{\varepsilon\}$ par hypothèse puisque $lub(<X_i>) = stop$
Dès lors, $Tr(X) = \{\varepsilon\}$, et par suite X $\underline{te}$ $stop$.
Or par hypothèse: $\neg$ *(stop* $\underline{red}$ $Q)$, ce qui contredit le fait que X $\underline{red}$ Q.

❐

Lemme 8.6

Soit la suite $<X_i>$ définie dans la proposition 8.5 avec $lub(<X_i>)$ $\underline{te}$ $stop$,
et soient X tel que P $\underline{ext}$ X $\underline{red}$ Q, et j tel que $Tr(X_{j+1}) \subseteq Tr(X) \subseteq Tr(X_j)$, *on a*
$\forall i \leq j$, $\forall \sigma \in Tr(X)$, *on a* $Ref(X_i, \sigma) \subseteq Ref(X, \sigma)$

La preuve est présentée en annexe

Nous n'avons par contre aucun résultat précis sur la plus faible précongruence plus forte que $\underline{conf}^*$, nous pensons que c'est encore $\underline{cred}$[9]. Auquel cas, la plus faible congruence plus forte que $\underline{conf}^*$-$\underline{eq}$ serait encore $\underline{tc}$.

---

[9] cf note 1

9. Protocole et service

Dans cette section, nous abordons le problème du lien entre un service et un protocole censé fournir ce service.

Afin d'éviter autant que possible les ambiguïtés attachées aux termes "service" et "protocole", nous en donnerons les définitions suivantes.

Un *protocole (N)* est un processus distribué, composé d'une couche (N) - à son tour composée d'entités (N) - et d'un service (N-1), dont la description contient d'une part les interactions possibles sous forme de primitives (N) entre les entités (N) et un environnement (i.e. les utilisateurs du protocole), et d'autre part les interactions possibles sous forme de primitives (N-1) entre les entités (N) et le service (N-1), sans participation de l'environnement.

Cette définition diffère de la vision intuitive d'un protocole comme un ensemble d'entités s'échangeant des PDU, telle qu'elle avait été proposée dans le modèle de référence OSI [ISO 7498]. Notre définition fait apparaître explicitement le service (N-1) comme une composante du protocole (N); ce qui est d'ailleurs logique puisque les entités n'ont pas de possibilités d'interactions directes et qu'elles doivent connaître et utiliser le service (N-1). Notre définition est parfois appelée vision verticale du protocole (car on met en évidence les primitives et les SAP) par opposition à la vision horizontale (où l'on met en évidence les PDU).

Un *service (N)* est un processus distribué, dont on ne connaît pas la structure, et dont la description contient les interactions possibles sous forme de primitives (N) avec un environnement.

En LOTOS, une spécification de protocole aura donc la structure suivante (dans le cas de deux entités) :

```
Protocol_N [N_saps] := hide N-1_saps in
                                    layer_N [N_saps, N-1_saps]
                                    |[N-1_saps]|
                                    service_N-1 [N-1_saps]
    where layer_N [N_saps, N-1_saps] := entity_N [N_saps, N-1_saps]
                                        |||
                                        entity_N [N_saps, N-1_saps]
```

Remarquons la présence de l'opérateur *hide* qui est nécessaire étant donné que l'environnement (i.e. les utilisateurs du protocole) ne peut intervenir aux *N-1_saps*.

Il ressort de ces définitions que le service est une abstraction du protocole et que plusieurs protocoles distincts peuvent fournir des services identiques. Ceci nous conduit à considérer que le protocole puisse être vu comme une première étape de transformation du service en vue d'implémenter ce service. Le protocole pouvant lui-même être transformé successivement afin d'atteindre cet objectif. La figure 9.1 illustre ce processus d'implémentation généralisé dont le nouveau point de départ est le service.

Si nous considérons que le protocole est la première étape de transformation du service, cette transformation doit respecter les mêmes règles que toute autre transformation ultérieure du protocole. Ce qui nous conduit à considérer que la relation la plus faible qui doit lier le protocole au service soit la relation *imp-restr*, si *imp* est la relation d'implémentation de référence.

Nous obtenons donc le critère suivant : *protocole imp-restr service*

Ainsi, toute implémentation conforme au protocole sera conforme au service.

