Laboratoire PPS :
Preuves, Programmes, Systèmes

Thèse de doctorat
de l'Université Paris VII – Denis Diderot
Spécialité : Informatique Fondamentale

Programmer, Calculer et Raisonner avec les Réseaux de la Logique Linéaire

présentée et soutenue publiquement le 16 décembre 2009 par

Stéphane Gimenez

devant le jury composé de

Thomas Ehrhard (Directeur de thèse)
Marcelo Fiore (Rapporteur)
Stefano Guerrini
Daniel Hirschkoff
Paul-André Melliès
Laurent Regnier (Rapporteur)

Révision : 2011-03-02

Les mises à jour de ce document, ainsi que d'autres formats,
sont disponibles à l'adresse :

http://www.pps.jussieu.fr/~gimenez/these

Merci

à plein de gens…
aux acteurs qui ont permis l'existence de ce document,
ainsi qu'aux lecteurs qui lui en donneront peut-être de nouvelles

Je remercie plus particulièrement Thomas, qui m'a présenté les divers aspects d'un monde à l'intersection des mathématiques et de l'informatique que je connaissais bien peu, et qu'il m'a plu de découvrir. Un monde qui révèle d'importants enjeux et dont l'intérêt apparaîtra plus clairement bientôt. En tout cas c'est ce que je pense, et j'aimerais bien avoir la chance de pouvoir explorer ce monde un peu plus. Un profond merci pour toute l'aide et pour la grande liberté que tu m'as laissée.

Je tiens aussi à remercier les membres de mon jury. Leur présence me fait honneur. Un merci spécial à *monsieur* Daniel, qui a si bien animé le MIM. Et un merci de plus à messieurs Laurent et Marcelo, car ils ont gentiment accepté de lire ces quelques pages et d'en faire un rapport.

L'entreprise à commencé alors que je jouais avec des petits graphes sur mon vieil ordinateur portable (remercié lui aussi). Jade m'a dit que je ferais bien d'en parler avec Manu, qui m'a dit d'aller voir monsieur Olivier. Grâce à eux (merci), j'avais appris que ces petits graphes c'était des réseaux d'interaction, en pas tout à fait aussi bien. Les réseaux d'interaction (merci monsieur l'inventeur), ce sont ces drôles d'objets qui sont dessinés un peu partout dans ce document. Peu après, j'ai rencontré Thomas, qui s'intéressait lui aussi à ces réseaux. Ayant déjà assisté à l'un de ses exposés, je savais qu'il s'amusait avec quelques acolytes (coucou à Lionel) à « dériver des programmes », ce qui était tout de même un peu fou. Tout ça avait donné naissance aux réseaux d'interaction différentiels, une espèce un peu particulière et très intrigante qui vit dans un espace un peu plus grand (mais on va garder les détails pour plus tard…). On pouvait espérer faire plein de choses avec, et ça me tentait bien d'essayer. En tout cas cela formait un passionnant sujet pour commencer une thèse.

Je me retrouvais donc dans ce bâtiment jaune, qui héberge le laboratoire PPS. Après seulement quelques jours on s'aperçoit qu'il y a beaucoup de langues officielles à PPS. Outre le franglais et l'agitage de mains (clin d'œil aux très conviviales troupes italiennes), il y a des gens qui parlent le lambda-calcul, d'autres qui parlent les catégories, certains les jeux, quelques uns le pi-calcul, beaucoup discutent dans des dialectes plutôt obscurs avec leurs ordinateurs (il ne faut pas oublier qu'on fait de l'informatique), et pour ma part, je me suis mis à dialoguer (un peu trop souvent) avec des réseaux. Au final, tout cela contribue à un excellent environnement pour faire de la science, car fort heureusement on trouve des interprètes pour toutes ces langues (merci aux encyclopédies vivantes qui traînent à PPS).

Je tiens à remercier les habitants des lieux, habituels ou occasionnels, bien sûr pour les discussions scientifiques, mais aussi pour la joyeuse effervescence autour de la table à café, ainsi que pour les bons moments passés ici ou ailleurs. Je remercierai plus particulièrement les personnes avec qui j'ai eu la chance de partager un bureau, Christine, Pierre, Paolo (grazie mille pour l'accueil à Rome), Séverine, Grégoire, Mehdi (le dealer de fourmis) et Stéphane. Plus tôt, il y a aussi eu Sam, Sylvain et Fabien. Tous ont su pimenter l'atmosphère d'une façon très agréable. L'ambiance n'aurait pas non plus été la même sans les sympathiques thésards d'à côté. Merci à Smimou, Nicolas, Fabien, re-Stéphane, Barbara, Thibault, Boris, Jonas, Gabriel, Beniamino… Merci aussi aux plus grands, Pierre, Vincent, Alexandre et les autres, ainsi qu'à la petite troupe du LIAFA : Claire, Julien, Florian, Mathilde, Marie et Eudes. Merci à David et Nicolas qui sont quelques fois venus du LIX pour rigoler avec nous, et aussi pour gribouiller un peu au tableau. Merci aux gens du chocolat.

Un merci++ pour mes compagnons de rédaction (Pierre, Christine et Grégoire) qui furent une source de soutient et de motivation fort appréciée pendant cette période. Merci à Odile, Audrey et Michèle qui rendent tout ça possible dans la bonne humeur.

Je pense également à d'autres amis, aux colocs en particulier, qui ont réussi à me supporter pendant ces quelques années. Si elles ont été formidables c'est aussi (et surtout) grâce à eux. Merci aux gens avec qui on a fait des truc cools, comme du roller, du go, de l'escalade, des gâteaux, du coiffage, du logiciel libre, du thé, de la fête de la science, du regardage de bon films, du regardage de mauvais films, de la coinche, du ski, des balades et bien d'autres choses. Merci Benoit pour les discussions diverses sur les choses et l'univers, à côté de ça on se dit qu'une thèse c'est quand même bien concret.

Enfin, je tiens à exprimer une immense reconnaissance à ma famille, à mon père Serge, à ma mère Brigitte, à mon frère Yannick (la petite créature est de lui) et aux autres, merci à eux pour tout ce qu'ils m'apportent et pour avoir toujours été là.

Table des matières

Introduction

Cette thèse gravite principalement autour de deux outils, la *logique linéaire* et les *réseaux d'interaction*.

De façon un peu surprenante, la *logique linéaire* n'est pas seulement un système dans lequel on peut formuler les *preuves* de certains théorèmes, c'est aussi un langage mathématique qui permet d'exprimer des *programmes*. En tout cas, c'est ce dernier aspect qui va susciter notre intérêt. Les programmes dont il est question sont tout à fait similaires à ceux qui s'exécutent sur un ordinateur, et nous parlons donc d'un véritable langage de programmation. Il est utilisé lorsqu'on souhaite exprimer et étudier des programmes avec une certaine rigueur mathématique : il donne une spécification à la fois concise, précise et assez canonique de leurs comportements. On devine donc un langage fondé sur un petit nombre de *constructions* qu'il faut utiliser en assez grand nombre pour espérer faire des choses intéressantes. Considérées séparément toutes ses constructions sont étonnement simples, on pourrait les comparer aux briques élémentaires utilisées pour bâtir un édifice architectural. Lorsqu'on commence à vouloir les faire fonctionner ensemble, on s'aperçoit que ces petites constructions sont dotées d'une expressivité suffisante pour décrire des relations compliquées entre un jeu de *données* et un *résultat*. C'est en cela qu'elles permettent de former des programmes, même si pour l'instant rien ne dit qu'il est possible de les exécuter facilement. Contrairement à beaucoup d'autres langages de programmation, il n'induit pas le programmeur à raisonner en termes d'*états* et d'*instructions* qui modifient ces états. C'est un langage qui vit indépendamment de toute stratégie de calcul et de tout support physique qui permettrait son exécution.

À l'opposé, les *réseaux d'interaction* forment ce qu'on appelle un modèle de calcul. Ce sont des objets qui ont été créés dans le but précis d'être exécutés. Cette fois la problématique est inversée, on sait que ces réseaux calculent mais on aimerait bien savoir ce qu'ils calculent, ou plutôt comment les utiliser pour calculer quelque chose d'utile. Nous y reviendrons. La spécificité de ce modèle de calcul est qu'il possède une notion intrinsèque de *parallélisme*, et c'est pour cela qu'on s'y intéresse. Le parallélisme s'oppose à la notion de *séquentialité*. Les différents pas d'un calcul séquentiel s'enchaînent, et on ne peut commencer le suivant qu'après avoir achevé le précédent. En particulier, lorsqu'un calcul est décrit de cette manière, rien ne permet à celui qui l'exécute de le partager en isolant certaines tâches indépendantes, même de façon temporaire. Il est donc inutile d'être deux pour effectuer ce calcul, on ne saurait comment s'y prendre pour l'effectuer plus vite. Le modèle des réseaux d'interaction est lui fondé sur une *réécriture locale* de graphes et son exécution se découpe naturellement en une multitude de petites tâches distinctes. On peut facilement le déployer sur un ordinateur, ou sur tout autre appareil capable d'effectuer un calcul, en tirant parti de leur propension éventuelle au calcul parallèle. Aussi, contrairement à d'autres modèles parallèles, il ne contourne pas le problème en décrivant plusieurs *fils d'exécution* qui restent séquentiels, sa qualité est plutôt due à une absence générale de séquentialité.

Réseaux et localisation

La première partie de cette thèse propose des initiatives nouvelles pour établir un pont entre ces deux outils. Elle présente un processus de *compilation* qui permet de traduire la logique linéaire, dans laquelle il est pratique de programmer, en un système de réseaux d'interaction avec lequel il naturel de calculer (surtout pour une machine).

Les formules de la logique linéaire s'écrivent dans la syntaxe suivante :

$$A \quad ::= \quad \begin{array}{c} \underbrace{A \otimes A \mid A \parr A \mid 1 \mid \bot}_{\text{multiplicatifs}} \mid \\ \underbrace{!A \mid ?A}_{\text{exponentiels}} \mid \\ \underbrace{A \oplus A \mid A \,\&\, A \mid 0 \mid \top}_{\text{additifs}} \end{array}$$

Le fragment *multiplicatif* de cette logique correspond dans une certaine mesure à ce que le programmeur connaît sur la formation de couples ou d'enregistrements, et également sur les constructions fonctionnelles. Sa traduction ne pose pas de problèmes spécifiques. Nous l'évoquerons peu après avoir défini le cadre dans lequel nous travaillons. Les difficultés n'apparaissent que lorsqu'on s'intéresse au fragment *exponentiel*. C'est celui qui introduit une capacité de *duplication* ou d'*effacement* des données. Trois formalismes qui présentent des avantages distincts seront évoqués pour traiter ce fragment problématique.

Si on oublie la logique qui les a fait naître, ces formalismes embrassent déjà toute l'expressivité computationnelle que l'on peut souhaiter. Cependant, pour conserver le confort de programmation que nous offre le typage, on préférera rester à l'intérieur de cette logique, quitte à la compléter. Cela nous amènera à considérer le fragment *additif*, qui correspond aux constructions conditionnelles des langages de programmation, puis à introduire une construction spécifique qui permet la *récursion*. La méthodologie utilisée précédemment pour le fragment exponentiel s'adapte relativement bien à ces nouvelles constructions.

On finira par la présentation d'une machine d'exécution parallèle qui à été réalisée grâce aux concepts développés dans cette première partie. Ce sera l'occasion de parler informellement de sujets annexes qui sont apparus principalement lors de l'élaboration de ce petit *logi*-ciel.

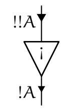
Approche différentielle

La deuxième partie est une contribution au monde — encore trop petit — de la logique linéaire différentielle. Ce sera une fois de plus les réseaux qui sont à l'origine de la majorité des idées développées, mais on s'éloigne de l'aspect computationnel pour s'orienter plutôt vers l'analyse de programmes.

C'est une modification de la couche exponentielle de la logique qui nous permet de passer dans le monde différentiel. On remplace la construction usuelle de *promotion*, qui autorise la réplication de la preuve qu'elle contient, par certaines constructions particulières qui regroupent plusieurs instances de cette preuve. La différence est subtile, mais la dynamique de cette logique est simplifiée : elle ne fait plus intervenir ces fameuses opérations de *duplication* ou d'*effacement*. Chaque hypothèse doit être fournie en quantité suffisante et sera consommée exactement une fois. La logique linéaire différentielle parle donc de ressources à usage unique.

Ce système est intéressant car on est toujours capable d'interpréter les preuves de logiques usuelles, il suffira de considérer un ensemble potentiellement infini de preuves écrites dans cette logique. Lorsqu'on considère d'autres langages (et on commencera d'ailleurs le plus souvent par évoquer le λ-*calcul* ou le π-*calcul* avant d'évoquer la logique), on s'aperçoit qu'on arrive par un procédé similaire à décomposer leurs termes en composants linéaires. Cela permet généralement de mieux comprendre ces langages, voire d'établir des liens entre eux.

Nous proposerons une logique différentielle complétée, combinant l'introduction de ressources à usage unique et la possibilité de réplication au sein d'un même calcul. Dans un premier temps, on se contentera d'indiquer comment réintroduire la *promotion* dans la logique linéaire différentielle. Par la suite, une construction de *super-promotion* sera présentée, elle est mieux adaptée au cadre différentiel que la *promotion* habituelle. Elle dévoile l'existence d'un acteur, plus ou moins sympathique mais néanmoins oublié jusqu'alors, le *co-enfouissement* :



Elle permet également au calcul de retrouver sa symétrie : les offres de ressources « !A » et les demandes de ressources « ?A » sont décrites de façon similaire.

Compléments

La troisième partie regroupe plusieurs considérations qui fourniront quelques éclairages supplémentaires sur les divers systèmes considérés. On rappellera la preuve de normalisation faible (obtenue par réalisabilité) des réseaux de preuve de la logique linéaire, et on proposera une extension de celle-ci lorsqu'on y ajoute les opérateurs différentiels. Nous parlerons aussi de sémantique relationnelle. C'est une matière que nous présenterons un peu tardivement faute de pouvoir parler de tout en même temps, mais qui a souvent été motrice lors de l'élaboration de nos divers systèmes.

Première partie

Réseaux et localisation

Chapitre 1

Réseaux

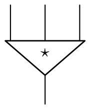
On se propose de coder les preuves issues de la logique linéaire de *Jean-Yves Girard* [Gir87 ; GLT89] dans la syntaxe des réseaux d'interaction introduits par *Yves Lafont* [Laf90 ; Laf95]. Ce premier chapitre sert d'introduction aux divers objets et systèmes que nous serons amenés à utiliser.

1.1 Réseaux d'interaction

Les réseaux d'interaction constituent un modèle de calcul simple dans lequel les opérations nécessaires au calcul sont atomiques. Fondés sur une réécriture de graphes, ils permettent d'exprimer d'autres modèles dont les réductions sont locales et peuvent éventuellement être effectuées de manière parallèle. Ils permettent en particulier d'exprimer la dynamique des machines de *Turing* ou des automates cellulaires.

1.1.1 Syntaxe

On se donne un ensemble $\mathfrak{O}$ d'*opérateurs*. On associe à chaque opérateur un entier naturel nommé *arité*. Chaque opérateur $\star \in \mathfrak{O}$ peut donner naissance à plusieurs *nœuds* que nous représenterons graphiquement comme ceci :



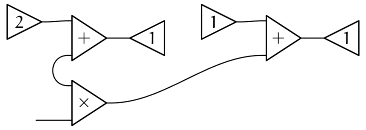
De chaque nœud part un fil qui est attaché à son *port principal* (ici en direction du bas), et d'autres fils attachés à ses *ports auxiliaires*. Le nombre de ports auxiliaires d'un nœud est déterminé par l'arité de l'opérateur qui étiquette ce nœud. Ici, on a donc supposé que l'opérateur $\star$ est d'arité 3.

Définition 1-1. *On nommera* réseau *(d'interaction) un ensemble de* nœuds *et de* ports libres *reliés par des* fils*. Usuellement un fil relie exactement deux ports distincts, sauf si celui-ci est une* boucle *auquel cas il est isolé et n'est lié à aucun port.*

Cette définition reste volontairement informelle, des présentations plus rigoureuses existent, on se référera par exemple à la thèse de *Lionel Vaux* [Vau07b] ou aux définitions à base de permutations de *Marc de Falco* [Fal09a ; Fal09b]. L'existence des boucles sera justifiée plus loin, en 1.1.3.

Exemple

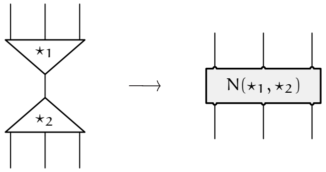
Avec deux opérateurs binaires (c'est-à-dire d'arité 2) nommés ‹ + › et ‹ × › et des opérateurs nullaires (d'arité 0) nommés ‹ 0 ›, ‹ 1 ›, ‹ 2 ›... on peut construire le réseau suivant :



il possède 7 nœuds, 7 fils et 1 port libre.

1.1.2 Sémantique opérationnelle

On introduit une notion de dynamique sur les réseaux, grâce à des règles de réduction. La forme générale d'une règle de réduction est donnée ci-dessous. $\mathsf{N}(\star_1, \star_2)$ représente un morceau de réseau particulier associé à une paire $(\star_1, \star_2)$ d'opérateurs donnée :



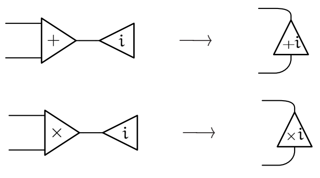
Une telle règle doit exister pour chaque couple $(\star_1, \star_2)$ d'opérateurs associés à des nœuds qui peuvent se rencontrer liés par leur ports principaux. On parle d'interaction entre les deux nœuds. Aussi, afin d'assurer le déterminisme de cette réduction, $\mathsf{N}(\star_1, \star_2)$ doit être symétrique lorsque $\star_1 = \star_2$.

Définition 1-2. *Étant donné un ensemble adéquat de règles de réduction, un réseau N se réduit en un réseau N' (on notera $N \longrightarrow N'$) lorsque N' est le réseau obtenu à partir de N en remplaçant un ensemble constitué de deux nœuds liés par leurs ports principaux (appelé* rédex*) selon la règle de réduction correspondante.*

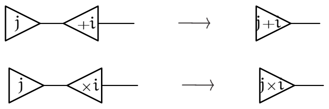
La clôture réflexive et transitive d'une relation de réduction $\longrightarrow$ sera notée $\twoheadrightarrow$.

Exemple

Pour les opérateurs de notre exemple précédent on peut penser aux réductions suivantes :

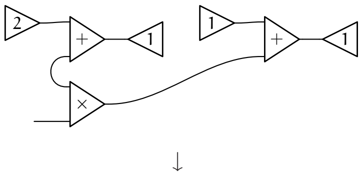


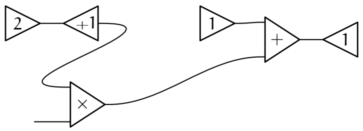
qui introduisent de nouveaux opérateurs $\langle +i \rangle$ et $\langle \times i \rangle$ unaires (d'arité 1) pour lequel il faut également définir une réduction :



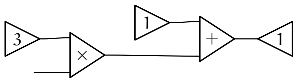
Ces règles couvrent toutes les rencontres que nous seront amenés à rencontrer. Plus tard, des systèmes de types nous permettront savoir quelles sont les interactions qu'il est nécessaire de définir.

Avec ces règles, examinons les réductions que l'on peut faire sur notre réseau de départ. Il possède initialement deux rédex, et, arbitrairement, nous allons choisir de réduire celui de gauche en premier :

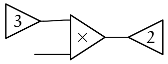




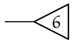
Si nous poursuivons la réduction, et que nous réduisons le nouveau rédex qui est apparu, nous obtenons :



On continue la réduction en effectuant un traitement similaire pour l'addition restante (en deux étapes), on obtient :



Enfin, on traite le cas de la multiplication, et on obtient un résultat qu'il n'est plus possible de réduire :



Mais que se serait-il passé si nous avions effectué les réductions dans un ordre différent ? Nous verrons que la structure même des réseaux d'interaction garantit que l'on obtient tout le temps le même résultat.

1.1.3 Forme normale et interblocages

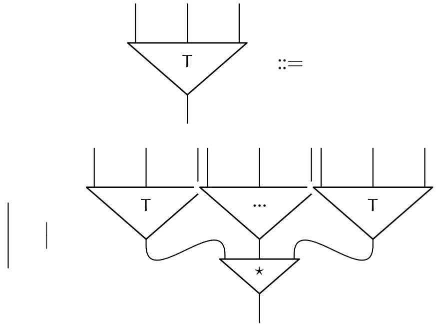
Après réductions successives d'un réseau de départ, on espère le plus souvent obtenir un résultat sous la forme d'un réseau qui ne possède plus aucun rédex.

Définition 1-3. *Un réseau est dit en* forme normale *lorsqu'il ne possède aucun rédex.*

Dans certains systèmes, il est possible que la réduction ne termine pas et qu'on n'atteigne jamais une forme normale pour certains réseaux de départ. On préférera bien sûr travailler dans des systèmes qui terminent. Contrairement à la confluence qui est automatiquement assurée, la terminaison devra être explicitement vérifiée.

Deux cas triviaux de réseaux en forme normale sont les arbres de nœuds et les câblages.

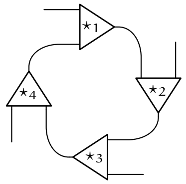
Les *arbres* sont définis inductivement : un fil est un arbre, le réseau formé par un nœud dont les ports auxiliaires ont été connectés aux liens principaux d'autres arbres est un arbre. Plus précisément la syntaxe graphique qui définit les arbres est la suivante, leurs liens principaux sont représentés en direction du bas :



Un *câblage* est un réseau constitué uniquement de fils, pas de nœuds, pas de boucles.

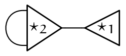
Interblocages

Les motifs de la forme suivante présents à l'intérieur d'un réseau plus grand ne peuvent être modifiés par aucune réduction, on parle d'*interblocage* :

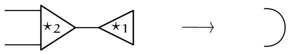


Les boucles, dont nous n'avons pas encore parlé, sont un cas particulier d'interblocage qui ne met en jeu aucun nœud :

Elles peuvent apparaître de différentes façons, par exemple après la réduction du réseau suivant :



lorsque la règle de réduction correspondante est la suivante :



On notera en particulier qu'un interblocage peut apparaître au cours de la réduction ; il n'était pas apparent avant réduction dans cet exemple.

Dans les réseaux de preuve dont il sera bientôt question, on s'arrange pour que ces motifs bloqués n'apparaissent jamais en rajoutant des contraintes de bonne formation pour ces réseaux, et en montrant qu'elles sont préservées par réduction.

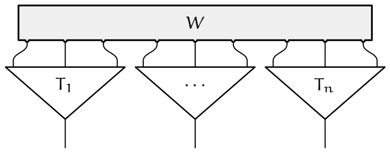
Réseaux sans interblocage

Un réseau sans interblocage est un réseau qui s'écrit sous la forme suivante (on n'a représenté que deux ports libres et un seul rédex, mais il peut y en avoir d'avantage) :

Voir Fig. 1.1 (page 13).

Les symboles $T_1, \ldots, T_4$ désignent des *arbres* de nœuds et W désigne un *câblage*. On rajoute également la contrainte que le lien entre T_3 et T_4 forme un vrai rédex, c'est-à-dire que ces deux arbres ne sont pas réduits à un fil, sans quoi on peut représenter tous les réseaux de cette façon.

Un réseau sans interblocage en forme normale est de la forme :



1.1.4 Propriétés

On étudiera plusieurs systèmes de réseaux d'interaction. Chacun est défini par la donnée d'une signature (un ensemble d'opérateurs munis d'une arité) et

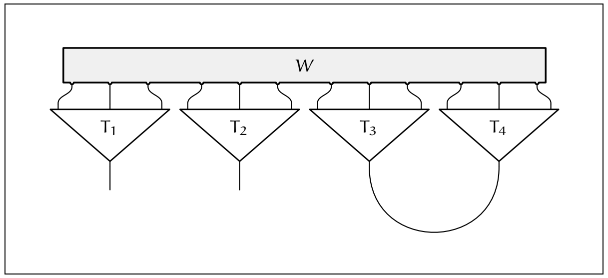


Figure 1.1

un ensemble de règles de réductions pour ces opérateurs.

Confluence

La réduction de ces systèmes est volontairement teintée d'indéterminisme (un réseau admet plusieurs réduits), sans cela il ne serait possible d'exprimer qu'un parallélisme synchrone. Mais cet indéterminisme se manifeste très peu. On peut en effet constater que la forme simple des règles de réduction ne permet pas l'apparition de paires critiques. Cela confère à tout système de réseaux d'interaction une propriété particulièrement intéressante de confluence.

Propriété 1-4. *La réduction $\longrightarrow$ d'un système de réseaux d'interaction est fortement confluente.*

Exprimé de façon concise à l'aide de relations composées, cela s'écrit :

$$\longleftarrow\longrightarrow \subseteq \longrightarrow\longleftarrow$$

En nommant les réseaux impliqués, cela signifie que pour tout N, N_1 *et* N_2, *si* $\mathsf{N} \longrightarrow \mathsf{N}_1$ *et* $\mathsf{N} \longrightarrow \mathsf{N}_2$, *alors il existe* N' *tel que* $\mathsf{N}_1 \longrightarrow \mathsf{N}'$ *et* $\mathsf{N}_2 \longrightarrow \mathsf{N}'$.

Cette propriété ne sera pas automatiquement vérifiée lorsque plus tard on considérera certains systèmes avec boîtes qui sortent légèrement du cadre des réseaux d'interaction.

Réduction contextuelle

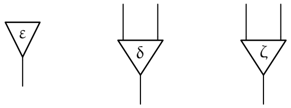
Un contexte C dans le formalisme des réseaux d'interaction est simplement un réseau dans lequel on distingue certains ports libres qui sont destinés à être connectés aux ports libres d'un réseau N pour former un réseau plus grand qu'on désignera par $\mathsf{C}(\mathsf{N})$. Par définition, la réduction des réseaux d'interaction est contextuelle :

Propriété 1-5. *Soit* C *un contexte quelconque. Si* $\mathsf{N} \longrightarrow \mathsf{N}'$ *alors* $\mathsf{C}(\mathsf{N}) \longrightarrow \mathsf{C}(\mathsf{N}')$.

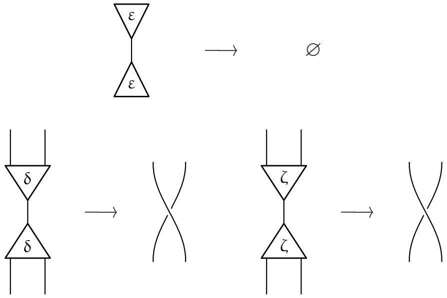
Cette propriété reflète d'une certaine façon le caractère parallèle et asynchrone de ce modèle de calcul. Il est vrai qu'elle ne garantit pas une forte densité de lieux où on peut effectuer une réduction, mais l'atomicité des rédex nous permet néanmoins de l'espérer.

1.1.5 Les combinateurs d'interaction

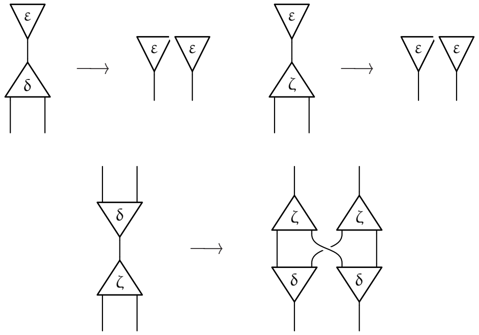
Un système particulier de réseaux d'interaction, constitué de seulement trois opérateurs, possède des propriétés très intéressantes : les combinateurs d'interaction. Certains opérateurs que nous serons amenés à rencontrer plus tard sont fortement similaires à ces combinateurs, et il est utile de comprendre leur fonctionnement. Nous présentons ici la version symétrique de ces combinateurs étudiée par *Damiano Mazza* [Maz07] :



Le combinateur ε est d'arité 0, les combinateurs δ et ζ sont d'arité 2. Deux nœuds identiques interagissent selon les règles décrites ci-dessous. Pour faciliter la lecture, un réseau vide (c'est-à-dire sans nœuds, sans fils et sans ports libres) qui se trouve à droite d'une règle de réduction sera représenté par un symbole ‹ ∅ ›.



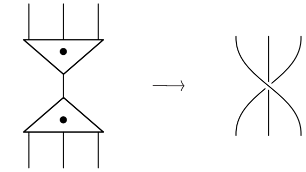
Les autres interactions sont définies comme suit :



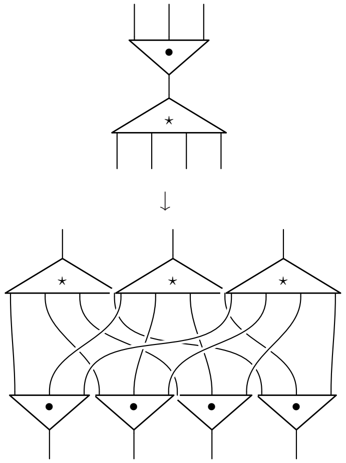
Les combinateurs d'interaction dans leur version non symétrique possèdent une propriété d'universalité : trois opérateurs suffisent pour simuler tout autre système d'interaction. Les combinateurs symétriques que nous venons de présenter possèdent une propriété similaire, quoiqu'un peu plus faible [Laf97 ; Maz07]. L'intérêt que nous trouverons dans ces combinateurs n'est pourtant pas lié directement à cette propriété. Nous nous y intéressons car ils ont des réductions très particulières que nous classifierons sous le nom de *multiplexage*.

1.1.6 Multiplexage

Définition 1-6. *Soit* $\bullet$ *un opérateur d'arité arbitraire (celle-ci sera le plus souvent* 0 *ou* 2*, mais pour conserver une part de généralité nous le représenterons avec une arité* 3*). L'opérateur* $\bullet$ *est dit multiplexant pour un ensemble d'opérateurs* $\mathfrak{D}$ *si ses réductions sont régies par les règles suivantes :*



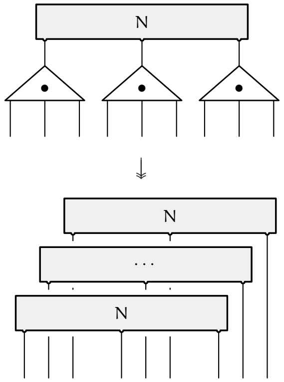
et pour tout $\star \in \mathfrak{D}$, *d'arité arbitraire (on le représente ici avec une arité* 4*) :*



Les combinateurs d'interaction symétriques forment un exemple intéressant de système d'interaction dans lequel on recense des opérateurs multiplexants. En fait, tous les opérateurs le sont : ε est multiplexant pour l'ensemble $\{\delta, \zeta\}$, δ pour l'ensemble $\{\varepsilon, \zeta\}$, et ζ pour l'ensemble $\{\varepsilon, \delta\}$.

La propriété remarquable des opérateurs multiplexants est leur capacité à effacer ou dupliquer un réseau entier. Le chapitre 3 sera consacré à l'exploitation et au contrôle de ces capacités. Elles sont issues de la propriété donnée ci-dessous.

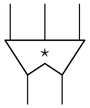
Propriété 1-7. *Un opérateur* $\bullet$ *multiplexant pour tous les opérateurs d'un réseau* N *en forme normale et sans interblocage, qui est branché face à tous les ports libres de ce réseau, « réplique » ce réseau avec la signification suivante :*



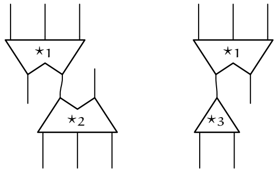
En particulier, un opérateur multiplexant d'arité 2 *duplique* le réseau N, un opérateur multiplexant d'arité 0 *efface* le réseau N, et un opérateur multiplexant d'arité 1 laisse ce réseau inchangé.

1.1.7 Nœuds avec ports principaux multiples

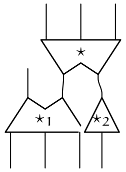
Il nous arrivera de travailler avec des nœuds particuliers qui ont plusieurs ports principaux :



On sortira alors du cadre strict des réseaux d'interaction. Le principe et la forme des règles de réduction restent cependant inchangés : un rédex est un couple de deux nœuds reliés par un fil qui relie un port principal du premier à un port principal du second :



La propriété de confluence forte n'est plus assurée car des paires critiques apparaissent. Par exemple, dans la configuration suivante, le nœud étiqueté ⋆ fait partie de deux rédex différents :



Généralement on parviendra à montrer la confluence locale des systèmes comprenant des multinœuds (on appellera ainsi les nœuds avec plusieurs ports principaux), il suffira pour cela de s'assurer que les paires critiques suscitées convergent.

Notons aussi que les définitions des opérateurs multiplexants et leur propriété s'adaptent à ce cadre élargi des réseaux d'interaction.

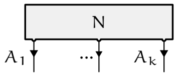
1.2 Réseaux de preuve

Les réseaux de preuve sont des réseaux d'interaction particuliers pour deux raisons. La première est qu'ils sont typés. La seconde est qu'on s'impose des règles de constructions spécifiques qui correspondent à des règles logiques, et qui n'autorisent pas à relier de façon quelconque plusieurs nœuds (même si cela reste en conformité avec le typage).

1.2.1 Typage

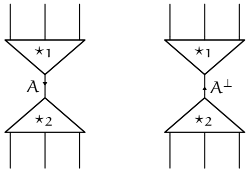
Dans les réseaux de preuve qu'on considère, les ports des nœuds (et les ports libres) sont typés par des formules de la logique linéaire. Le principe de base étant que les fils ne peuvent relier que deux ports typés par des formules duales.

Un réseau N est d'interface $\vdash A_1, \dots, A_k$ lorsqu'il présente k ports libres de types $A_1, \dots, A_k$. Il sera représenté graphiquement de la façon suivante :



Lorsqu'un fil est représenté de façon orientée et qu'il est annoté par un certain type, cela signifie que son port destination est de ce type et que son port

source est du type dual. En particulier, les deux annotations suivantes sont équivalentes :



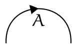
Notons qu'il nous arrivera d'utiliser la notation $A_1, \dots, A_k \vdash A$ pour désigner l'interface $\vdash {A_1}^\perp, \dots, {A_k}^\perp, A$.

1.2.2 Règles d'identité de la logique linéaire

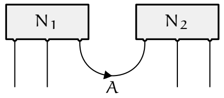
Les constructions autorisées pour les réseaux de preuve vont correspondre aux règles de la logique linéaire. À toute règle $\frac{\vdash \Gamma_1 \quad \dots \quad \vdash \Gamma_k}{\vdash \Delta}$ de cette logique, qui permet de déduire un séquent conclusion $\vdash \Delta$ à partir de séquents hypothèses $\vdash \Gamma_i$, on va associer une construction qui construit un réseau dont l'interface est $\vdash \Delta$ à partir de réseaux déjà construits d'interfaces $\vdash \Gamma_i$.

Les deux premières constructions que l'on va autoriser sont celles qui correspondent aux règles fondamentales dites « d'identité » de la logique linéaire.

Axiome Le réseau associé à la règle axiome $\overline{\vdash A^\perp, A}$ de la logique linéaire est un simple fil reliant deux ports libres.



Coupure Soit N_1 et N_2 deux réseaux d'interfaces respectives $\vdash \Gamma_1, A$ et $\vdash A^\perp, \Gamma_2$. Selon la règle de coupure $\frac{\vdash \Gamma_1, A \quad \vdash A^\perp, \Gamma_2}{\vdash \Gamma_1, \Gamma_2}$, on peut construire un nouveau réseau d'interface $\vdash \Gamma_1, \Gamma_2$; cela consiste à fusionner le port libre de type A du réseau N_1 et le port libre de type $A^\perp$ du réseau N_2 dont il est question.



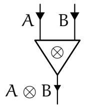
Nous ne considérerons pas ici les règles « *mix* » parfois ajoutées à la logique linéaire :

$$\overline{\vdash} \qquad \frac{\vdash \Gamma_1 \quad \vdash \Gamma_2}{\vdash \Gamma_1, \Gamma_2}$$

une juxtaposition de réseaux de preuve n'est pas en général un réseau de preuve.

1.2.3 Contraintes de typage

Dans le cadre typé, un opérateur ⋆ est non seulement muni d'une arité, mais également d'une contrainte de typage. Les types apparaissant sur des fils qui se raccordent à un même nœud devront obéir à la contrainte imposée par l'opérateur correspondant. Afin d'illustrer cela, anticipons légèrement sur la suite où sera introduit un opérateur ⊗ et présentons sa contrainte de typage :



Les orientations étant données sur la figure, la formule qui type le port principal d'un tel nœud devra être $A \otimes B$ (le symbole ‹ ⊗ › ne désigne pas ici l'opérateur mais le connecteur logique) lorsque les formules qui typent ses deux ports auxiliaires sont A et B.

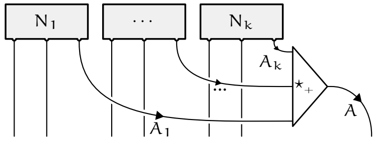
1.2.4 Construction de réseaux

Seules, les règles *axiome* et *coupure* ne permettent bien sûr pas de construire des réseaux qui contiennent des nœuds. Des règles de construction spécifiques devront être spécifiées pour chaque opérateur considéré.

La majorité des opérateurs avec lesquels nous travaillerons appartiennent à deux grandes catégories, les opérateurs *introduits positivement* et les opérateurs *introduits négativement*. Nous allons décrire les constructions qui leur sont associés. Signalons cependant que l'on sera amenés à travailler avec certains opérateurs qui échappent à ces deux catégories. Les constructions correspondantes seront alors explicitement données.

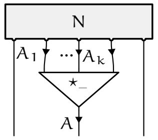
Opérateurs introduits positivement

On dira qu'un opérateur est *introduit positivement* lorsque la construction qui permet de l'introduire est de la forme décrite ci-dessous. Les ports auxiliaires des nœuds correspondants doivent être branchés sur les sorties de réseaux distincts. Par exemple, pour un opérateur $\star_+$ d'arité k, étant donnés k réseaux N_1, …, N_k possédant des liens sortants de type adéquats, le réseau suivant pourra être construit :



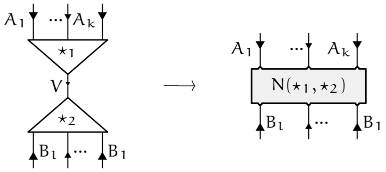
Opérateurs introduits négativement

On dira qu'un opérateur est *introduit négativement* lorsque la construction qui permet de l'introduire est de la forme décrite ci-dessous. Les ports auxiliaires des nœuds correspondants doivent être branchés sur les sorties d'un même réseau. Par exemple, pour un opérateur $\star_-$ d'arité k, étant donné un réseau N possédant des liens sortants de type adéquat, le réseau suivant pourra être construit :



1.2.5 Sémantique opérationnelle

La forme générale d'une règle de réduction est celle des réseaux d'interaction. Un rédex est toujours constitué de deux nœuds liés par leurs ports principaux. En ce qui concerne les types, on impose que l'interface d'un rédex soit préservé par la règle de réduction qui lui correspond. C'est nécessaire pour pouvoir étendre ces règles par mise en contexte.



On notera que les contraintes de typage restreignent les possibilités de rencontre entre opérateurs. En effet, le type V du fil qui relie les deux nœuds du rédex doit satisfaire à la fois la contrainte de typage de l'opérateur $\star_1$ et celle de l'opérateur $\star_2$, or l'unification de ces deux contraintes ne sera pas toujours possible. En pratique cela réduit grandement le nombre de règles de réduction qu'il faut définir.

Toutes les propriétés des réseaux d'interaction que nous avons énoncées sont conservées. Dans un cadre typé, l'interface externe d'un réseau est bien sûr préservée par réduction.

Nous étudierons des systèmes où malgré la réduction tous les réseaux obtenus restent valides (il peuvent être construits à partir des règles autorisées), et d'autres où il est nécessaire de prolonger suffisamment la réduction d'un réseau pour retrouver un réseau valide. Dans ce dernier cas on décrit les étapes intermédiaires en manipulant des réseaux d'interaction typés (le typage est lui préservé par réduction) qui ne correspondent pas directement à des preuves.