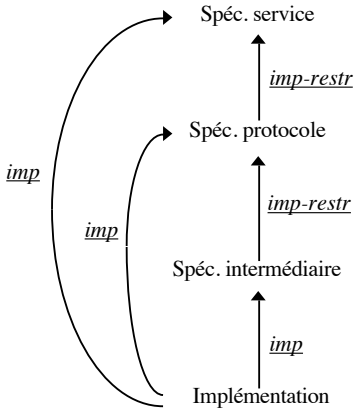

Figure 9.1 : Implémentation d'un service

Si nous tenons compte du fait que le service ainsi transformé est situé dans le contexte du protocole de la couche supérieure, il faut être plus prudent et tenir compte des règles applicables pour les transformations locales : nous exigerons donc *protocole* $\underline{cimp\text{-}restr}$ *service*.On peut bien entendu exiger un lien plus fort entre protocole et service; comme, par exemple, l'exigence supplémentaire que le protocole soit *complet* [BrS 86] vis-à-vis du service, i.e. *protocole* $\underline{imp\text{-}restr}^{-1}$ *service*. L'argument étant que si ce n'est pas le cas, cela complique inutilement le protocole de la couche supérieure qui a été conçu pour fonctionner avec le service et non le protocole. Si le protocole n'est pas complet vis-à-vis du service, on est sûr qu'aucune implémentation du protocole ne sera complète non plus. Si c'est jugé inacceptable, soit le protocole, soit le service doit être changé de façon à avoir *protocole* $\underline{imp\text{-}eq}$ *service* (cf. proposition 2.4 (iii)).

Illustrons cette problématique en prenant $\underline{imp}$ = $\underline{conf}$. Dans ce cas, $\underline{imp\text{-}restr}$ = $\underline{conf\text{-}restr}$ et $\underline{cimp\text{-}restr}$ = $\underline{cext}$[10]. On voit que dans ce cas, la règle à respecter est *protocole* $\underline{cext}$ *service* [11].

Cela correspond au fait que le protocole et le service doivent avoir les mêmes traces, mais que le protocole puisse être plus déterministe que le service.

10. Conclusion

Dans cet article, nous avons souligné le rôle d'une relation d'implémentation de référence $\underline{imp}$ lors du processus d'implémentation d'une spécification. En nous effor-

---

[10] cf note 1

[11] Pour autant que l'on vérifie en plus que le service ne donne pas lieu à des divergences dans le contexte du protocole de niveau supérieur.

çant de faire un minimum d'hypothèses sur la relation *imp*, nous avons développé un cadre général permettant de raisonner sur les transformations successives autorisées à chaque étape du processus d'implémentation. Les transformations sont caractérisées par une relation dérivée de *imp*, à savoir *imp-restr*. La relation d'implémentation *imp* induit également directement une équivalence *imp-eq*. Afin de permettre des transformations locales, nous avons introduit une dernière relation, *cimp-restr*, qui est la plus faible précongruence plus forte que *imp-restr*.

Ce cadre général a ensuite été particularisé en prenant comme relation *imp* diverses relations définies dans [BrS 86, BSS 87], à savoir *conf*, *red* et *ext*. La relation *conf* est particulièrement intéressante car elle est non transitive, ce qui permet d'illustrer diverses subtilités du cadre général. Il apparaît par exemple qu'une transformation de réduction de fonctionnalité (i.e. respectant la relation *red*) n'est pas autorisée en général même si $red \subseteq conf$. Par contre, une étape d'extension de la fonctionnalité est toujours possible.

Une nouvelle relation a également été introduite; il s'agit de la fermeture transitive de *conf*, notée *conf**. Le choix de cette relation comme relation d'implémentation de référence a l'avantage de permettre un plus grand nombre de transformations à chaque étape d'implémentation : on peut ainsi effectuer des extensions de la fonctionnalité mais aussi des réductions; et cela dans n'importe quel ordre. Nous avons démontré que *conf** est la relation d'implémentation la plus forte permettant ces deux types de transformations; ce qui est sans doute sa caractéristique principale. Quelques propriétés de *conf** ont aussi été établies, mais aucune justification "intuitive" de *conf** n'a été fournie. Nous avons donné des conditions nécessaires simples et une condition nécessaire et suffisante pour que $P\ conf^*\ S$ soit vérifié, ainsi que la propriété $conf^* = conf \circ conf = ext \circ red = ext \circ conf$.