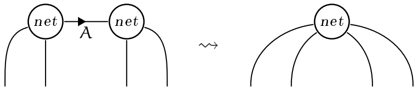
1.2.6 Critère de validité

On peut reconnaître si un réseau quelconque peut être formé grâce aux règles de construction classiques que nous avons évoquées. La méthode que nous présentons apparaît dans la thèse de *Vincent Danos* [Dan90], elle a également été étudiée par *Yves Lafont* [Laf95] puis étendue par *Stefano Guerrini* et *Andrea Masini* [GM01].

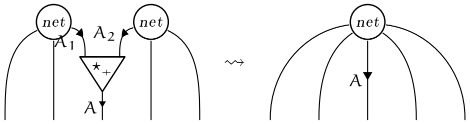
On introduit à cette fin des nœuds multi-arité, que nous nommerons ici ‹ *net* ›, qui vont représenter des sous réseaux déjà validés. On considère ensuite les règles de réécriture suivantes, à chacune correspond l'utilisation d'une construction valide.

Axiome

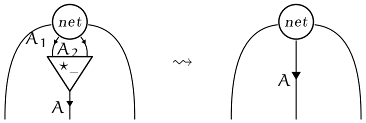
Coupure



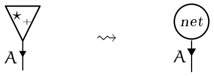
Nœud-Binaire-Positif



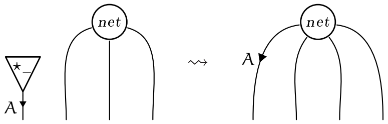
Nœud-Binaire-Négatif



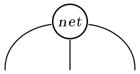
Nœud-Nullaire-Positif



Nœud-Nullaire-Négatif



Un réseau est valide s'il existe une suite de réécritures ⇝ qui le réduit à un unique nœud *net* :



Cette méthode permet non seulement de vérifier la validité d'un réseau, mais elle permet également de retrouver une suite de constructions qui permet de construire ce réseau, c'est-à-dire une preuve logique qui lui correspond. Il peut d'ailleurs exister plusieurs façons de construire un même réseau, mais l'information que l'on perd lorsqu'on projette une preuve en le réseau d'interaction correspondant n'est généralement pas intéressante. Il suffit de savoir qu'il en existe une pour assurer la validité du réseau.

Les choses deviendront plus complexes lorsque nous serons amenés à introduire des boîtes, mais on fera en sorte que la réduction dans ces réseaux préserve un critère de validité similaire. Sans un tel critère nous ne serions pas capables d'assurer une propriété importante de la plupart de nos systèmes : la terminaison de la réduction.

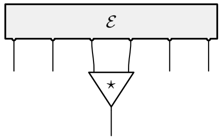
D'autres critères

Nous avons présenté un critère qui fonctionne dans un cadre très général, et qui est de type « existentiel » : il repose sur l'existence d'une suite de réécritures.

D'autres critères sont de type « universel » et fournissent un ensemble particulier de conditions que doivent vérifier un réseau pour être valide. Historiquement le premier critère de bonne formation de ce type, qui garantit également l'absence d'interblocages, a été donné par *Jean-Yves Girard* dans le cadre des réseaux de preuve multiplicatifs et en considérant seulement des nœuds binaires. Les réseaux de preuve multiplicatifs sont l'objet du prochain chapitre. Un critère plus simple dû à *Vincent Danos* et *Laurent Regnier* [DR89] est à présent le plus célèbre. Il en existe d'autres, et dans des cas particuliers certains fournissent des algorithmes très efficaces pour vérifier la validité d'un réseau.

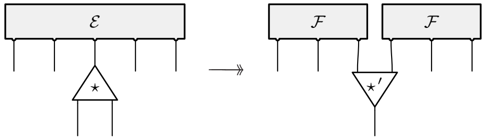
1.3 Conventions

Nous considérerons à plusieurs reprises des ensembles de réseaux, et nous souhaiterons souvent exprimer des propriétés pour tous les réseaux de ces ensembles. Si $\mathcal{E}$ désigne un ensemble de réseaux (de même interface) alors la notation :

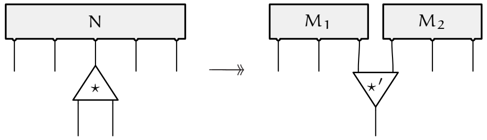


désigne un nouvel ensemble de réseaux, celui qu'on obtient en choisissant des éléments dans l'ensemble $\mathcal{E}$ pour remplir l'emplacement correspondant.

Il nous arrivera aussi de vouloir exprimer des réductions qui parlent d'ensembles de réseaux. Par exemple, si $\mathcal{E}$ et $\mathcal{F}$ sont deux ensembles, il nous arrivera d'écrire :



Lorsqu'on utilise une cellule rectangulaire étiquetée par un ensemble de réseaux, cela désigne implicitement l'un de ses éléments. Par défaut, la quantification sera universelle dans les parties gauches de règles de réduction et existentielle ailleurs. On comprendra donc : pour tout $N \in \mathcal{E}$, il existe $M_1 \in \mathcal{F}$ et $M_2 \in \mathcal{F}$ tel que :



En particulier, lorsqu'avec ces notations un même ensemble apparaît plusieurs fois, les éléments choisis pour chacune de ses occurrences peuvent tout à fait être distincts.

Chapitre 2

Réseaux multiplicatifs

L'objectif de ce chapitre est d'introduire les quatre opérateurs élémentaires dits « multiplicatifs » qui vont nous permettre de programmer dans le langage des réseaux d'interaction. Pour se convaincre de l'utilité de ces « briques » élémentaires nous allons montrer qu'elles permettent de simuler une partie significative du λ-calcul : le λ-calcul linéaire enrichi avec des constructions tensorielles.

Cela permet déjà d'exprimer quelques petits programmes de la vie réelle dans lesquels on n'effectue aucune duplication et aucun effacement des données, comme la manipulation de permutations (ici sur trois éléments) :

```
let id =
  fun (x, y, z) -> (x, y, z)

let swap p =
  fun (x, y, z) -> p (y, x, z)

let rotate_left p =
  fun (x, y, z) -> p (y, z, x)

let rotate_right p =
  fun (x, y, z) -> p (z, x, y)

let result =
  rotate_left (swap (rotate_right id))
```

Pour atteindre la puissance complète du λ-calcul, nous aurons besoin d'introduire de nouveaux opérateurs dits « exponentiels », mais ce sera l'objet du prochain chapitre.

2.1 Le λ-calcul linéaire

Le λ-calcul linéaire est un sous-calcul du λ-calcul ordinaire dans lequel chaque variable introduite par une abstraction est utilisée une et une seule fois. On va commencer par la présentation de ce calcul, mais c'est aussi et surtout un prétexte à l'introduction de notations et de définitions qui seront réutilisées tout au long de ce document.

2.1.1 Présentation

Soit Σ un ensemble fini de variables. On utilisera les noms x, y, z... pour représenter ces variables. On utilisera aussi $\bar{w}$ pour désigner des ensembles variables. La notation $\bar{w}_1, \bar{w}_2$ désignera la réunion des deux ensembles de variables $\bar{w}_1$ et $\bar{w}_2$ lorsque ceux-ci sont supposés (le plus souvent implicitement) d'intersection vide. Enfin, l'ensemble contenant l'unique variable x sera simplement noté x, et l'ensemble vide sera noté $\varnothing$. Un ensemble de variables sera donc écrit comme une suite sans répétition $w_1, \ldots, w_n$, mais en réalité ce ne sont pas des suites, l'ordre des éléments n'importe pas.

Définition 2-1. *Le λ-calcul linéaire est défini grâce à la syntaxe suivante, dans laquelle $t_{\bar{w}}$ représente un terme du λ-calcul linéaire dont les variables libres sont les éléments de $\bar{w}$.*

$$t_x \quad ::= \quad x$$

$$t_{\bar{w}} \quad ::= \quad \underline{\lambda} x\, t_{x,\bar{w}} \qquad t_{\bar{w}_1,\bar{w}_2} \quad ::= \quad (t_{\bar{w}_1})t_{\bar{w}_2}$$

On utilisera t, mais également s, u, v, ... pour désigner des termes. Les termes sans variables libres seront appelés termes clos.

Il est possible de former le terme $\underline{\lambda} x\, \underline{\lambda} y\, (x)y$, en revanche il n'est pas possible de former le terme $\underline{\lambda} x\, \underline{\lambda} y\, ((x)y)y$, ni le terme $\underline{\lambda} x\, \underline{\lambda} y\, x$ en raison des contraintes de linéarité : on n'utilise que la réunion de deux ensembles disjoints de variables, et on a imposé qu'une variable apparaisse sous son lieur $\langle\, \underline{\lambda} x\, \rangle$. Or dans le premier terme mal formé la variable y est utilisée deux fois sous le même lieur, et dans le second elle n'est pas utilisée du tout.

Définition 2-2. *La taille $|t|$ d'un terme t est définie par le nombre de constructions utilisées pour le former (contrairement à ce qui se fait le plus souvent, nous préférons ici ne pas compter l'utilisation d'une variable) :*

$$|x| \;:=\; 0 \qquad |\underline{\lambda} x\, t| \;:=\; |t| + 1$$

$$|(s)t| \;:=\; |s| + |t| + 1$$

Substitution linéaire

Par convention on supposera que les variables liées (non libres) ne sont jamais réutilisés à l'extérieur de leur lieur. On s'y ramène par le procédé standard d'α-conversion des calculs avec lieurs.

Définition 2-3. *On note* $t[u/z]$ *la substitution d'une variable* z *par un terme* u *dans un terme* t*. Elle n'est définie dans le cas des* λ*-termes linéaires que lorsque les variables libres de* t *et* u *sont deux à deux distinctes. On procède de façon inductive selon la structure de* t *:*

$$x[{}^{u}/_{z}] \;:=\; x \qquad z[{}^{u}/_{z}] \;:=\; u$$
$$(\underline{\lambda} x\, t)[{}^{u}/_{z}] \;:=\; \underline{\lambda} x\, t[{}^{u}/_{z}]$$
$$((s)t)[{}^{u}/_{z}] \;:=\; (s[{}^{u}/_{z}])t[{}^{u}/_{z}]$$

Le résultat de cette substitution n'est pas un λ-terme linéaire si contrairement à ce qui est demandé t et u possèdent des variables libres communes. Précisons aussi que dans les bonnes conditions de substitution, $t[u/z]$ admet pour variables libres l'union de celles de u et de celles de t sauf z.

On qualifie cette substitution de « linéaire » parce qu'exactement une occurrence de variable est à substituer.

Réduction

Pour établir une dynamique intéressante dans notre calcul nous allons également expliquer la notion de *clôture contextuelle.* Une relation $\succ$ est close par contexte lorsqu'elle est stable par ajout d'un contexte autour des termes :

$$t \succ t' \;\Rightarrow\; \underline{\lambda} x\, t \succ \underline{\lambda} x\, t' \qquad t \succ t' \;\Rightarrow\; (s)t \succ (s)t'$$
$$s \succ s' \;\Rightarrow\; (s)t \succ (s')t$$

Les intersections et les unions (quelconques) de plusieurs relations closes par contexte sont closes par contexte. Cela nous permettra de parler de la plus petite (ou de la plus grande) relation close par contexte qui vérifie une propriété. On parlera dans ce cas de clôture contextuelle engendrée par des relations élémentaires. Une relation d'équivalence close par contexte est appelée *congruence.*

Définition 2-4. *La réduction d'un terme se fait par substitution. On définit la relation* $\longrightarrow$ *par clôture contextuelle de la réduction élémentaire :*

$$(\underline{\lambda} x\, s)t \;\longrightarrow\; s\left[{}^{t}/_{x}\right]$$

Et on ne manquera pas de remarquer que dans un cadre linéaire cette réduction a le bon goût de terminer.

Théorème 2-5. *Dans le* λ*-calcul linéaire la réduction* $\longrightarrow$ *termine.*

Preuve. Pour une substitution linéaire on a $|t[u/z]| = |t| + |u|$. Dès lors, on montre que si $t \longrightarrow t'$, alors $|t'| = |t| - 2 < |t|$. Mais la taille des termes étant bornée inférieurement, une suite infinie de réductions n'est pas possible. □

2.1.2 Extension tensorielle

Contrairement à ce qu'il se passe dans le λ-calcul ordinaire, il n'est pas possible de coder la construction et la déconstruction de couples au sein de ce calcul. Pourtant ce concept n'interfère en rien avec l'idée de linéarité. Nous proposons d'ajouter ces possibilités en étendant la syntaxe.

Définition 2-6. *Le* λ*-calcul linéaire tensoriel est obtenu en rajoutant les constructions d'introduction de* 0*-uplets et de* 2*-uplets (communément appelés élément* unité *et* couples*) :*

$$t_{\varnothing} \;::=\; \langle\rangle \qquad t_{\bar{w}_1,\bar{w}_2} \;::=\; \langle t_{\bar{w}_1}, t_{\bar{w}_2}\rangle$$

et les éliminations qui leur sont associées :

$$t_{\bar{w},\bar{w}'} \;::=\; \underline{\gamma}\langle\rangle\; t_{\bar{w}} \cdot t_{\bar{w}'}$$
$$t_{\bar{w},\bar{w}'} \;::=\; \underline{\gamma}\langle x,y\rangle\; t_{x,y,\bar{w}} \cdot t_{\bar{w}'}$$

Une construction $\underline{\gamma}\langle x,y\rangle\; s \cdot t$ rend accessible les deux composantes d'un couple t sous les noms x et y dans s. La construction $\underline{\gamma}\langle\rangle\; s \cdot t$ permet juste d'oublier un élément unité t. Ces constructions existent dans le langage interne des catégories autonomes [MRA93], elles peuvent aussi être utilisées dans un cadre non linéaire.

Définition 2-7. *On définit les tailles des termes qui utilisent les nouvelles constructions en complétant la*

définition précédente :

$$|\langle\rangle| \;:=\; 1 \qquad |\langle u,v\rangle| \;:=\; |u| + |v| + 1$$

Les introductions et éliminations de n-uplets, lorsque n est un entier arbitraire peuvent être codées à l'aide de ces nouvelles constructions. On commence à pouvoir exprimer des programmes qui ont un peu d'intérêt. Si on représente les triplets `(x, y, z)` du programme donné en exemple par des expressions $\langle x, \langle y,z\rangle\rangle$, on arrive a exprimer toutes les fonctions évoquées :

$$\mathtt{id} \;:=\; \underline{\lambda} c\, \underline{\gamma}\langle x,c'\rangle\; \left(\underline{\gamma}\langle y,z\rangle\; \langle x,\langle y,z\rangle\rangle \cdot c'\right) \cdot c$$
$$\mathtt{swap} \;:=\; \underline{\lambda} f\, \underline{\lambda} c\, \underline{\gamma}\langle x,c'\rangle\; \left(\underline{\gamma}\langle y,z\rangle\; (f)\, \langle y,\langle x,z\rangle\rangle \cdot c'\right) \cdot c$$

$$\mathtt{rotate\_left} \;:=$$
$$\underline{\lambda} f\, \underline{\lambda} c\, \underline{\gamma}\langle x,c'\rangle\; \left(\underline{\gamma}\langle y,z\rangle\; (f)\, \langle y,\langle z,x\rangle\rangle \cdot c'\right) \cdot c$$

$$\mathtt{rotate\_right} \;:=$$
$$\underline{\lambda} f\, \underline{\lambda} c\, \underline{\gamma}\langle x,c'\rangle\; \left(\underline{\gamma}\langle y,z\rangle\; (f)\, \langle z,\langle x,y\rangle\rangle \cdot c'\right) \cdot c$$

La définition de `result` s'exprime alors en λ-calcul linéaire par l'expression (`rotate_left`)(`swap`) (`rotate_right`)`id`.

Réduction

Nous allons à présent compléter la dynamique de façon à ce que cette expression complexe se réduise en le résultat espéré, qui s'exprime simplement :

$$\underline{\lambda} c\, \underline{\gamma}\langle x,c'\rangle\; \left(\underline{\gamma}\langle y,z\rangle\; \langle z,\langle y,x\rangle\rangle \cdot c'\right) \cdot c$$

Définition 2-8. *Les règles de réduction qu'on ajoute pour réduire les termes du* λ*-calcul linéaire tensoriel sont les suivantes :*

$$\underline{\gamma}\langle\rangle\; s \cdot \langle\rangle \;\longrightarrow\; s$$
$$\underline{\gamma}\langle x,y\rangle\; s \cdot \langle u,v\rangle \;\longrightarrow\; s\left[{}^{u,v}/_{x,y}\right]$$

où $s[u,v/x,y]$ *désigne de façon identique* $(s[u/x])[v/y]$ *ou* $(s[v/y])[u/x]$ *lorsqu'on a pris des représentants* x *et* y *qui n'apparaissent pas dans* u *et* v*.*

On notera que la dynamique de la réduction est simplement bloquée lorsqu'on rencontre des termes :

- de la forme $\underline{\gamma}\langle\rangle\; s \cdot (\underline{\lambda} z\, t)$ ou $\underline{\gamma}\langle x,y\rangle\; s \cdot (\underline{\lambda} z\, t)$,
- de la forme $\underline{\gamma}\langle\rangle\; s \cdot \langle u,v\rangle$ ou $\underline{\gamma}\langle x,y\rangle\; s \cdot \langle\rangle$,
- et également ceux de la forme $(\langle\rangle)t$ ou $(\langle u,v\rangle)t$.

Théorème 2-9. *Dans le* λ*-calcul linéaire tensoriel la réduction* $\longrightarrow$ *termine.*

Preuve. Comme précédemment, si $t \longrightarrow t'$, alors $|t'| = |t| - 2 < |t|$ pour la nouvelle notion de taille. □

2.1.3 Types linéaires

Un système de types va être introduit, il permettra en particulier d'éviter toute configuration inopportune qui bloquerait la réduction (ces configurations ne seront pas typables). Les types que nous utilisons sont formés à partir de types de base α par :

$$\tau \;::=\; \alpha \;\mid\; \tau \multimap \tau \;\mid\; 1 \;\mid\; \tau \otimes \tau$$

On utilisera la notation $t^{\tau}_{\bar{w}}$ pour désigner un terme typé de type τ dans un environnement $\bar{w} = z_1{:}\sigma_1, \ldots, z_n{:}\sigma_n$ qui spécifie les types σ_1, …, σ_n de ses variables libres z_1, …, z_n. Les termes typés peuvent être formés selon :

$$t^{\sigma}_{x:\sigma} \;::=\; x \qquad t^{\sigma\multimap\tau}_{\bar{w}} \;::=\; \underline{\lambda} x\, t^{\tau}_{x:\sigma,\bar{w}}$$
$$t^{\tau}_{\bar{w}_1,\bar{w}_2} \;::=\; \left(t^{\sigma\multimap\tau}_{\bar{w}_1}\right) t^{\sigma}_{\bar{w}_2}$$

ou selon les constructions tensorielles :

$$t^{1}_{\varnothing} \;::=\; \langle\rangle \qquad t^{\tau_1\otimes\tau_2}_{\bar{w}_1,\bar{w}_2} \;::=\; \left\langle t^{\tau_1}_{\bar{w}_1}, t^{\tau_2}_{\bar{w}_2}\right\rangle$$
$$t^{\tau}_{\bar{w},\bar{w}'} \;::=\; \underline{\gamma}\langle\rangle\; t^{\tau}_{\bar{w}} \cdot t^{1}_{\bar{w}'}$$
$$t^{\tau}_{\bar{w},\bar{w}'} \;::=\; \underline{\gamma}\langle x,y\rangle\; t^{\tau}_{x:\tau_1,y:\tau_2,\bar{w}} \cdot t^{\tau_1\otimes\tau_2}_{\bar{w}'}$$

La substitution linéaire $s[t/x]$ n'est définie pour les termes typés que lorsque $s^{\tau}_{\bar{w},x:\sigma}$ et $t^{\sigma}_{\bar{w}'}$ (comprendre : lorsque s désigne un terme de type τ dans un environnement $\bar{w}, x{:}\sigma$ et t désigne un terme de type τ dans un environnement $\bar{w}'$). Elle désigne alors un terme de type τ dans un environnement $\bar{w}$.

Les réductions que nous avons données :

$$(\underline{\lambda} x\, s)t \;\longrightarrow\; s\left[{}^{t}/_{x}\right]$$
$$\underline{\gamma}\langle\rangle\; s \cdot \langle\rangle \;\longrightarrow\; s$$
$$\underline{\gamma}\langle x,y\rangle\; s \cdot \langle u,v\rangle \;\longrightarrow\; s\left[{}^{u,v}/_{x,y}\right]$$

s'adaptent à ce calcul typé. Dans la première règle le rédex est bien construit lorsqu'on suppose $s^{\tau}_{\bar{w},x:\sigma}$ et $t^{\sigma}_{\bar{w}'}$, cette règle relie des termes qui sont tous les deux typés τ dans un environnement $\bar{w},\bar{w}'$. Dans la deuxième règle on suppose $s^{\tau}_{\bar{w}}$, elle relie des termes qui sont tous les deux typés τ dans un environnement $\bar{w}$. Dans la troisième règle on suppose $s^{\tau}_{\bar{w},x:\tau_1,y:\tau_2}$, $u^{\tau_1}_{\bar{w}_1}$ et $u^{\tau_2}_{\bar{w}_2}$, elle relie des termes qui sont tous les deux typés τ dans un environnement $\bar{w},\bar{w}_1,\bar{w}_2$.

C'est ce calcul qu'on peut espérer coder dans le système de réseaux d'interaction que nous allons introduire maintenant.

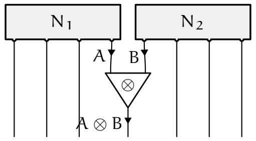
2.2 Le système multiplicatif

Sans l'indiquer, nous sommes partis du λ-calcul et nous y avons ajouté des restrictions. Le sous-calcul obtenu n'est donc pas apparu de façon très naturelle. Dans le formalisme des réseaux d'interaction, nous allons voir qu'on peut atteindre de façon simple et concise ce même niveau d'expressivité. C'est pour dépasser ce niveau élémentaire qu'il faudra introduire des considérations plus complexes.

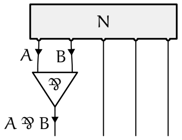
2.2.1 Opérateurs multiplicatifs

Les opérateurs multiplicatifs sont au nombre de quatre. Les nœuds binaires *tenseur* et *par*, notés $\langle \otimes \rangle$ et $\langle \parr \rangle$, ainsi que leur correspondants nullaires *unité* et *co-unité*, notés $\langle 1 \rangle$ et $\langle \perp \rangle$, proviennent directement de la logique linéaire. Ils vont être détaillés ici.

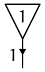
Tenseur Soit N_1 et N_2 deux réseaux d'interfaces respectives $\vdash \Gamma_1, A$ et $\vdash B, \Gamma_2$. Selon la règle logique $\frac{\vdash \Gamma_1, A \quad \vdash B, \Gamma_2}{\vdash \Gamma_1, A \otimes B, \Gamma_2}$, on peut introduire un nœud *tenseur* dont les deux ports auxiliaires seront branchés sur les sorties de deux réseaux distincts. C'est un nœud binaire introduit positivement.



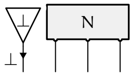
Par Soit N un réseau d'interface $\vdash A, B, \Gamma$. Selon la règle $\frac{\vdash A, B, \Gamma}{\vdash A \parr B, \Gamma}$, on peut introduire un nœud *par* dont les deux ports auxiliaires seront branchés sur les sorties d'un même réseau. C'est un nœud binaire introduit négativement.



Unité Un nœud *unité* peut être introduit à partir de rien selon la règle $\overline{\vdash 1}$. C'est un nœud nullaire introduit positivement.



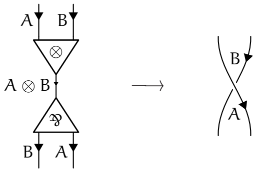
Co-unité Étant donné un réseau N d'interface $\vdash \Gamma$, un nœud *co-unité* peut être introduit selon la règle $\frac{\vdash \Gamma}{\vdash \perp, \Gamma}$. C'est un nœud nullaire introduit négativement.



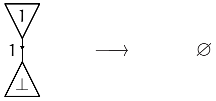
2.2.2 Réduction des opérateurs multiplicatifs

Selon le typage des nœuds multiplicatifs, seuls deux rédex sont possibles. Les règles de réduction qui leur sont associées sont les suivantes :

Par — Tenseur Les nœuds *tenseur* et *par* se réduisent en connectant leurs ports auxiliaires correspondants.



Co-unité — Unité Les nœuds *unité* et *co-unité* s'annihilent, ils disparaissent du réseau originel.



Ces règles sont écrites dans le formalisme des réseaux d'interaction, et les réseaux qu'elles mettent en jeu ne sont pas bien construits. Cependant, on peut montrer sans difficultés que lorsqu'on les utilise pour réduire des réseaux valides on obtient des réseaux valides.

Propriétés

La confluence de la réduction est assurée automatiquement par le formalisme des réseaux d'interaction. Une fois de plus (et c'est la dernière fois…), une simple mesure de la taille des objets que l'on manipule suffit à prouver la terminaison de la réduction qu'on considère.

Théorème 2-10. *La réduction des réseaux multiplicatifs termine.*

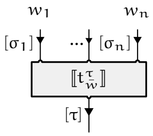
Preuve. Le nombre de nœuds d'un réseau diminue de 2 à chaque étape de réduction. □

2.3 Utilisation des réseaux

Nous allons constater que la règle entre nœuds *tenseur* et *par* permet à elle seule de modéliser l'application d'une fonction linéaire à son argument, ou bien l'accès aux composantes d'un couple.

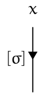
2.3.1 Codage du λ-calcul linéaire

On peut en effet traduire le λ-calcul linéaire tensoriel typé (vu en 2.1.3) à l'aide des opérateurs multiplicatifs. Un terme $t^{\tau}_{\bar{w}}$ avec $\bar{w} = w_1{:}\sigma_1, \ldots, w_n{:}\sigma_n$ sera codé en un réseau :

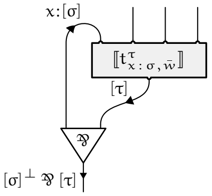


où $[\cdot]$ désigne une traduction triviale des types : les connecteurs 1 et $\otimes$ sont interprétés par leurs correspondants, et la flèche linéaire est interprétée par $[\sigma \multimap \tau] = [\sigma]^{\perp} \parr [\tau]$. La définition de ce codage est faite inductivement sur la structure du terme :

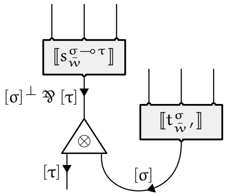
$[\![x^{\sigma}_{x:\sigma}]\!] :=$



$[\![\underline{\lambda} x\, t^{\tau}_{x:\sigma,\bar{w}}]\!] :=$

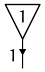


$[\![(s^{\sigma \multimap \tau}_{\bar{w}})\, t^{\sigma}_{\bar{w}'}]\!] :=$

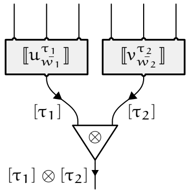


Les constructions tensorielles sont codées comme suit :

$[\![\langle\rangle]\!] :=$

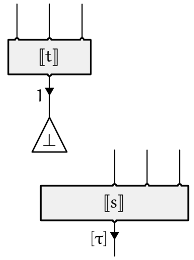


$[\![\langle u^{\tau_1}_{\bar{w}_1}, v^{\tau_2}_{\bar{w}_2}\rangle]\!] :=$

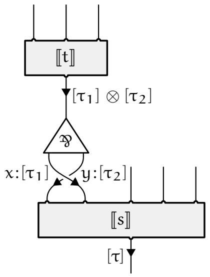


et :

$$[\![\underline{\gamma}\langle\rangle\ s\cdot t]\!] \ :=$$



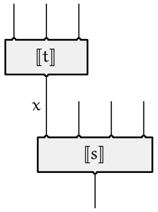
$$[\![\underline{\gamma}\langle x,y\rangle\ s\cdot t]\!] \ :=$$



On remarque que la traduction d'une substitution se fait de façon particulièrement simple (c'est du au fait que les substitutions qu'on considère sont linéaires), elle nous est donnée par le lemme ci-dessous.

Lemme 2-11. *La traduction d'un terme décrit par une substitution linéaire* $s[t/x]$ *est le réseau obtenu en branchant la traduction du terme* t *sur le lien correspondant à la variable* x *dans la traduction du terme* s *:*

$$[\![s[{}^{t}/_{x}]]\!] \ =$$



Preuve. On procède par induction sur le terme s. □

Théorème 2-12. *La réduction des réseaux simule (fortement) la réduction du λ-calcul linéaire tensoriel typé : si* $t \longrightarrow t'$ *alors* $[\![t]\!] \longrightarrow [\![t']\!]$.

Preuve. On utilise le lemme précédent et on vérifie facilement cette propriété pour chacune des trois formes de réduction. □

2.3.2 Intérêt de la syntaxe graphique

Il y a plusieurs avantages à travailler dans la syntaxe graphique des réseaux plutôt qu'avec la syntaxe traditionnelle du λ-calcul. Celle-ci évite une certaine « bureaucratie » : plusieurs λ-termes similaires sont représentés par le même réseau, et un plus grand nombre de réductions peuvent être menées en parallèle.

La σ-équivalence définie par *Laurent Regnier* dans [Reg94] qui permet de faire apparaître des rédex cachés s'adapte au cas des λ-termes linéaires.

Définition 2-13. *La* σ-équivalence *sur les λ-termes linéaires, notée* $\equiv_\sigma$, *est la congruence engendrée par les équivalences élémentaires suivantes :*

$$((\underline{\lambda} z\, s)u)t \ \equiv_\sigma\ (\underline{\lambda} z\,(s)t)u$$
$$(s)(\underline{\lambda} z\, t)u \ \equiv_\sigma\ (\underline{\lambda} z\,(s)t)u$$
$$\underline{\lambda} x\,(\underline{\lambda} z\, t)u \ \equiv_\sigma\ (\underline{\lambda} z\,\underline{\lambda} x\, t)u$$

avec la convention que les variables liées ne sont jamais réutilisées à l'extérieur de leur lieur.

Chacune de ces équivalences associe deux termes qui ont la même traduction dans les réseaux multiplicatifs. On a pu rajouter la deuxième équivalence uniquement parce qu'on est dans un cadre linéaire : nous verrons que les termes correspondant du λ-calcul usuel peuvent aussi être traduits, mais ils n'auront pas exactement la même traduction.

Dans l'extension tensorielle on peut également ajouter :

$$\langle(\underline{\lambda} z\, t)u, v\rangle \ \equiv_\sigma\ (\underline{\lambda} z\ \langle t, v\rangle)u$$
$$\langle v, (\underline{\lambda} z\, t)u\rangle \ \equiv_\sigma\ (\underline{\lambda} z\ \langle v, t\rangle)u$$

$$\underline{\gamma}\langle\rangle\ s\cdot(\underline{\lambda} z\, t) \ \equiv_\sigma\ (\underline{\lambda} z\,\underline{\gamma}\langle\rangle\ s\cdot t)u$$
$$\underline{\gamma}\langle\rangle\ \underline{\lambda} z\, s\cdot t \ \equiv_\sigma\ (\underline{\lambda} z\,\underline{\gamma}\langle\rangle\ s\cdot t)u$$

$$\underline{\gamma}\langle x,y\rangle\ s\cdot(\underline{\lambda} z\, t) \ \equiv_\sigma\ (\underline{\lambda} z\,\underline{\gamma}\langle x,y\rangle\ s\cdot t)u$$
$$\underline{\gamma}\langle x,y\rangle\ \underline{\lambda} z\, s\cdot t \ \equiv_\sigma\ (\underline{\lambda} z\,\underline{\gamma}\langle x,y\rangle\ s\cdot t)u$$

On pense que cette équivalence est l'équivalence induite par les réseaux : deux termes sont σ-équivalents si et seulement s'ils ont même traduction dans les réseaux.

Chapitre 3

Réseaux exponentiels

À ce jour, les langages de programmation de la vraie vie considèrent que toute chose existante peut être dupliquée à volonté et effacée au besoin. En effet, si cela possède une représentation mémoire, alors on peut facilement le dupliquer… ou l'effacer… Cette « commodité » n'est pas anecdotique, on peut enfin envisager faire des multiplications, et beaucoup plus… Le codage des entiers proposé par *Alonzo Church* illustre cela.

```
let zero =
  fun f x -> x

let one =
  fun f x -> f x

let two =
  fun f x -> f (f x)

let succ n =
  fun f x -> n f (f x)

let add n m =
  fun f x -> (n f) ((m f) x)

let mult n m =
  fun f x -> m (n f) x

let result =
  mult two two
```

Syntaxiquement ces possibilités de duplication et d'effacement se traduisent par une utilisation libre des variables. La variable nommée `f` a été utilisée plusieurs fois dans la fonction `two` et oubliée dans la définition de `zero`.

Remarque 3-1. *Il serait intéressant de se demander s'il est vraiment légitime d'autoriser la réplication n'importe quel objet. C'est problématique lorsqu'on veut modéliser un objet de la vie réelle par un objet informatique, puisque une opération de duplication ou d'effacement n'est hélas pas forcément évidente dans le monde physique.* Mais *il peut aussi être intéressant d'interdire la réplication de certains objets même s'ils sont de nature purement informatique. Par exemple, empêcher la réplication de certaines références permettrait de se passer d'un surcoût dû à un mécanisme de « garbage collection ». On peut aussi garantir simplement l'atomicité de l'écriture sur un déscripteur de fichier pourvu que l'on empêche toute duplication de celui-ci.*

La logique linéaire est, elle, capable de parler à la fois d'objets répliquables et d'objets qui ne le sont pas. Une paire de connecteurs duaux est utilisée pour représenter cette capacité. Si une formule A est habitée par certaines preuves, la formule $!A$ fournit de telles preuves en quantité arbitraire, adaptée au besoins de son utilisation. De façon duale, la formule $?A$ fournit une preuve de A en quantité arbitraire, mais c'est alors l'utilisateur de s'adapter à la quantité qu'il reçoit (éventuellement nulle).

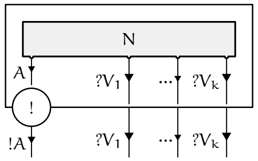
Le calcul habituel qui exploite cette capacité de réplication des données est le λ-calcul, dans lequel on n'ajoute aucune contrainte de linéarité cette fois. Celui-ci a été beaucoup étudié et est particulièrement bien connu. Nous rappellerons comment il est possible de l'interpréter dans les réseaux de preuves de la logique linéaire en utilisant des boîtes exponentielles, qui sortent du cadre des réseaux d'interaction. Nous évoquerons ensuite diverses façons de ramener tout cela dans le formalisme des réseaux d'interaction. La réduction doit s'y exprimer de façon locale, or ce n'est pas le cas des boîtes dans le formalisme des réseaux de preuve.

3.1 Réseaux de preuve avec boîtes exponentielles

Nous allons commencer par une présentation des réseaux de preuve de la logique linéaire avec constructions exponentielles.

3.1.1 Constructions exponentielles

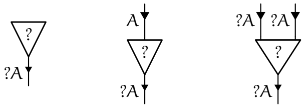
Promotion Soit N un réseau d'interface $\vdash A, ?\Gamma$. On peut construire selon la règle logique $\frac{\vdash A, ?\Gamma}{\vdash !A, ?\Gamma}$, un réseau d'interface $\vdash !A, ?\Gamma$. On matérialise cette *promotion* à l'aide d'une boîte exponentielle, le réseau obtenu sera représenté graphiquement comme suit :



Une telle boîte doit être considérée comme un seul nœud du réseau extérieur. Sa *sortie* est un port de type $!A$, ses *entrées* sont des ports de types $?V_i$.

En termes de ressources, on considérera une boîte exponentielle comme une usine qui fournit, de façon adaptée à la demande, un nombre arbitraire de produits. Ces produits sont des ressources de type A, et l'usine se fournit elle-même auprès de sous-traitants, qui lui affrètent des ressources de type $V_i{}^\perp$.

Affaiblissement, Déréliction, Contraction Face à nos boîtes qui offrent des liens de type $!A$, nous allons considérer les trois opérateurs suivants, qui seront introduits négativement :



L'*affaiblissement* est nullaire, la *déréliction* est unaire et la *contraction* est binaire. Ce sont ces opérateurs qui vont permettre de sélectionner le nombre d'objets de type A désiré.

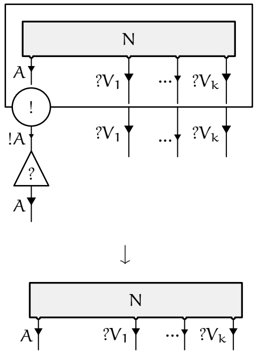
3.1.2 Réduction des boîtes

Précisons tout d'abord qu'une boîte est un nœud particulier dont tous les ports sont principaux. Ces ports donnent par conséquent lieu à des réductions lorsque d'autres nœuds (ou boîtes) se présentent face à eux. Il faudra ici faire attention au fait que l'on sort du cadre des réseaux d'interaction. La conservation de la propriété de confluence (et celle-ci ne sera plus nécessairement forte) doit être explicitement vérifiée car des paires critiques non triviales apparaissent.

Réductions principales

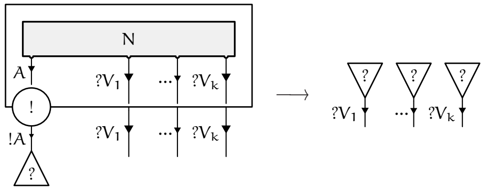
Nous commençons par les réductions qui interviennent au niveau de la sortie d'une boîte, lorsque l'un des trois opérateurs évoqués la rencontre.

Sortie-Promotion — Déréliction Lorsqu'une *déréliction* rencontre une *promotion*, elle ouvre simplement la boîte.



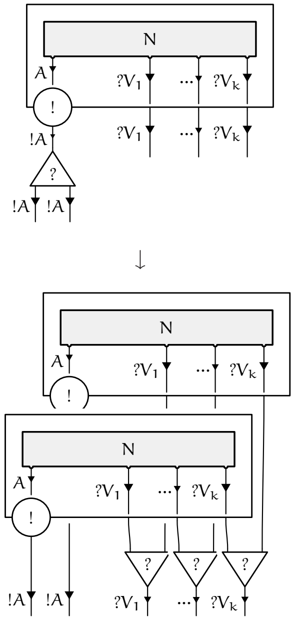
En termes de ressources, la *déréliction* est un consommateur qui demande à l'usine (matérialisée par la boîte exponentielle) de lui fournir exactement une instance du produit qu'elle propose. Celle-ci libère son ultime produit et laisse à ce produit la possibilité d'utiliser les ressources fournies par ses sous-traitants.

Sortie-Promotion — Affaiblissement Lorsqu'un *affaiblissement* rencontre une boîte, il supprime cette boîte, et lance des *affaiblissements* sur les entrées de la boîte.



Un *affaiblissement* est un consommateur qui ne demande aucun produit à l'usine qui le fournit. L'usine disparaît et ses sous-traitants sont informés qu'elle n'aura dans ce cas eu besoin d'aucune ressource.

Sortie-Promotion — Contraction Lorsqu'une *contraction* rencontre une boîte, elle duplique cette boîte, en laçant des *contractions* sur les entrées de la boîte.

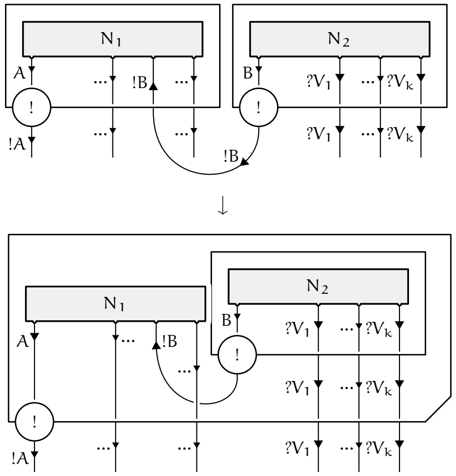


En termes de ressources, une *contraction* est un consommateur multiple. En quelque sorte, une usine qui la fournit doit se scinder, et les filiales (identiques) obtenues se partagent dans ce cas les ressources affrétées par les sous-traitants.