Nous avons résumé les résultats obtenus dans le tableau suivant :

imp	*imp-restr*	*imp-eq*	*cimp-restr*
conf	*conf-restr*	*conf-eq*	*cext*
red	*red*	*te*	*cred*
ext	*ext*	*te*	*cext*
*conf**	*conf*-restr*	*conf*-eq*	*cred* (?)

Si nous considérons que *imp-restr* (ou *cimp-restr*) représente la liberté de l'implémenteur à chaque étape de transformation en fonction de la relation d'implémentation *imp*, nous constatons que cette liberté peut être plus grande alors que la relation d'implémentation est plus restrictive. En effet, il suffit de comparer les lignes du tableau relatives aux relations *conf* et *red* : alors que $red \subseteq conf$, on a $\neg\ (red \subseteq conf\text{-}restr)$. Ce paradoxe apparent a été étudié en détail dans [Led 91].

Bien que nous nous soyons limité à l'étude des relations *red*, *ext*, *conf* [BrS 86] et *conf**, cela ne restreint en rien le cadre théorique présenté qui pourrait également être

instancié par d'autres relations réflexives au fur et à mesure de leur apparition. On pourrait par exemple imaginer des relations telles que "diverge moins que", "est plus orienté ressources que", …

Il est possible que certaines de ces relations ne puissent pas être exprimées dans le modèle sémantique de LOTOS. D'autres modèles, tels que les systèmes de transitions modaux de [Lar 89], pourraient permettre la définition de relations plus riches.

Remerciements

Je remercie le Professeur Ed Brinksma pour ses remarques judicieuses formulées durant la rédaction de ma thèse d'agrégation de l'enseignement supérieur d'où le présent article est tiré.

Bibliographie

[AbL 88] M. Abadi, L. Lamport,
The Existence of Refinement Mappings
in: Third Annual Symposium on Logic in Computer Science, Edinburgh, Scotland, July 88, 165-175.

[BDH 88] D. Blyth, E. Dubuis, H. Hansson, G. Juanole, M. Kapus-Kolar, H. Kerner, G. Leduc, G. Le Moli, A. Lombardo, S. Marchena, W. Orth, J. Pavon, B. Pehrson, M. Tienari, F. Vogt,
Architectural and Behavioural Modelling in Computer Communication,
in: M. H. Barton, E. L. Dagless, G. L. Reijns, eds., Distributed Processing (North-Holland, Amsterdam, 1988, ISBN 0-444-70419-1), 53-70.

[BeK 85] J.A. Bergstra, J. W. Klop,
Algebra of Communicating Processes with Abstraction,
Theoretical Computer Science 37 (1985) 77-121 (North-Holland, Amsterdam).

[BHR 84] S.D. Brookes, C.A.R.Hoare, A.W.Roscoe,
A theory of Communicating Sequential Processes
Journ. ACM, Vol. 31, No. 3, July 1984, 560-599.

[BoB 87] T. Bolognesi, E. Brinksma,
Introduction to the ISO Specification Language LOTOS,
Computer Networks and ISDN Systems 14 (1) 25-59 (1987).
Also in : P.H.J. van Eijk, C.A. Vissers, M. Diaz, eds., The Formal Description Technique LOTOS, Results of the ESPRIT SEDOS Project, (North-Holland, Amsterdam, 1989, ISBN 0-444-87267-1) 23-73.

[Bog 89] K. Bogaards,
LOTOS supported system development,
in: K.J. Turner, ed., Formal Description Techniques (North-Holland, Amsterdam, 1989, ISBN: 0-444-87126-8) 279-294.

[Bri 87] E. Brinksma,
On the existence of canonical testers
Rept. No. INF-87-5, Twente University of Technology, Department of Informatics, Enschede, The Netherlands, January 1987.

[Bri 88] E. Brinksma,
A Theory for the Derivation of Tests,
in: S. Aggarwal, K. Sabnani, eds., Protocol Specification, Testing and Verification, VIII (North-Holland, Amsterdam, 1988, ISBN 0-444-70542-2) 63-74.

[BrS 86] E. Brinksma, G. Scollo,
Formal notions of implementation and conformance in LOTOS
Rept. No. INF-86-13, Twente University of Technology, Department of Informatics, Enschede, The Netherlands, Dec. 1986.