Commutation exponentielle

Une réduction de commutation a lieu lorsque des opérateurs se présentent face aux entrées des boîtes exponentielles. Au vu du système de types, seule une autre boîte exponentielle peut se présenter face à une telle entrée. La dernière règle de réduction régit donc l'interaction entre deux boîtes exponentielles.

Entrée-Promotion — Sortie-Promotion Une boîte arrivant sur l'entrée d'une autre boîte rentre dans cette deuxième.



En termes de ressources, cette étape de réduction n'a pas d'interprétation précise. Cela pourrait correspondre au fait que la première usine à racheté sa sous-traitante.

3.1.3 Propriétés

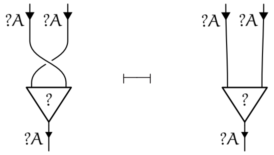
On montre facilement que ce système est localement confluent. Il est aussi fortement normalisant, mais la preuve n'est pas évidente (on rappellera plus tard la preuve de normalisation donnée par *Jean-Yves Girard* pour la normalisation faible, voir 11.2). On en déduit une propriété de confluence globale.

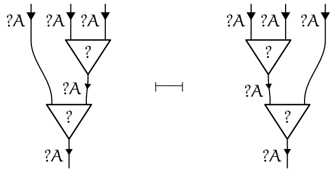
La propriété de normalisation forte permet l'emploi de la notation $N{\downarrow}$ pour désigner l'unique réseau de preuve en forme normale qu'on peut obtenir par réductions successives du réseau N.

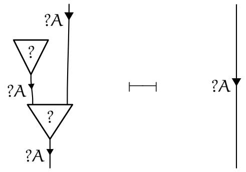
Remarque 3-2. *Les réductions entre constructions axiomes et coupures sont invisibles dans la présentation que nous avons donnée. Ce n'est pas le cas dans la formulation originelle des réseaux de preuve, mais il ne s'agit là que d'un détail, cela ne change rien de fondamental.*

3.1.4 Équivalences

On va introduire une notion d'équivalence opérationnelle. Deux réseaux seront équivalents lorsqu'ils se comportent de façon identique du point de vue de la réduction. On constate par exemple qu'on peut mettre les réseaux suivants en correspondance :







Ces trois correspondances concernent les opérateurs structurels de *contraction* et d'*affaiblissement*, elles rappellent certaines propriétés algébriques de *commutativité*, d'*associativité* et de *neutralité* d'un élément.

La relation $\longmapsto$ qu'on considère est la relation obtenue à partir de ces trois correspondances élémentaires lorsqu'on les étend naturellement par mise en contexte : si $\mathsf{N}_1 \longmapsto \mathsf{N}_2$ alors $\mathsf{C}(\mathsf{N}_1) \longmapsto \mathsf{C}(\mathsf{N}_2)$ pour tout contexte C. On notera $\Vdash\!\!\longmapsto\!\!\Vert$ la clôture réflexive et transitive de la relation $\longmapsto$, c'est une relation d'équivalence. On la qualifie d'équivalence opérationnelle car elle vérifie une propriété de cohérence locale pour la réduction $\longrightarrow$ qu'on a défini.

Propriété 3-1. *La réduction* $\longrightarrow$ *est localement cohérente avec la relation* $\longmapsto$*, c'est-à-dire :*

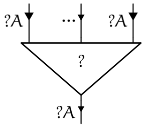
$$\longmapsto\longrightarrow \;\subseteq\; \twoheadrightarrow\Vdash\!\!\longmapsto\!\!\Vert\twoheadleftarrow$$

Preuve. (idée) Le typage fait que ces réseaux ne peuvent interagir qu'avec des boîtes de *promotion*, et qu'il n'est donc pas nécessaire de considérer d'autres contextes. On traite chacune des cinq possibilités de réduction. □

La notion de cohérence locale est utilisée dans un théorème de confluence donné par *Gérard Huet*. Nous aurons l'occasion d'utiliser ce théorème pour une autre équivalence que celle considérée ici.

Dans le cas présent, chacune de nos trois correspondances élémentaires pourrait générer indépendamment des deux autres une relation d'équivalence qui a cette propriété. Mais rien ne nous empêche de considérer une équivalence opérationnelle qui les regroupe (de façon générale, l'union de deux relations qui satisfont la propriété de cohérence locale pour une réduction donnée possède toujours cette propriété).

Par ailleurs, ces équivalences font qu'il serait naturel de considérer des macro-constructions pour le produit qu'elles évoquent, qui peuvent avoir une arité quelconque :



Celles-ci représenteraient l'ensemble des arbres de *contraction* éventuellement suivis d'*affaiblissements* qui offrent un nombre adéquat de fils libres. Ils sont tous équivalents.

3.1.5 Réductions restreintes

Nous allons définir deux sous réductions de la réduction générale que nous avons présentée, elles seront utilisées dans la suite.

Définition 3-2. *On note* $\xrightarrow{\dagger}$ *la réduction* sans commutations, *qui interdit l'utilisation de la règle de commutation exponentielle.*

La deuxième sous réduction que nous allons considérer est proche de la réduction standard définie par *Jean-Yves Girard* qui n'effectue certaines réductions de boîtes que lorsque leur contenu est en forme normale [Gir87, page 72]. Nous allons ici généraliser cette restriction à toutes les réductions qui font intervenir des boîtes.

Définition 3-3. *Nous définissons la réduction interne* $\xrightarrow{\star}$ *exactement comme la réduction habituelle, mais les quatre règles de réduction des boîtes, qui ont étés données en 3.1.2, sont munies d'une contrainte supplémentaire. On ne permet l'utilisation de ces règles que lorsque le contenu des boîtes qui interagissent sur leur sortie sont déjà en forme normale.*

En particulier, dans cas de la règle de commutation exponentielle, seul le contenu de la boîte ingérée (le réseau qu'on a nommé N_2) est supposé en forme normale. Dans le cas où on interdit complètement ces commutations exponentielles on note la réduction correspondante $\xrightarrow{\dagger\star}$.

On peut montrer que ces restrictions sont localement confluentes, et par conséquent fortement normalisantes car ce sont des restrictions d'une réduction elle-même fortement normalisante. On peut donc définir $\mathsf{N}{\downarrow_\star}$, $\mathsf{N}{\downarrow_\dagger}$, $\mathsf{N}{\downarrow_{\dagger\star}}$, les opérations de normalisation d'un réseau N qui correspondent à ces réductions. On a par ailleurs $\mathsf{N}{\downarrow_\star} = \mathsf{N}{\downarrow}$ et $\mathsf{N}{\downarrow_{\dagger\star}} = \mathsf{N}{\downarrow_\dagger}$ car grâce à la confluence, on peut toujours choisir de normaliser le contenu des boîtes en premier.

3.1.6 Problématique

La dynamique donnée par la réduction est directement liée à la notion de calcul. Elle « simplifie » les réseaux de la même manière qu'un calcul « simplifie » un programme. On peut juger que chaque étape du calcul que l'on vient de décrire est relativement simple. Notamment lorsqu'on considère les réductions qui ne font intervenir que des nœuds simples, il ne paraît pas étonnant qu'on puisse traiter matériellement ces étapes de calcul avec seulement quelques instructions d'un processeur tout à fait classique. Le cas des opérations sur les boîtes nécessite pourtant réflexion. Des boîtes de tailles arbitraires peuvent apparaître, et le coût d'une opération sur les boîtes n'est donc pas borné. Lorsqu'on souhaite paralléliser le calcul, il faut autant que possible éviter les longues opérations qui bloquent le reste du calcul. C'est l'une des raisons pour lesquelles nous souhaitons « localiser » le calcul, et c'est l'objet des sections qui suivent.

3.2 Le système exponentiel localisé actif

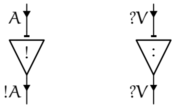
Dans cette section, nous allons proposer un système localisé et asynchrone (c'est-à-dire qui n'utilise que des nœuds normaux, pas de boîtes) qui essaye de mimer le fonctionnement du système avec boîtes. On va reproduire le fonctionnement des boîtes en utilisant des nœuds dédiés. C'est le premier de trois formalismes qui ont été développés dans ce but. C'est le plus concis dans sa formulation et c'est pourquoi nous le présentons en premier. En réalité cette concision est plutôt un défaut car elle en fait également le plus difficile à comprendre. Nous donnerons des intuitions quant à son fonctionnement, mais nous attendrons d'avoir étudié les deux autres formalismes avant de déduire des propriétés pour ce système.

Notre premier système (dit *actif*) est en réalité une version alternative des *sharing graphs*, dont *John Lamping* est le précurseur [Lam90], et qui ont été étendus à la logique linéaire par *Georges Gonthier*, *Martín Abadi* et *Jean-Jacques Lévy* [GAL92] par. Il possède certainement la même propriété d'optimalité que ce dernier pour la réduction séquentielle. Il a l'avantage de préserver une plus grande partie de la structure logique des réseaux, et cela permettra plus tard (cf. 3.5) de comprendre comment l'étendre pour éliminer les problèmes de « relecture ».

La comparaison avec les systèmes de *Ian Makie* et *Jorge Sousa Pinto* [MS02] est également intéressante. Certes, notre système utilise un nombre infini d'opérateurs, mais nous souhaitons conserver un système optimal. En outre, aucun contenu de boîte n'est « figé », et cela augmente significativement le degré de parallélisation potentiel. De plus, grâce à une meilleure préservation de la structure logique, nous parviendrons à définir une certaine sémantique dénotationnelle pour notre modèle (cf. 12.3).

3.2.1 Opérateurs exponentiels actifs

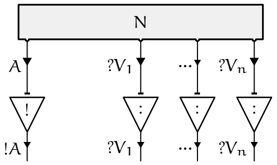
Sortie-Promotion $_\mathfrak{a}$, ***Entrée-Promotion*** $_\mathfrak{a}$ On introduit deux opérateurs, les sorties de *promotion* (actives), et les entrées de *promotion* (actives). Ils seront graphiquement représentés avec des cellules « renforcées » qui nous laissent imaginer la limite d'une boîte qui n'existe pourtant pas. Ce sont des opérateurs avec *niveau incrémenté*, nous verrons plus tard ce que cela signifie.



Les opérateurs d'*affaiblissement*, de *déréliction* et de *contraction* restent quant à eux inchangés.

Traduction des boîtes exponentielles

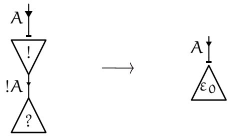
Promotion$_{a}$ On remplace les boîtes exponentielles contenant un réseau N par la construction suivante :



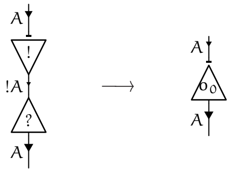
3.2.2 Réduction des opérateurs exponentiels

Dans la logique linéaire, les interactions qui se produisent au niveau des sorties de boîtes se traduisent en des modifications de la boîte considérée comme un tout. Ici, les modifications correspondantes vont être effectuées seulement au niveau des têtes de boîte, de façon locale. On explicitera en premier ces réductions. Elles s'inscrivent dans le cadre originel des réseaux d'interaction, et font intervenir de nouveaux nœuds, dits « opérateurs de boîtes », dénotés par ε_0, o_0, δ_0 et ι_0. Ces derniers font partie de familles ε_i, o_i, δ_i et ι_i d'opérateurs indexés par $i \in \mathbb{N}$, qui seront explicités juste après, en 3.2.3. Ce sont ces nœuds qui, suite à de nouvelles étapes de réduction, propageront l'effet des modifications attendues en profondeur dans la structure interne des boîtes.

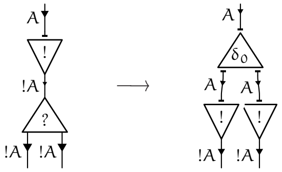
Affaiblissement — Sortie-Promotion$_{a}$ Un *affaiblissement* face à une sortie de boîte lance un opérateur *effacement* qui commence à effacer le reste de la boîte.



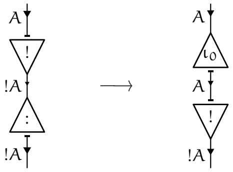
Déréliction — Sortie-Promotion$_{a}$ Une *déréliction* face à une sortie de boîte lance un opérateur *ouverture* qui commence à ouvrir le reste de la boîte.



Contraction — Sortie-Promotion$_{a}$ Une *contraction* face à une sortie de boîte lance un opérateur *duplication* qui commence à dupliquer le reste de la boîte.



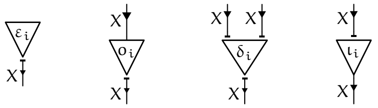
Entrée-Promotion$_{a}$ — ***Sortie-Promotion***$_{a}$ Lorsque une sortie de boîte se présente face à l'entrée d'une autre boîte, cette sortie entre dans cette seconde boîte et lance un opérateur *ingestion* qui commence le processus d'ingestion pour le reste de la boîte.



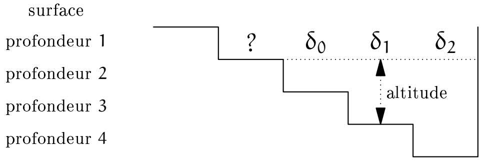
3.2.3 Opérateurs de boîtes

Les réductions précédentes ont fait intervenir quatre sortes d'opérateurs particuliers : les *effacements* ε_0 nullaires, les *ouvertures* o_0 unaires avec *niveau décrémenté*, les *duplications* δ_0 binaires, ainsi que les *ingestions* ι_0, unaires et avec *niveau incrémenté*. Ils obéissent tous à une règle de typage simple : les ports auxiliaires ont tous le même type, le type dual de celui qui est attribué au port principal.

Nous aurons en réalité besoin d'indexer ces opérateurs par un élément de $\mathbb{N}$ dit *altitude*, pour écrire les réductions attendues. Nous définissons donc les opérateurs ε_i, o_i, δ_i et ι_i pour $i \in \mathbb{N}$. Les opérateurs de boîtes d'altitude i seront graphiquement représentés comme suit :



L'altitude d'un opérateur qui agit sur une certaine boîte est nulle lorsqu'il est aux abords des limites externes de cette boîte ; elle va augmenter d'une unité avec chaque entrée dans une sous boîte. Voici un petit schéma qui tente d'expliquer la situation. Il explique la duplication d'une boîte de profondeur 2 par une contraction ‹ ? › située dans la boîte de profondeur 1 qui la contient.



L'opération de duplication est propagée par des opérateurs δ_0 dans la boîte de profondeur 2 dont il est question, puis des opérateurs δ_1 dans ses sous-boîtes, etc... dans les sous-boîtes de celles-ci. C'est en réalité un peu plus compliqué que ça car les sous-boîtes sont elles-mêmes soumises à cette dynamique et des changements d'altitude devront être effectués lorsque l'opérateur traverse une *ouverture* ou une *ingestion* qui manipule une autre boîte.

Remarque 3-3. *L'approche dans les « sharing graphs » est similaire, au lieu d'indexer les opérateurs duplication par une altitude (relative à la boîte sur laquelle on opère), l'opérateur « fan » qui joue le même rôle est indexé par une profondeur (absolue cette fois), cela permet également de distinguer les nœuds issus d'opérations de duplication distinctes. Ces opérateurs « fan » servent aussi à représenter les contractions. Les « croissants » correspondent à la fois aux opérateurs ouverture et aux déréliction, les « crochets » correspondent à la fois aux opérateurs ingestion et aux sorties de promotions. Ce point de vue identifie un trop grand nombre d'opérateurs et altère le contenu logique de la représentation. Il n'a pas permis de découvrir l'intérêt du canal de contrôle dont il sera question plus tard, qui en particulier offre une solution efficace au problème de la relecture.*

3.2.4 Réduction des opérateurs de boîtes

Les opérateurs ε_i, o_i, δ_i, ι_i, pour toute altitude i, sont des opérateurs de multiplexage (comme défini précédemment) pour la plupart des nœuds sans altitude et des nœuds d'altitude j strictement inférieure. Ce ne sera exactement le cas lorsqu'ils rencontrent des nœuds avec *niveau incrémenté* :

- les entrées et sorties de *promotions*
- les opérateurs d'*ingestion* ι_j

ou avec *niveau décrémenté* :

- les opérateurs d'*ouverture* o_j

On veut que l'altitude des opérateurs de boîtes varie quand ils traversent des opérateurs avec changement de niveau. On appellera ce multiplexage un peu modifié, qui fait intervenir des familles d'opérateurs, le multiplexage *à niveaux*. Les propriétés qu'on obtient pour celui-ci seront présentées en 3.2.7.

Concrètement, les opérateurs ε_i, o_i, δ_i, ι_i sont régis par les méta-règles décrites ci-dessous. Dans ces règles, on considère l'un de ces opérateurs de boîtes, i désignera son altitude, et $\star_j$ désignera un autre opérateur. Il peut s'agir :

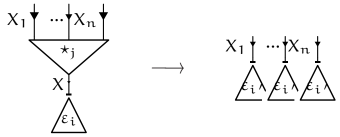
- d'un autre opérateur de boîte, d'altitude j que l'on suppose strictement inférieure à l'altitude du premier : $j < i$
- ou d'un opérateur qui n'est pas un opérateur de boîte et qui est donc sans altitude (on peut considérer que $j = -\infty$)

L'altitude i' désignera quant à elle :

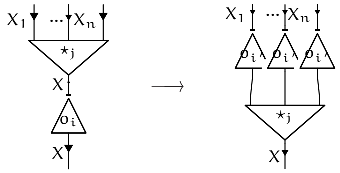
- $i+1$ si $\star_j$ est un opérateur avec niveau incrémenté,
- $i-1$ si $\star_j$ est un opérateur avec niveau décrémenté,
- ou i dans le cas standard.

Les quatre premières règles de réduction couvrent toutes les possibilités de rencontre pour les opérateurs de boîte, sauf lorsqu'il s'agit d'une interaction entre opérateurs de boîtes de même altitude.

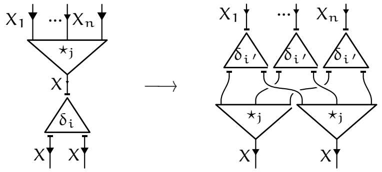
***Effacement* — $\star$**



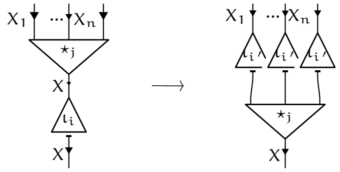
***Ouverture* — $\star$**



***Duplication* — $\star$**



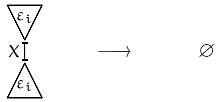
***Ingestion* — $\star$**



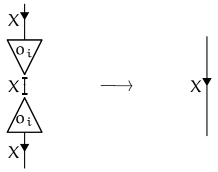
Réductions entre opérateurs de même altitude

L'action de ces nœuds se propage dans la boîte qui leur est associée. Deux nœuds identiques, lorsqu'ils se rencontrent de part et d'autre de ce qui était un axiome, connectent leur liens et disparaissent. C'est le démultiplexage habituel évoqué dans le premier chapitre :

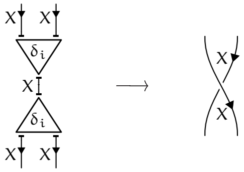
Effacement — Effacement



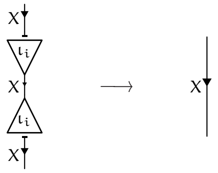
Ouverture — Ouverture



Duplication — Duplication

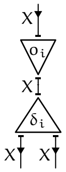


Ingestion — Ingestion



Les rédex qui n'apparaissent pas

Nous avons traité les réductions de tous les rédex compatibles avec le typage, hormis celles qui font intervenir deux opérateurs de boîte différents de même altitude. Mais on peut remarquer que les opérations que subit une boîte sont toujours amorcées au niveau de leur unique sortie. Cela laisse imaginer que des opérateurs de boîtes de même altitude vont parcourir le contenu de la boîte les uns derrière les autres, sans interférence, et que par exemple la configuration suivante n'est jamais rencontrée (pour tout i) :



Cela n'est bien sûr vrai que si on part de réseaux bien construits. Nous prouverons ce point plus tard (cf. corollaire 3-16). En attendant, on considérera la réduction localement bloquée lorsque des rédex de ce type se présentent. Ils n'ont pas de règle de réduction associée.

3.2.5 Récapitulatif pour le système localisé actif

Nous avons présenté tous les opérateurs et toutes les réductions de notre premier système localisé.

Définition 3-4. *On nomme* système exponentiel localisé actif *le système de réseaux d'interaction (très souvent utilisé conjointement avec système multiplicatif vu en 2.2) qui est constitué par :*

- *les opérateurs affaiblissement, contraction et déréliction*
- *les opérateurs entrées et sorties de promotion actives*
- *les opérateurs de boîtes d'altitude $i \in \mathbb{N}$: effacement, duplication, ouverture et ingestion*

et muni des réductions :

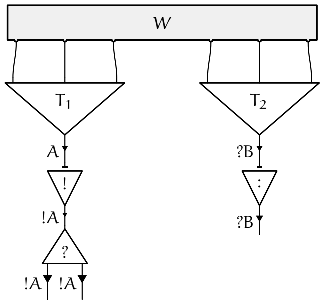
- *qui amorcent les opérations à effectuer sur les boîtes, données en 3.2.2*
- *de multiplexage à niveau des opérateurs de boîtes, données en 3.2.4, qui propagent ces opérations dans la structure interne des boîtes.*

3.2.6 Équivalences opérationnelles

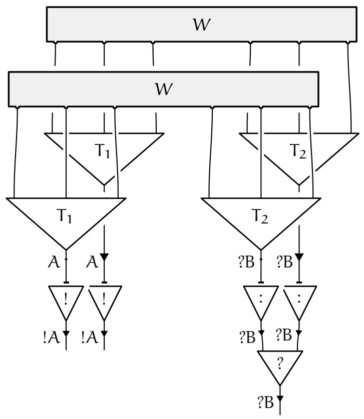
Les opérations sur les boîtes restent inachevées

Un opérateur de boîte se propage dans la boîte sur laquelle il opère. Les réductions de multiplexage font apparaître plusieurs copies de cet opérateur. Certaines de ces copies vont se rencontrer de part et d'autre de ce qui était un axiome et vont disparaître par application de l'une des règles données en 3.2.4. D'autres copies parviendront jusqu'aux ports auxiliaires de leurs limites de boîte, ou seulement sur les ports auxiliaires d'autres nœuds qui les séparent de celles-ci. Ces ports sont précisément auxiliaires et donc aucune réduction n'a lieu.

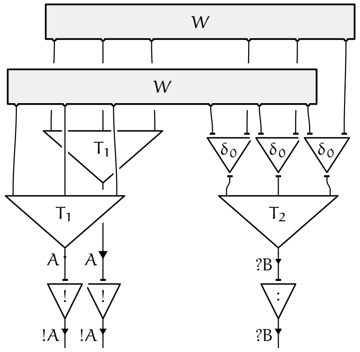
Pour illustrer cela, présentons le cas typique correspondant à la duplication d'une boîte localisée. On simplifie en supposant qu'elle n'a qu'une entrée. On suppose de plus qu'elle contient un réseau qui est déjà en forme normale et qui s'écrit donc avec deux arbres de nœuds T_1 et T_2 et un câblage W (voir 1.1.3) :



Voici le résultat que l'on obtiendrait si la *contraction* dupliquait entièrement la boîte :



Or avec la réduction localisée active, seule la sortie de boîte et la partie T_1 du réseau que la boîte contient vont pouvoir être dupliquées, le reste de la duplication reste en attente :



Nous allons maintenant décrire certaines équivalences opérationnelles qui permettent de comprendre pourquoi ces réseaux, dans lesquels certaines opérations sur les boîtes ne sont pas achevées, sont équivalents aux réseaux que l'on obtiendrait si cette opération avait été effectuée jusqu'au bout.

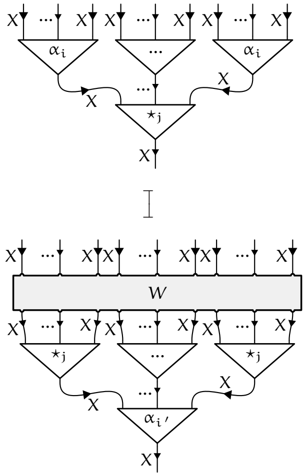
Ce sont des équivalences qu'on aimerait orienter et intégrer au processus de réduction pour terminer jusqu'au bout l'opération effectuée sur une boîte. Le système passif de la section suivante est d'ailleurs né de cette remarque, mais il s'arrange pour faire cela avec la réduction usuelle des réseaux d'interaction.

Remarque 3-4. *La réplication du câblage W ne se passe en réalité pas tout à fait comme décrit ci-dessus lorsque des axiomes relient certains liens auxiliaires de l'arbre T_2. Nous passons sur ce petit détail qui nécessiterait la description d'une équivalence supplémentaire.*

Commutation des opérateurs de boîtes

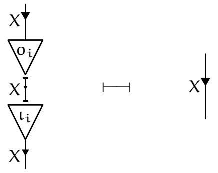
L'exemple que nous venons de donner nous suggère de faire passer les opérateurs δ_0 au travers de l'arbre T_2 en le dupliquant. Bien que ça ne puisse être exprimé par une réduction des réseaux d'interaction, c'est possible au moyen d'une équivalence opérationnelle. Cette équivalence opérationnelle nous sera d'ailleurs utile plus tard pour prouver un théorème de confluence.

Les opérateurs de boîtes se comportent de façon identique lorsqu'on effectue la commutation suivante :



Ici α_i représente l'un des opérateurs de boîtes ε_i, o_i, δ_i ou ι_i, et $\star_j$ désigne un opérateur sans altitude, ou d'altitude $j < i$. On a nommé W le câblage qui joue le rôle de la permutation nécessaire pour que chaque lien rejoigne la racine du réseau par les mêmes ports auxiliaires au niveau des nœuds $\star$ et α, malgré que leur ordre ait été inversé. L'altitude i' est définie une fois de plus selon la variation de niveau associée à l'opérateur $\star_j$, mais ici le sens de la réduction est inversé. On prendra donc $i' = i+1$ si $\star_j$ est avec niveau décrémenté, $i' = i-1$ si $\star_j$ est avec niveau incrémenté et $i' = i$ dans le cas standard.

La deuxième équivalence importante est :

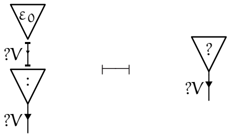


Ces équivalences peuvent être considérées indépendamment de toutes les autres. Les opérateurs de boîtes seront présents et inchangés dans les systèmes localisés qui seront étudiés après, et sur lesquels il sera plus facile de travailler. Les deux formes d'équivalences que nous venons de voir ne dépendent pas de la réduction des autres opérateurs et seront toujours vérifiées.

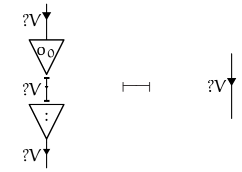
Équivalences aux limites de boîtes

Revenons brièvement à notre exemple. Après avoir hypothétiquement dupliqué l'arbre T_2 il nous faudrait utiliser une dernière équivalence pour achever la duplication de la boîte dont on est parti. On s'intéresse donc ici à des équivalences opérationnelles remarquables lorsqu'un opérateur de boîte d'altitude nulle atteint la frontière interne d'une boîte (la boîte sur laquelle il opère). Pour montrer la cohérence locale de la réduction avec cette équivalence il faut considérer en même temps les commutations des opérateurs de boîtes données dans le paragraphe précédent.

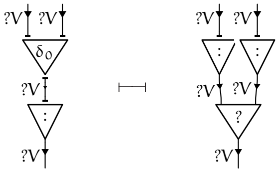
Effacement → Entrée-Promotion$_a$



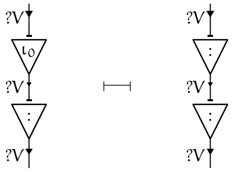
Ouverture → Entrée-Promotion$_a$



Duplication → Entrée-Promotion$_a$



Ingestion → Entrée-Promotion$_a$



Contrairement aux équivalences de commutation des opérateurs de boîtes, ces équivalences aux limites de boîtes entreront en conflit avec la structure qui sera ajoutée dans les systèmes suivants. Elles ne sont donc valides que dans le système actif.

Remarque 3-5. *Les équivalences données en 3.1.4 pour le système avec boîte posent certaines difficultés dans les systèmes localisés. Les opérateurs δ_0 ne peuvent pas être commutés, il faudrait commuter en même temps toutes les copies de ces opérateurs qui correspondent à une même opération. Cette impossibilité se répercute sur les opérateurs contraction, et la réduction localisée n'est plus cohérente avec les équivalences qui existaient dans le système avec boîtes. De façon rassurante, lorsqu'on adoptera un point de vue sémantique, ces équivalences seront valides sémantiquement, même dans les systèmes localisés (voir le chapitre sur la sémantique localisée, 12.3).*

3.2.7 Multiplexage à niveaux

On a déjà étiqueté les fils des réseaux par des types, pour mieux comprendre notre système, nous pouvons leur rajouter une information : une profondeur décrite par un entier. Ces profondeurs seront comme les types soumises à des contraintes. Les ports des nœuds logiques normaux (sans changement de niveau) devront être reliés par des fils de même profondeur. Les ports auxiliaires des nœuds avec niveau incrémenté (respectivement décrémenté) devront être reliés par des fils dont la profondeur est une unité supérieure (inférieure) au fil qui est relié à son port principal.

De façon peu étonnante, toutes les règles de réduction sont compatibles avec cette notion de profondeur : un rédex est réduit sans changer la profondeur des liens de son interface. Si on suppose qu'un réseau en forme normale (et sans interblocage) ne contient que des nœuds dont l'altitude est strictement inférieure à la profondeur du fil branché sur leurs ports principaux, on retrouve une propriété de multiplexage. Des opérateurs ε_i, δ_i, o_i ou ι_i branchés sur les ports libres d'un tel réseau, chacun ayant une altitude égale à la profondeur du fil sur lequel il est branché, réplique ce réseau.

Nous ne donnons pas plus de détails car en réalité nous n'aurons pas recourt à un tel raffinement du multiplexage. Il fournit cependant l'intuition qui est présente derrière les règles présentées plus haut, et c'est pourquoi nous jugeons nécessaire de le citer.

Multiplexage à niveaux pour réseaux valides

Nous ne considérerons que le cas où le réseau N à répliquer est valide et ne contient donc pas d'autres opérateurs avec altitude. Il peut néanmoins contenir des opérateurs avec changement de niveau : les sorties ou les entrées de promotion. Aussi, lorsqu'un réseau est valide, on sait déjà qu'il ne contient aucun interblocage, car il est de la forme présentée en 1.1.3.

Propriété 3-5. *Soit* N *un réseau qui est supposé valide et en forme normale. Soit* $\{\bullet_i\}_{i\in\mathbb{N}}$ *une famille d'opérateurs avec altitude qui est multiplexante (au sens du multiplexage à niveau) pour tous les opérateurs du réseau* N. *Lorsque l'un des opérateurs* $\bullet_i$ *est branché face à tous les ports libres du réseau* N, *il « réplique » ce réseau avec la signification habituelle.*

Preuve. La validité du réseau N nous permet montrer cette propriété par induction sur sa structure. Comme N est supposé en forme normale, on n'a pas besoin de traiter le cas de la construction *coupure*. Le cas de l'*axiome* est simple. Les autres constructions (sans boîtes) font chacune intervenir un nœud qui est immédiatement répliqué, et les réplications des sous réseaux qu'elles contiennent sont assurées par hypothèse d'induction. Le cas des boîtes est tout petit peu plus compliqué : les entrées et sorties d'une boîte contenant un réseau P sont immédiatement répliquées, mais elles sont avec niveau incrémenté et ce sont donc des opérateurs $\bullet_{i+1}$ qui se retrouvent face aux liens de P. Mais l'hypothèse de récurrence s'applique encore, ces opérateurs répliquent le contenu P de la boîte. □

Nous n'utiliserons cette propriété que plus tard, car dans le système actif les conditions requises pour l'appliquer ne sont pas réunies.

3.3 Le système exponentiel localisé passif

Dans cette section, nous allons proposer un nouveau système localisé et asynchrone dit *passif*. Il sera lui aussi équivalent au système avec boîtes, mais on oubliera cette fois la dernière règle de réduction : la règle de commutation exponentielle donnée en 3.1.2. On va simuler le fonctionnement des boîtes en considérant les ports associés aux entrées comme des ports auxiliaires. La réduction devient proche de celle du λ-calcul en appel par nom.

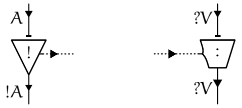
Si on compare son fonctionnement avec celui du système actif que nous venons de présenter, on notera des similarités mais aussi de sensibles différences. Il s'inscrit comme le premier dans le cadre simple des réseaux d'interaction, mais dans ce nouveau système les opérations effectuées sur les boîtes sont opérées jusqu'au bout (sans invoquer d'équivalence opérationnelle). De ce fait, il contourne naturellement un problème dont nous n'avons pas encore parlé qui est celui de la « relecture », ainsi que des problèmes connexité liés à celui-ci. Ces problèmes sont décrits par *Stefano Guerrini*, *Simone Martini*, et *Andrea Masini* dans [GMM03] ; une autre façon de les contourner y est donnée. Pour la même raison, il est aussi plus facile de le relier au système avec boîtes, et c'est ce que nous allons faire dès que nous l'aurons présenté. Les opérateurs de boîtes précédemment introduits seront réutilisés, mais par chance, dans ce système passif (et pour celui-là uniquement) il est possible d'oublier leurs altitudes sans modifier la réduction. On pourrait donc travailler avec un jeu d'opérateurs fini.

Le principe de ce nouveau système repose sur une structure supplémentaire, un canal de contrôle, qui pourra être mélangée au formalisme actif pour donner un troisième système : le système localisé *contrôlé* dont nous parlerons dans la section suivante. C'est avec ce dernier qu'on obtiendra le meilleur degré de parallélisme. Il permettra également de mieux comprendre les deux autres.

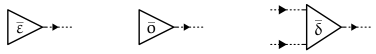
Le système présenté dans [Mac04] par *Ian Mackie* contient une ébauche du canal de contrôle que nous allons introduire ici. Néanmoins, placé dans le cadre du λ-calcul son concept manque d'une certaine clarté.

3.3.1 Opérateurs exponentiels passifs

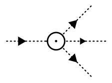
Sortie-Promotion$_p$, ***Entrée-Promotion***$_p$ Les opérateurs exponentiels passifs sont semblables aux opérateurs actifs : ils vont servir à matérialiser sorties et les entrées de boîtes. Ils possèdent cependant un lien de contrôle sans type (ou plutôt, dont le type est réduit à une simple orientation), émergeant d'un port auxiliaire des sorties de boîte et incident au port principal des entrées de boîtes.



Les messages suivants vont circuler sur le canal de contrôle d'une boîte, celui-ci reliera son unique sortie à ses multiples entrées.



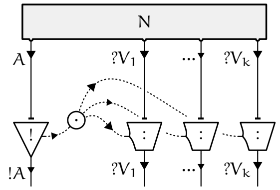
Afin de transmettre ces messages à plusieurs entrées, nous aurons également besoin de nœuds de *diffusion*. Ce sont simplement des opérateurs multiplexants d'arités 0 et 2 pour les trois messages que nous venons d'introduire. Une version d'arité variable peut être obtenue en combinant ces deux opérateurs. On ne s'intéresse pas aux arborescences utilisées pour cela, celles-ci seront toutes représentées de la façon suivante (leur port principal est connecté à l'arête incidente) :



Nous n'introduisons pour l'instant pas de message $\bar{\iota}$ car une *promotion* ne réagit pas devant l'entrée d'une autre boîte (matérialisée par un port auxiliaire). Précisons de nouveau que nous restons dans le formalisme standard des réseaux d'interaction, chaque nœud possède un unique port principal.

3.3.2 Construction

Promotion$_p$ On remplace maintenant les boîtes exponentielles par la construction suivante :

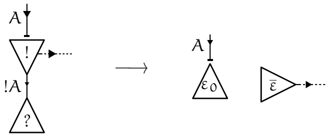


On utilise des nœuds de diffusion pour connecter la sortie d'une construction exponentielle à toutes ses entrées. Ils constituent le canal de contrôle de cette boîte localisée.

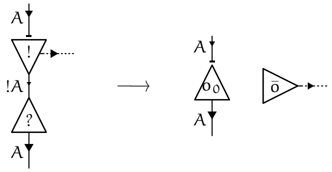
3.3.3 Réductions d'amorçage

Une première série de réductions des opérateurs passifs s'obtient facilement à partir de leur correspondantes actives vues en 3.2.2. En ce qui concerne le devenir du contenu de la boîte, elles commencent la même tâche en lançant un opérateur de boîte sur le lien attaché à la sortie. La différence réside dans le fait que, grâce à l'envoi du message correspondant sur le canal de contrôle, elles informent également les entrées de boîte de l'action à effectuer.

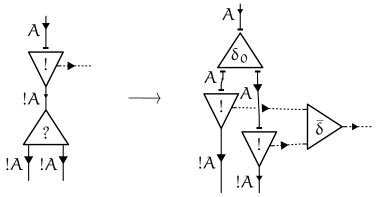
Affaiblissement – Sortie-Promotion$_p$ On commence à effacer la boîte avec un effacement ε_0, et on en avertit les entrées à l'aide d'un message $\bar{\varepsilon}$.



Déréliction — ***Sortie-Promotion***$_p$ On commence à ouvrir la boîte avec une ouverture o_0, et on en avertit les entrées à l'aide d'un message $\bar{o}$.



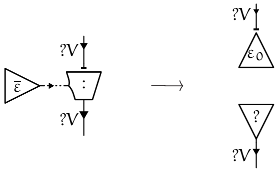
Contraction — ***Sortie-Promotion***$_p$ On commence à dupliquer la boîte avec une duplication δ_0, et on en avertit les entrées à l'aide d'un message $\bar{\delta}$.



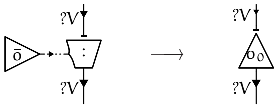
3.3.4 Réductions annexes

Il reste maintenant à spécifier ce qu'il advient lorsqu'un message lancé sur un canal de contrôle arrive sur une entrée. Le fonctionnement est intuitif, connaissant celui du système avec boîtes.

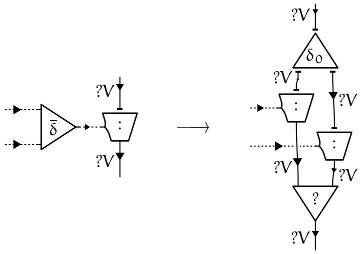
Entrée-Promotion$_p$ — ***Message***$\bar{\varepsilon}$ On efface le contenu de la boîte avec un opérateur ε_0 et on lance un *affaiblissement* à l'entrée de la boîte.



Entrée-Promotion$_p$ — ***Message***$\bar{o}$ On ouvre le contenu de la boîte avec un opérateur o_0 après avoir fait disparaître l'entrée de boîte.



Entrée-Promotion$_p$ — ***Message***$\bar{\delta}$ On duplique le contenu de la boîte avec un opérateur δ_0 et on lance une *contraction* à l'entrée de la boîte.



Remarque 3-6. *Lorsqu'on s'est intéressés à l'étape intermédiaire de duplication d'une boîte (décrite en 3.2.6) que permet d'atteindre la réduction du système actif, on a remarqué que la duplication de l'arbre* T_2 *est mise en attente : on attend qu'une autre boîte se présente face à l'entrée de boîte sur laquelle est branchée le port principal de* T_2*. Dans le système passif, après l'arrivée d'un message* $\bar{\delta}$*, on duplique au contraire l'entrée de boîte et l'arbre* T_2 *pour achever l'opération amorcée. Les interactions qu'auraient pu avoir cette entrée avec une autre boîte sont mise en attente. Mais cette nouvelle façon de procéder duplique de futurs rédex, ceux qui vont apparaître pendant l'interaction entre* T_2 *et le contenu de l'autre boîte. Cette duplication du calcul n'est bénigne que pour la réduction parallèle (toutes les instances d'un rédex répliqué seront de toute façon réduites simultanément). En revanche on perd l'optimalité de la réduction séquentielle à cause de la dernière règle que nous venons de présenter.*

3.3.5 Récapitulatif pour le système localisé passif

Pour ce qui concerne la réduction des opérateurs de boîtes, on reprend celles du système actif données en 3.2.4. Il n'y a pas de mal à les considérer toutes, même si la gamme de rédex qui n'apparaissent jamais s'élargit (voir 3.3.6). Avec celles-ci, nous avons présenté tous les opérateurs et toutes les réductions de notre deuxième système localisé.