[BSS 87] E. Brinksma, G. Scollo, C. Steenbergen,
Process specification, their implementations and their tests,
in: G.v. Bochmann, B. Sarikaya, eds., Protocol Specification, Testing and Verification, VI (North-Holland, Amsterdam, 1987, ISBN 0-444-70126-5), 349-360.

[ChM 89] K. M. Chandy, J. Misra,
Parallel Program Design - A Foundation,
(Addison-Wesley, 1989, ISBN 0-201-05866-9).

[DGH 88] E. Dubuis, R. Gotzhein, H. Hansson, G. Juanole, H. Kerner, P. Lahtinen, G. Leduc, A. Lombardo, S. Marchena, W. Orth, S. Palazzo, J. Pavon, U. Thalmann, M. Tienari, I. Tvrdy,
A Framework for the Taxonomy of Synthesis and Analysis Activities in Distributed System Design
in: R. Speth, ed., Research into Networks and Distributed Applications (North-Holland, Amsterdam, 1988, ISBN 0-444-70428-0), 859-871.

[dNH 84] R. De Nicola, M.C.B. Hennessy,
Testing equivalences for processes,
Theoretical Computer Science 34 (1984) 83-133 (North-Holland, Amsterdam).

[dNi 87] R. De Nicola,
Extensional Equivalences for Transition Systems,
Acta Informatica 24 (1987) 211-237 (Springer - Verlag, Berlin Heidelberg).

[Hoa 85] C.A.R. Hoare,
Communicating Sequential Processes,
(Prentice-Hall International, London, 1985, ISBN 0-13-153271-5).

[ISO 7498] ISO-TC97/SC16/WG1,
Information Processing Systems, Open Systems Interconnection, Basic Reference Model,
ISO 7498, 1984

[ISO 8807] ISO/IEC-JTC1/SC21/WG1/FDT/C,
Information Processing Systems - Open Systems Interconnection - LOTOS, a Formal Description Technique Based on the Temporal Ordering of Observational Behaviour,
IS 8807, February 1989.

[Lam 83] L. Lamport,
Specifying concurrent program modules,
ACM Transactions on Programming Languages and Systems 5 (2) 190-222 (1983).

[Lar 89] K. G. Larsen,
Modal Specifications,
in: J. Sifakis, ed., Automatic Verification Methods for Finite State Systems (LNCS 407, Springer - Verlag, Berlin Heidelberg New York, 1990, ISBN 3-540-52148-8), 232-246.

[LaS 84] S.S. Lam, A. U. Shankar,
Protocol verification via projections,
IEEE Transactions on Software Engineering, Vol. 10, No. 4, April 1984, 325-342.

[Led 90] G. Leduc,
On the role of implementation relations in the design of distributed systems,
Agrégation dissertation, University of Liège, Dept. Systèmes et Automatique, B28, Liège, Belgium, 1990.

[Led 91] G. Leduc,
A Framework based on implementation relations for implementing LOTOS specifications,
Paper accepted in: Computer Networks & ISDN Systems, 1991.

[LyF 81] N. Lynch, M. Fisher,
On describing the behavior and implementation of distributed systems,
Theoretical Computer Science 13 (1981) 17-43 (North-Holland, Amsterdam).

[LyT 87] N. Lynch, M. Tuttle,
Hierarchical correctness proofs for distributed algorithms,
in: 6th ACM Symposium on Principles of Distributed Computing, Vancouver, British Columbia, Canada , Aug. 87, 137 - 151.

[Mer 89] M. Merritt,
Completeness Theorems for Automata,
Rept. AT&T Bell Laboratories, Murray Hill, NJ, May 1989.

[Mil 89] R. Milner,
Communication and Concurrency,
(Prentice-Hall International, London, 1989, ISBN 0-13-114984-9).

[vEi 89] P. van Eijk,
Tools for LOTOS Specification Style Transformations,
in: S. T. Vuong, ed., FORTE '89, Vancouver, Canada, Dec. 89 (North-Holland, Amsterdam, 1990).

[vGl 90] R.J. van Glabeek,
The Linear Time - Branching Time Spectrum,
in: J.C.M. Baeten, J.W. Klop, eds., CONCUR '90, Theories of Concurrency: Unification and Extension, (LNCS 458, Springer - Verlag, Berlin Heidelberg New York, 1990, ISBN 3-540-53048-7), 278-297.