Définition 3-6. *On nomme* système exponentiel localisé passif *le système de réseaux d'interaction (très souvent utilisé conjointement avec système multiplicatif vu en 2.2) qui est constitué par :*

- *les opérateurs affaiblissement, contraction et déréliction*
- *les opérateurs entrées et sorties de promotion passives*
- *les opérateurs de diffusion*
- *les messages d'effacement de duplication et d'ouverture*
- *les opérateurs de boîtes d'altitude* $i \in \mathbb{N}$ *: effacement, duplication et ouverture*

et muni des réductions :

- *qui amorcent les opérations à effectuer sur les boîtes, données en 3.3.3*
- *de multiplexage des opérateurs de diffusion, qui permettent de transmettre les messages aux différentes entrées d'une boîte*
- *annexes qui interprètent les messages lorsque ceux-ci atteignent les entrées de boîtes, données en 3.3.4*
- *de multiplexage à niveau des opérateurs de boîtes, données en 3.2.4, qui propagent les opérations effectuées sur une boîte dans leur structure interne (les opérateurs* ι_i *ne sont pas utilisés)*

3.3.6 Propriétés

Le système passif présenté s'inscrivant (comme le système actif) dans le cadre des réseaux d'interaction, une confluence forte de la réduction est automatiquement assurée. Aussi, la nouvelle façon de traduire les boîtes exponentielles ne nous fait pas perdre la propriété 3-5 : le multiplexage à niveau pour réseaux valides.

Simulation

Un théorème de simulation peut être donné, il permet de simuler le système avec boîte avec ce système passif. Ce n'est en fait possible que pour la réduction interne sans commutations $\xrightarrow{\dagger\star}$ que nous avons définie en 3.1.5. Dorénavant, on appellera *preuves*, et on notera Π les réseaux du système avec boîtes. Cela permettra de mieux les distinguer des réseaux notés N des systèmes localisés.

Théorème 3-7. *Le codage* $\langle \mapsto_p \rangle$ *des preuves de la logique linéaire dans les réseaux d'interaction exponentiels passifs simule faiblement la réduction interne sans commutations* $\xrightarrow{\dagger\star}$ *à l'aide de la réduction des réseaux. Cela s'écrit :*

$$\xleftarrow{\dagger\star}\mapsto_p \;\subseteq\; \mapsto_p \twoheadleftarrow$$

ou bien en nommant les éléments, que pour toute preuve Π *:*

$$\begin{cases}\Pi \mapsto_p N\\ \Pi \xrightarrow{\dagger\star} \Pi'\end{cases} \Rightarrow \exists N' \begin{cases}\Pi' \mapsto_p N'\\ N \twoheadrightarrow N'\end{cases}$$

Preuve. On procède par induction sur Π. On ne s'intéresse qu'aux coupures exponentielles car les autres se passent de façon similaire dans les deux formalismes. De plus, étant donné que nous ne considérons pas les commutations exponentielles, il n'y a que trois règles de réduction à simuler. Lorsqu'une *promotion* est coupée avec une *contraction* en logique linéaire, celle-ci est immédiatement dupliquée, mais on suppose par ailleurs que le contenu de la promotion est en forme normale puisqu'on considère seulement la réduction interne. Dans le formalisme localisé passif la première étape de réduction ne duplique que la sortie de la boîte correspondante, elle lance un message $\bar{\delta}$ sur son canal de contrôle et un opérateur δ_0 sur son lien de tête. Cependant, en poursuivant immédiatement la réduction, le message $\bar{\delta}$ va dupliquer chaque entrée de boîte en plaçant également des opérateurs δ_0 face aux autres liens. On utilise alors la propriété de multiplexage à niveau (l'intérieur de la boîte est un réseau valide et en forme normale) qui nous dit qu'en poursuivant la réduction des opérateurs δ_0 la boîte se retrouve entièrement dupliquée. On a pris l'exemple d'une coupure impliquant une *contraction*, cela se passe de façon similaire lorsqu'il s'agit d'un *affaiblissement* ou d'une *déréliction*. □

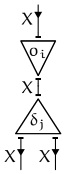
On ne simule que la réduction interne $\xrightarrow{\dagger\star}$. Cela est dû au fait que le fonctionnement des systèmes localisés ne permet pas la duplication ou l'effacement de réseaux qui ne sont pas préalablement normalisés (et cela est généralement un avantage en termes de complexité). On pourra juger ceci suffisant puisqu'on obtient un corollaire pour la normalisation associée $\downarrow_{\dagger\star} = \downarrow_{\dagger}$.

Corollaire 3-8. *Le codage* $\langle \mapsto_{\mathrm{p}} \rangle$ *permet la normalisation (pour la réduction sans commutations) d'une preuve* Π *à l'aide de la réduction du système localisé passif : si* $\Pi \mapsto_{\mathrm{p}} \mathsf{N}$ *alors il existe un unique réseau* N' *en forme normale tel que* $\Pi{\downarrow_{\dagger}} \mapsto_{\mathrm{p}} \mathsf{N}'$ *et* $\mathsf{N} \twoheadrightarrow \mathsf{N}'$.

> **Preuve.** En effet, on remarque que si $\Pi \mapsto_{\mathrm{p}} \mathsf{N}$, Π est en forme normale si et seulement si N est en forme normale. Ainsi, par simulation d'une réduction $\Pi \xrightarrow{\dagger\star\;}\!\!\!\!\!\rightarrow \Pi{\downarrow_{\dagger}}$ on obtient un réduit N' de N en forme normale. De plus, étant donné que N vit dans un système qui vérifie une propriété de confluence forte, sa normalisation faible entraîne sa normalisation forte, c'est-à-dire l'unicité de N'. □

Intégrité

L'absence de réduction avec l'extérieur au niveau des entrées de boîtes empêche que des opérateurs d'altitude strictement positive parcourent les boîtes dans le sens qui va de leurs entrées vers leurs sorties. Seuls des opérateurs d'altitudes nulles, créés lors de l'arrivée d'un message sur les entrées d'une boîte, vont parcourir celles-ci dans cette direction. Ils vont rencontrer les opérateurs de boîtes issus de la réduction d'amorçage correspondante. En particulier, la configuration suivante n'est jamais rencontrée (cette fois, pour tout i et tout j) lors de la réduction d'un réseau valide :



On va montrer cela grâce au théorème de simulation et à la propriété de normalisation des preuves de la logique linéaire (c'est un lourd attirail, et nous aimerions avoir une preuve directe). Dans le cas passif, deux opérateurs de boîtes ne peuvent se rencontrer que s'ils sont identiques. C'est une propriété plus forte que celle annoncée pour le système actif, dans lequel deux opérateurs différents peuvent se rencontrer, à moins qu'ils n'aient la même altitude.

Propriété 3-9. *Partant d'un réseau valide, dans le système passif, la réduction ne fait apparaître aucun rédex liant deux opérateurs de boîtes différents.*

> **Preuve.** Supposons qu'on ait défini le système passif en bloquant les rédex concernés. On remarque que la simulation donnée par le théorème 3-7 est faite sans utiliser ces réductions, il reste donc valide. Partons du traduit N d'une preuve Π, et supposons l'existence d'un réseau N' tel que $\mathsf{N} \twoheadrightarrow \mathsf{N}'$ dans lequel est apparu un rédex bloqué. Par simulation de la normalisation de Π on obtient un réseau N_0 en forme normale tel que $\mathsf{N} \twoheadrightarrow \mathsf{N}_0$. Celui-ci étant en forme normale, par confluence locale de la réduction (elle est même forte ici), on peut faire confluer les deux réductions de N : on a $\mathsf{N}' \twoheadrightarrow \mathsf{N}_0$. On arrive à une contradiction, car un rédex bloqué reste inchangé par réduction, et un réseau qui en contient un ne peut donc être réduit à une (vraie) forme normale. □

Cette propriété fait du paradigme passif un cas un peu particulier : on peut oublier les altitudes sans que cela ne pose de problème. Il peut donc être implanté avec un nombre fini d'opérateurs.

Par ailleurs, un raisonnement identique à celui utilisé pour prouver cette dernière propriété nous permet de dire que partant d'un qu'un réseau valide la réduction ne fait apparaître aucun interblocage (même si on sort de la stratégie de simulation).

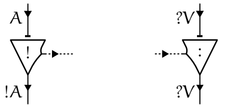
3.4 Le système exponentiel localisé contrôlé

Nous allons ici présenter une ultime version localisée de l'exponentielle. Elle combine les deux localisations précédentes, qui apparaissent comme des sous systèmes. Elle permet de réduire entièrement les réseaux de preuve sans problèmes de relecture, et nous pensons que cette version possède une certaine propriété d'optimalité pour la réduction parallèle. Elle nécessite cependant l'utilisation de bi-nœuds (deux ports principaux). Cette utilisation semble bien évidement légitimée par le fait que les boîtes du système originel sont elles-mêmes des multi-nœuds.

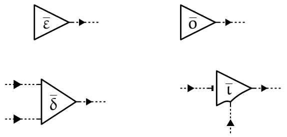
On notera aussi qu'il est possible dans certains cas qu'on gagne en performance vis-à-vis du système actif, car on peut éviter de laisser certaines structures de contrôle inutiles (cf. parties gauches des équivalences opérationnelles aux limites de boîte dans le système actif) qui peuvent modifier la complexité de certaines réductions, comme l'ont découvert *Andrea Asperti* et *Juliusz Chroboczek* dans [AC97].

3.4.1 Opérateurs exponentiels contrôlés

Sortie-Promotion$_{\mathrm{c}}$**,** ***Entrée-Promotion***$_{\mathrm{c}}$ Les opérateurs exponentiels contrôlés sont une nouvelle version des opérateurs précédents. Ils possèdent toujours un canal de contrôle, mais cette fois les sorties de boîtes et les entrées sont des bi-nœuds. Elles ont un premier port principal rattaché au canal de contrôle, et un second port principal dirigé vers l'interface externe de la *promotion* :



Les messages suivants vont passer sur le canal de contrôle d'une boîte, toujours depuis sa sortie et vers ses entrées.

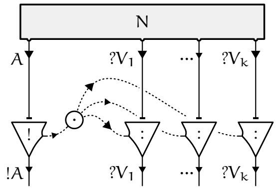


L'introduction d'un nouveau message $\bar{\iota}$ est nécessaire car à présent l'interaction entre deux boîtes est possible. C'est un opérateur à niveau incrémenté. Il possède un second port principal qui est rattaché au canal de contrôle la boîte qui ingère celle dont le canal est parcouru par cet opérateur.

Enfin, afin de transmettre ces messages aux différentes entrées, nous aurons de nouveau recourt à des nœuds de diffusion. Ceux-ci ne réagissent pas avec les nœuds matérialisant l'interface des boîtes, ni entre eux. Ils interagissent seulement avec les messages ou les opérateurs de boîtes, qu'ils transmettent en provenance de tout lien et vers chacun des autres liens. Tous les ports d'un nœud de diffusion doivent par conséquent être considérés principaux, y compris pour des agrégats d'arité arbitraire.

3.4.2 Construction

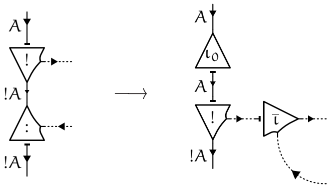
Promotion$_{\mathrm{c}}$ On remplace les boîtes exponentielles par la construction suivante :



3.4.3 Réductions d'amorçage

On reprend les réductions initiales du cas passif (voir 3.3.3) et on les adapte facilement au cas contrôlé. On remplace simplement les nœuds passifs par des nœuds contrôlés. Seule une règle doit être ajoutée qui concerne les commutations exponentielles. Elle est donnée ici.

Entrée-Promotion$_{\mathrm{c}}$ **—** ***Sortie-Promotion***$_{\mathrm{c}}$ On commence à ingérer la boîte avec une *ingestion* ι_0, et on en avertit les entrées à l'aide d'un message $\bar{\iota}$.

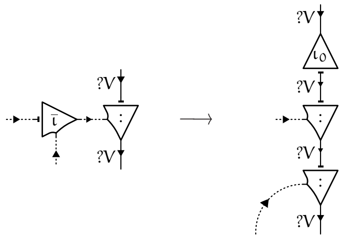


On remarquera avec cette règle que les paramètres d'un message $\bar{\iota}$ sont utilisés de façon totalement dissymétrique. L'idée est de transmettre un lien de contrôle de la boîte qui ingère en le faisant glisser le long du canal de contrôle de la boîte ingérée.

3.4.4 Réductions annexes

De la même façon que pour les réductions d'amorçage, on adapte naturellement les réductions annexes du cas passif (voir 3.3.4) au cas contrôlé. Une nouvelle règle est nécessaire.

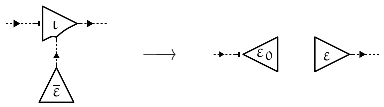
Entrée-Promotion$_{\mathrm{c}}$ **—** ***Message*** $\bar{\iota}$ On ingère le contenu de la boîte avec un opérateur ι_0, et on matérialise l'entrée de la boîte englobante.



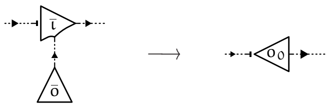
Au niveau des messages d'ingestion

Lorsqu'une boîte qui en ingère une autre subit elle même une opération, on peut mettre en place des réductions qui font intervenir le message correspondant à cette opération avec certains messages $\bar{\iota}$ lancés pour l'opération d'ingestion. Cela arrive lorsque ceux-ci n'ont pas encore rencontré les entrées de boîtes qu'ils devaient atteindre.

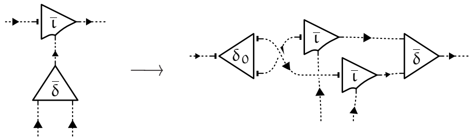
Message $\bar{\varepsilon}$ — Message $\bar{\iota}$



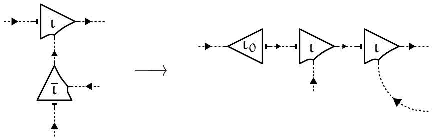
Message $\bar{o}$ — Message $\bar{\iota}$



Message $\bar{\delta}$ — Message $\bar{\iota}$



Message $\bar{\iota}$ — Message $\bar{\iota}$



3.4.5 Récapitulatif pour le système localisé contrôlé

Nous avons présenté tous les opérateurs et toutes les réductions de notre troisième système localisé.

Définition 3-10. *On nomme* système exponentiel localisé contrôlé *le système de réseaux d'interaction (très souvent utilisé conjointement avec système multiplicatif vu en 2.2) qui est constitué par :*

- *les opérateurs affaiblissement, contraction et déréliction*
- *les opérateurs entrées et sorties de promotion contrôlés*
- *les opérateurs de diffusion contrôlés*
- *les messages d'effacement de duplication, d'ouverture et d'ingestion*
- *les opérateurs de boîtes d'altitude $i \in \mathbb{N}$: effacement, duplication, ouverture et ingestion*

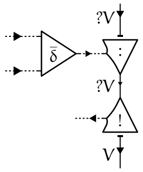
et muni des réductions :

- *qui amorcent les opérations à effectuer sur les boîtes, adaptées de celles données pour le cas passif en 3.3.3, et complétées par celle donnée en 3.4.3*
- *de multiplexage des opérateurs de diffusion, qui permettent de transmettre les messages aux différentes entrées d'une boîte*
- *annexes qui interprètent les messages lorsque ceux-ci atteignent les entrées de boîtes ou les messages d'ingestion, adaptées de celles données pour le cas passif en 3.3.4, et complétées par celles données en 3.4.4*
- *de multiplexage à niveau des opérateurs de boîtes, données en 3.2.4, qui propagent les opérations effectuées sur une boîte dans leur structure interne*

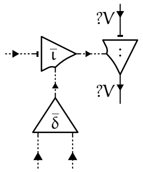
3.4.6 Confluence

Le cas contrôlé qui utilise des bi-nœuds nécessite quelques vérifications supplémentaires lors de l'arrivée simultanée de :

i. un message ($\bar{\varepsilon}$, $\bar{o}$, $\bar{\delta}$ ou $\bar{\iota}$) sur le port de contrôle d'une entrée de boîte, et la tête d'une autre boîte face à l'autre port principal de cette même entrée :

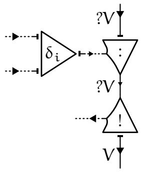


ii. un message ($\bar{\varepsilon}$, $\bar{o}$, $\bar{\delta}$ ou $\bar{\iota}$) sur le canal de contrôle d'un message $\bar{\iota}$ qui atteint en même temps son entrée de boîte :

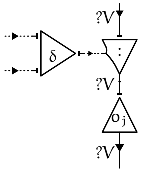


Il y a aussi tous les cas où un (ou deux) opérateurs multiplexants se présentent sur les ports principaux d'un bi-nœud comme par exemple :

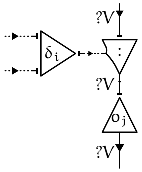
- un opérateur de boîte (ε_i, o_i, δ_i ou ι_i) sur le canal de contrôle d'une entrée de boîte, et la tête d'une autre boîte face à l'autre port principal :



- ou un message sur le canal de contrôle d'une entrée de boîte, et un opérateur de boîte face à l'autre port principal :



- ou un opérateur de boîte sur le canal de contrôle d'une entrée de boîte, et un autre opérateur de boîte face à l'autre port principal :

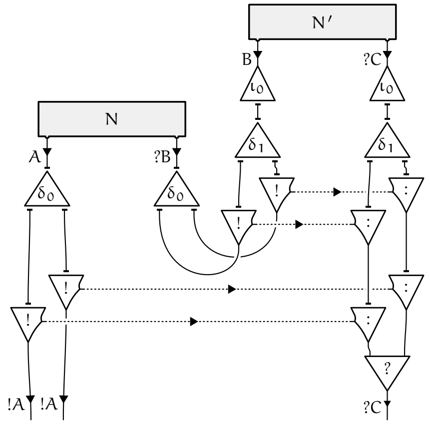


Ces phénomènes apparaissent au voisinage de tous les bi-nœuds, c'est-à-dire les entres et sorties de *promotion*, les messages d'*ingestion* et les nœuds de diffusion. Mais ce sont les seules paires critiques qui restent.

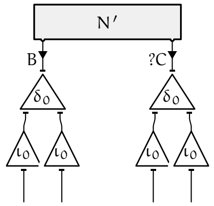
Théorème 3-11. *Le système de réduction des réseaux contrôlés est localement confluent modulo commutation des opérateurs de boîtes.*

Preuve. Les paires critiques évoqués plus haut convergent modulo commutation des opérateurs de boîtes. Excepté (i.) et (ii.) les paires critiques évoquées se résolvent trivialement car au moins l'un des deux rédex se réduit par simple multiplexage. L'autre rédex sera répliqué soit avant soit après réduction et donnera le même réseau. Ceci à l'ordre près des opérateurs multiplexants seulement, c'est pour cela qu'il est nécessaire de travailler modulo les équivalences vérifiées par les opérateurs de boîtes. Les cas les plus compliqués sont les quatre sous cas de (i.) et de (ii.), chacun nécessite plusieurs étapes de réduction et utilise également les équivalences vérifiées par les opérateurs de boîtes. □

Dans le cas de réseaux issus de constructions valides, nous pourrions presque parler de convergence sans modulo. La paire critique (i.) provient vraisemblablement de la rencontre de deux représentations complètes de boîtes et elle atteint un point de quasi-convergence en quelques étapes. La figure donnée ici représente ce point, lorsqu'un message $\bar{\delta}$ a interagi en premier avec l'entrée de boîte.