[VSv 88] C.A. Vissers, G. Scollo, M. van Sinderen,
Architecture and Specification Style in Formal Descriptions of Distributed Systems,
in: S. Aggarwal, K. Sabnani, eds., Protocol Specification, Testing and Verification, VIII (North-Holland, Amsterdam, 1988, ISBN 0-444-70542-2) 189-204.

Annexe : lemme 8.6

Soit la suite $<X_i>$ définie dans la proposition 8.5 avec *lub* $(<X_i>)$ *te stop*,

et soient X tel que P $\underline{ext}$ X $\underline{red}$ Q, et j tel que $Tr(X_{j+1}) \subseteq Tr(X) \subseteq Tr(X_j)$, on a $\forall i \leq j$, $\forall \sigma \in Tr(X)$, on a $Ref(X_i, \sigma) \subseteq Ref(X, \sigma)$

Preuve

Le cas de base (i) et le cas inductif (ii).

(i) Thèse : $Ref(X_0, \sigma) \subseteq Ref(X, \sigma)$

De $Tr(X) \subseteq Tr(P) \cap Tr(Q) = Tr(X_0)$, on obtient par définition de X_0

$\forall \sigma \in Tr(X)$, on a $Ref(X_0, \sigma) =$

$$\{A \cup B \mid A \in Ref(P, \sigma) \wedge B \subseteq Out(P, \sigma) - Out(X_0, \sigma)\}$$

Soient $\sigma \in Tr(X)$ et $A \in Ref(X_0, \sigma)$,

$A = A_1 \cup A_2$ où $A_1 \in Ref(P, \sigma)$ et $A_2 \subseteq Out(P, \sigma) - Out(X_0, \sigma)$.

Puisque X existe, nous obtenons :

$\forall \sigma \in Tr(X)$, on a $Ref(P, \sigma) \subseteq Ref(X, \sigma) \subseteq Ref(Q, \sigma)$.

En conséquence, $A_1 \in Ref(X)$. (*)

De plus, $A_2 \subseteq Out(P, \sigma) - Out(X_0, \sigma)$

$\subseteq Out(P, \sigma) - Out(X, \sigma)$,

car $Tr(X) \subseteq Tr(X_0)$, d'où $Out(X, \sigma) \subseteq Out(X_0)$

$\subseteq L - Out(X, \sigma)$ (**)

De (*) et (**), nous obtenons $A = A_1 \cup A_2 \in Ref(X, \sigma)$.

(ii) Thèse : $i < j \Rightarrow \forall \sigma \in Tr(X)$,

on a $Ref(X_i, \sigma) \subseteq Ref(X, \sigma) \Rightarrow Ref(X_{i+1}, \sigma) \subseteq Ref(X, \sigma)$

On sait par hypothèse que $Tr(X_{j+1}) \subseteq Tr(X) \subseteq Tr(X_j)$

Dès lors, puisque i < j, on a $Tr(X) \subseteq Tr(X_j) \subseteq Tr(X_{i+1})$

On obtient alors par définition de la suite $<X_i>$ et puisque $Tr(X) \subseteq Tr(X_{i+1})$:

$\forall \sigma \in Tr(X)$, $Ref(X_{i+1}, \sigma) =$

$$\{A_1 \cup A_2 \mid A_1 \in Ref(P, \sigma) \wedge A_2 \subseteq out(P, \sigma) - out(X_{i+1}, \sigma)\}.$$

Soient $\sigma \in Tr(X)$ et $A \in Ref(X_{i+1}, \sigma)$,

on a $A = A_1 \cup A_2$

où $A_1 \in Ref(P, \sigma)$ et $A_2 \subseteq out(P, \sigma) - out(X_{i+1}, \sigma)$.

De P $\underline{ext}$ X, on a $A_1 \in Ref(X, \sigma)$ (*)

De plus, on peut prouver que $A_2 \subseteq L - Out(X, \sigma)$ (**)

Puisque $A_2 \subseteq out(P, \sigma) - out(X_{i+1}, \sigma)$ par définition

$\subseteq out(P, \sigma) - out(X, \sigma)$ car $Tr(X) \subseteq Tr(X_{i+1})$

$\subseteq L - out(X, \sigma)$

De (*) et (**), on obtient $A = A_1 \cup A_2 \in Ref(X, \sigma)$.

❐