Si l'autre réduction a lieu avant, on obtient un réseau similaire, seul l'ordre des nœuds δ et ι présents face au réseau N' se trouve inversé :



La convergence exacte s'obtient après réduction complète du réseau N', et jonction des nœuds δ et ι correspondants. Ce phénomène qui retarde la confluence de la réduction n'existe pas dans les réseaux avec boîtes car la duplication et l'ingestion des boîtes se fait entièrement en une étape. En particulier, lorsqu'on travaille avec de vraies boîtes, ces manipulations peuvent être effectuées sur des réseaux qui ne sont pas encore normalisés.

Théorème 3-12. *Si la réduction $\longrightarrow$ des réseaux contrôlés est fortement normalisante modulo $\Vdash\!\dashv$ (ce qui signifie que $\Vdash\!\dashv\longrightarrow\Vdash\!\dashv$ est fortement normalisante) alors elle est Church–Rosser modulo $\Vdash\!\dashv$. Ce qui veut dire :*

$$(\longleftarrow \cup \vdash\!\!\dashv \cup \longrightarrow)^* \;\subseteq\; \twoheadrightarrow\Vdash\!\dashv\twoheadleftarrow$$

Preuve. C'est une application directe d'un théorème de *Gérard Huet* [Hue80], qui donne ce résultat dès que $\longrightarrow$ est localement cohérente avec $\vdash\!\!\dashv$, et localement confluente modulo $\Vdash\!\dashv$. □

3.4.7 Simulation

La localisation contrôlée des réseaux de preuve fournit un moyen de simuler précisément leur exécution :

Théorème 3-13. *Le codage $\langle\ \mapsto_{\mathsf{c}}\ \rangle$ des preuves de la logique linéaire dans les réseaux contrôlés simule faiblement la réduction interne $\xrightarrow{\star}$ à l'aide de la réduction des réseaux, c'est-à-dire :*

$$\xleftarrow{\star}\mapsto_{\mathsf{c}} \;\subseteq\; \mapsto_{\mathsf{c}}\twoheadleftarrow$$

ou de façon équivalente, que pour toute preuve Π :

$$\begin{cases}\Pi \mapsto_{\mathsf{c}} \mathsf{N} \\ \Pi \xrightarrow{\star} \Pi'\end{cases} \Rightarrow \quad \exists \mathsf{N}' \begin{cases}\Pi' \mapsto_{\mathsf{c}} \mathsf{N}' \\ \mathsf{N} \twoheadrightarrow \mathsf{N}'\end{cases}$$

Preuve. On peut considérer que la réduction du système passif est une sous réduction du système contrôlé. On sait déjà par conséquent que toutes les règles de réductions de la logique linéaire, à l'exception de la règle de commutation exponentielle, sont simulées dans ce système. La règle de commutation exponentielle qui existe dans ce nouveau système ne pose en fait pas de difficulté particulière en ce qui concerne la simulation, elle fonctionne comme les autres opérations sur les boîtes, il suffit d'achever la réduction des messages $\bar{\iota}$ et des opérateurs ι_0 qui apparaissent tout comme il suffisait précédemment d'achever la réduction des autres messages et opérateurs de boîte. □

Une fois de plus on ne simule que la réduction interne $\xrightarrow{\star}$, mais cette fois en gardant les commutations exponentielles. Or, étant donné que cette traduction respecte les formes normales, on en déduit une façon d'effectuer une normalisation complète $\downarrow = \downarrow_\star$ à l'aide d'une réduction localisée.

Corollaire 3-14. *Le codage $\langle\ \mapsto_{\mathsf{c}}\ \rangle$ permet la normalisation d'une preuve Π à l'aide de la réduction du système localisé contrôlé : si $\Pi \mapsto_{\mathsf{c}} \mathsf{N}$ alors il existe un réseau N' en forme normale tel que $\Pi\!\downarrow\, \mapsto_{\mathsf{c}} \mathsf{N}'$ et $\mathsf{N} \twoheadrightarrow \mathsf{N}'$.*

Preuve. Une fois de plus, lorsque $\Pi \mapsto_{\mathsf{c}} \mathsf{N}$, la preuve Π est en forme normale si et seulement si N est en forme normale. Par simulation d'une réduction $\Pi \xrightarrow{\star}\!\!\!\!\twoheadrightarrow \Pi\!\downarrow$ on obtient un réduit N' de N en forme normale. □

Remarquons que nous n'avons pas besoin d'utiliser un multiplexage à niveaux très complexe pour montrer ces propriétés. Un équivalent de la propriété 3-5 suffit. On fait disparaître tous opérateurs de boîte d'altitude 0 utilisés pour effectuer une opération de boîte (ainsi que les opérateurs d'altitude supérieure issus de leurs réductions) avant d'entamer une nouvelle réduction logique. Rien n'est très compliqué lorsqu'on respecte cette discipline. C'est la propriété de confluence qui nous dit que tout se passe bien si on s'autorise une réduction plus libre.

3.4.8 Intégrité

Pour montrer que les rédex qui n'ont pas été pourvus de règles de réductions n'apparaissent jamais, nous nous appuierons une fois de plus sur la propriété de normalisation des preuves de la logique linéaire.

Propriété 3-15. *Partant d'un réseau valide, dans le système contrôlé, la réduction ne fait apparaître aucun rédex liant deux opérateurs de boîtes différents de même altitude.*

Preuve. Même méthode que dans le cas passif. Supposons qu'on ait défini le système contrôlé en bloquant les rédex concernés. On remarque que la simulation donnée par le théorème 3-13 est faite sans utiliser ces réductions, il reste donc valide. Partons du traduit N d'une preuve Π, et supposons l'existence d'un réseau N' tel que $\mathsf{N} \twoheadrightarrow \mathsf{N}'$ dans lequel est apparu un rédex bloqué. Par simulation de la normalisation de Π on obtient un réseau N_0 en forme normale tel que $\mathsf{N} \twoheadrightarrow \mathsf{N}_0$. Celui-ci étant en forme normale, par confluence locale modulo de la réduction, on peut faire confluer les deux réductions de N : on a $\mathsf{N}'\ (\longrightarrow \cup \vdash\!\!\dashv)^*\ \mathsf{N}_0$. On arrive à une contradiction car un rédex bloqué reste inchangé par réduction et laisse de nouveaux rédex bloqués par modulo (c'est ce modulo qui nous empêche de prouver la même propriété que dans le cas passif, propriété qui n'est pas vérifiée ici). □

Notons que la réduction du système exponentiel localisé actif peut être vue comme une sous réduction du système exponentiel localisé contrôlé. En effet système actif est identique au système contrôlé dans lequel on a rendu auxiliaires les second ports principaux (connectés au canal de contrôle) des entrées de boîtes. On obtient donc un corollaire pour le système actif.

Corollaire 3-16. *Partant d'un réseau valide, dans le système actif, la réduction ne fait apparaître aucun rédex liant deux opérateurs de boîtes différents de même altitude.*

3.5 Synthèse

Notons en premier lieu que par principe toutes réductions considérées préservent les types, et que par conséquent, la « subject reduction » est une propriété évidente des systèmes de réduction que nous avons été amenés à étudier.

La traduction des boîtes exponentielles dans chaque système localisé que nous avons présenté est directe et se fait sans considérations d'altitude. En fait, les opérateurs de boîtes, seuls nœuds qu'il est nécessaire de paramétrer par une altitude, n'apparaîtront que lors de leur réduction localisée.

Formulons ensuite un récapitulatif succinct des caractéristiques de nos systèmes (il contient une entrée *actif+* dont la présence est expliquée après) :

Voir Fig. 3.1 (page 47).

Nous avons vu que les codages passif $\langle\ \mapsto_{\mathsf{p}}\ \rangle$, et contrôlé $\langle\ \mapsto_{\mathsf{c}}\ \rangle$ évoqués lors de la construction des réseaux correspondants possèdent des propriétés intéressantes de simulation des systèmes de réduction usuels.

Système actif avec relecture

En ce qui concerne le système actif, on a déjà indiqué qu'il est identique au système contrôlé dans lequel on a inactivé les interactions entre messages et entrées de boîtes. Le canal de contrôle ne joue alors plus aucun rôle dans la réduction, mais sa présence peut être mise à contribution. En effet, lorsqu'on a terminé la réduction on tombe typiquement dans un état où certaines opérations sur les boîtes n'ont pas été achevées. Mais comment savoir à quelle preuve de la logique linéaire cet état correspond ? C'est d'ailleurs ce problème qui nous empêche de formuler un théorème de simulation pour le cas actif, et qui est désigné sous l'appellation problème de « relecture ». Si on a astucieusement gardé les canaux de contrôle (même rendus inactifs) on a la possibilité future de les réactiver pour procéder à la « relecture » et terminer la réduction.

C'est ce système, qui s'appuie sur la syntaxe des réseaux contrôlés, où on épuise les réductions du système actif avant de les compléter par quelques réductions du système passif, qu'on a appelé *actif+*.

système	cardinalité	com. exp.	relecture	nœuds	optimalité ?
actif	infini	oui	non	normaux	séquentielle
actif+	infini	oui	oui	normaux	séquentielle
passif	fini	non	oui	normaux	-
contrôlé	infini	oui	oui	multiples	parallèle

Figure 3.1

Chapitre 4

Réseaux additifs

Les réseaux additifs vont permettre de modéliser les choix (ou les absences de possibilités). Ils permettent d'interpréter des programmes dans lesquels une donnée peut être construite à l'aide de constructeurs différents, comme dans l'exemple suivant :

```
type shape =
  | Point
  | Disk of int
  | Rectangle of int * int

let area s =
  match s with
    | Point ->
        0
    | Disk (radius) ->
        pi * radius * radius
    | Rectangle (width, height) ->
        width * height

let result =
  area (Disk 2)
```

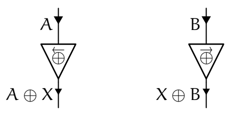
L'utilisation la plus courante de ces types de données est cachée dans les instruction conditionnelles fournies par la plupart des langages de programmation. Un type booléen possède deux constructeurs sans paramètres `true` et `false`, et une instruction de filtrage nommée `if`.

En logique linéaire, deux connecteurs duaux permettent d'exprimer le choix. Pour prouver la formule $A \oplus B$, il suffit de fournir une preuve de A ou une preuve de B à un utilisateur qui devra s'adapter au choix qui lui est proposé. De façon duale, si on prouve une formule $A \& B$ on est capable de fournir une preuve de A ou de B selon l'utilisation qui en est faite. Le but de ce chapitre est de présenter les constructions additives associées à ces connecteurs, nous les utiliserons par la suite. Nous montrerons également que la localisation telle qu'elle a été faite pour les constructions exponentielles s'adapte relativement bien au cas des constructions additives. La localisation passive (qui semble plus raisonnable que les autres dans le cas des constructions additives) fonctionne sans problème. Néanmoins si on souhaite effectuer des commutations additives on se heurte à quelques difficultés, et on se placera pour les contourner dans une logique linéaire légèrement restreinte.

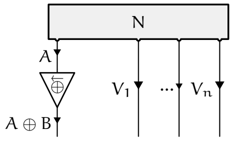
4.1 Réseaux de preuve avec boîtes additives

4.1.1 Constructions additives

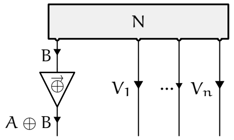
$\overleftarrow{Plus}$, $\overrightarrow{Plus}$ On introduit deux nœuds unaires, nommés *plus gauche* et *plus droit*, que l'on notera ‹ $\overleftarrow{\oplus}$ › et ‹ $\overrightarrow{\oplus}$ ›. Leur typage et leur représentation graphique sont les suivants :



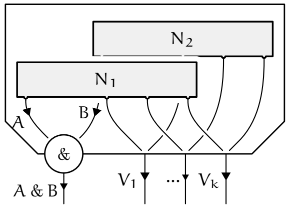
Ces nœuds sont introduits selon les conventions usuelles associées aux nœuds unaires positifs. La première construction correspond à l'utilisation de la règle logique $\frac{\vdash A, \Gamma}{\vdash A \oplus B, \Gamma}$:



La seconde, à la règle $\frac{\vdash B, \Gamma}{\vdash A \oplus B, \Gamma}$:

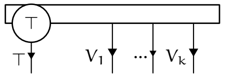


Avec Étant donnés deux réseaux N_1 et N_2 d'interfaces respectives $\vdash A, \Gamma$ et $\vdash B, \Gamma$, la construction *avec* se fait à l'aide d'une boîte (double) selon la règle logique $\frac{\vdash A, \Gamma \quad \vdash B, \Gamma}{\vdash A \,\&\, B, \Gamma}$. On nomme *tranches* les deux composantes N_1 et N_2 d'une telle boîte, elles vivent séparément et sont complètement déconnectées.



Cette construction peut être comparée aux constructions de filtrage par profil de certains langages de programmation.

Top La construction *top* se fait à l'aide d'une boîte (vide) selon la règle $\overline{\vdash \top, \Gamma}$.



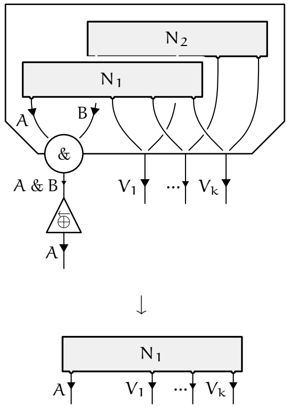
Nul Aucune construction n'existe en logique linéaire pour le dual 0 de la formule $\top$.

4.1.2 Réduction des boîtes additives

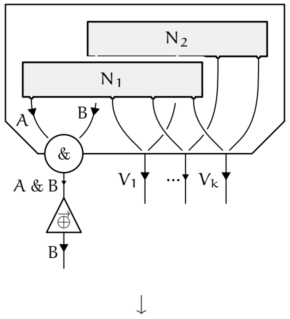
Deux réductions logiques existent pour une boîte *avec*. Aucune pour une boîte *top*. Nous verrons ensuite les réductions de commutation qui reflètent une certaine identité logique entre certains réseaux construits avec ces boîtes.

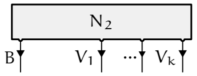
Réductions logiques

Avec **– *$\overleftarrow{\text{Plus}}$*** L'opérateur *plus gauche* va permettre de demander à la boîte additive correspondante de se comporter tel le réseau contenu dans sa partie gauche.



Avec **– *$\overrightarrow{\text{Plus}}$*** La règle pour l'opérateur *plus droit* est similaire, on garde le réseau droit N_2 à la place du réseau N_1 :

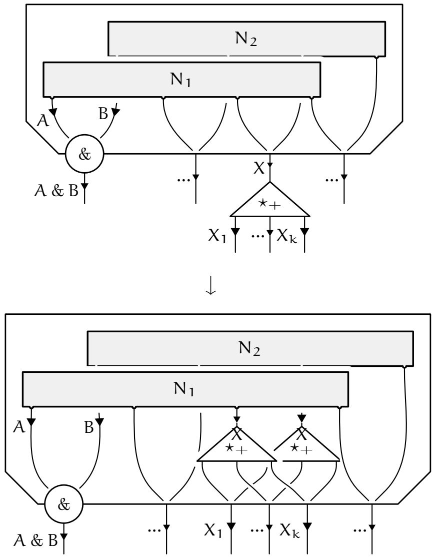




Commutations positives

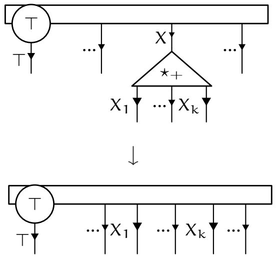
Comme dans le cas des boîtes exponentielles, on peut mettre en place des réductions de commutation lorsque des opérateurs se présentent face aux entrées des boîtes additives. Cela n'est pas réellement nécessaire, particulièrement dans le cas des boîtes additives où cela conduit à une explosion exponentielle de la quantité de mémoire qu'il faut pour représenter certains réseaux. Elles augmentent cependant le degré de parallélisme que l'on peut atteindre si on est prêt à payer le coût en mémoire et en nombre de processeurs utilisés.

$\overline{\textit{Avec}}-\star_+$ Lorsqu'un opérateur logique positif se présente face à l'entrée d'une boîte additive celui-ci rentre dans cette boîte. Dans le cas d'une boîte *avec* celui-ci est ajouté aux deux tranches de la boîte.



On notera que cette règle s'applique également lorsque l'opérateur $\star_+$ est une boîte exponentielle (ou éventuellement additive) connectée par sa sortie, ou éventuellement l'une de ses entrées.

$\overline{\textit{Top}}-\star_+$ Dans le cas d'une boîte *top* tout opérateur positif qui se présente face à une entrée est oublié ainsi :



Comme précédemment, cette règle s'applique lorsque l'opérateur $\star_+$ est une boîte exponentielle. Elle peut également s'appliquer lorsqu'il s'agit d'une boite additive.

Dans le cas où deux boîtes additives sont connectées par leurs entrées, deux des règles précédentes peuvent être appliquées, la réduction n'est alors pas déterministe.

Commutations négatives

La gestion des commutations dans le cas des opérateurs négatifs nécessite une mécanique plus complexe. Il faudrait comme dans l'article original sur les réseaux de preuve de *Jean-Yves Girard* utiliser des boîtes pour toutes les constructions négatives, de manière à pouvoir les faire rentrer (dédoublées) dans les boîtes *avec* en une étape. Cela contredirait l'objectif des réseaux dans lequel on souhaite oublier le maximum de bureaucratie lié à la séquentialisation pour privilégier l'exécution parallèle, et nous n'allons donc pas aller dans ce sens.

Néanmoins, tout se passe bien lorsqu'on restreint le contexte des boîtes additives à des types entrants *extérieurement positifs*. Ces types sont définis de la façon suivante :

$$D \quad ::= \quad X \;\mid\; D \otimes D \;\mid\; D \oplus D \;\mid\; !A$$

où A désigne une formule quelconque de la logique linéaire. Cette restriction est stable par réduction. En particulier, la traduction du langage *PCF* que nous donnerons plus tard et qui utilise ces boîtes s'accommode très bien de cette restriction. C'est principalement du au fait que le type récursif $o^\perp = !o \otimes o^\perp$ qui permet de coder le λ-calcul peut être considéré comme un type extérieurement positif.

En outre, on évite du même coup le non déterminisme des commutations entre deux boîtes additives. Pour parler de systèmes localisés actifs ou contrôlés on se placera dans ce cadre restreint.

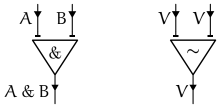
4.2 Le système additif localisé actif

La première construction additive localisée que nous allons considérer est la plus concise (comme dans le cas des boîtes exponentielles), mais elle ne correspond pas à un flot d'exécution vraiment naturel.

Le système d'exécution obtenu correspond à celui qui, pour évaluer l'expression « `if b then x else y` » d'un langage de programmation habituel, évaluerait à la fois b, x et y en poursuivant en entier l'évaluation de x et y en utilisant le contexte, et qui ne déciderait qu'à la fin lequel des deux garder (le choix est fait en fonction du résultat obtenu pour b). Dans les réseaux de preuve, cela revient à effectuer toutes les commutations additives avant la vraie réduction logique des boîtes *avec*.

4.2.1 Opérateurs additifs actifs

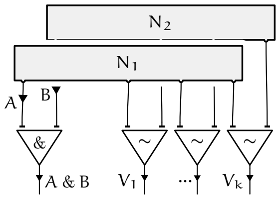
Sortie-Avec$_a$, ***Entrée-Avec***$_a$ Les opérateurs qui codent la sortie et les entrées d'une boîte additive pour la construction *avec* sont binaires. Ce sont tout deux des opérateurs avec *niveau incrémenté*.



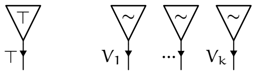
Sortie-Top$_a$, ***Entrée-Top***$_a$ Les opérateurs qui codent la sortie et les entrées d'une boîte additive pour la construction *top* sont nullaires.

4.2.2 Constructions additives localisées actives

Avec$_a$ On remplace les boîtes *avec* par la construction suivante :



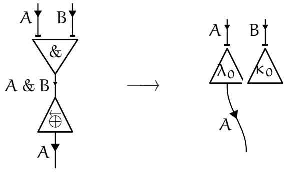
Top$_a$ Les boîtes *top* sont quant à elles remplacées par :



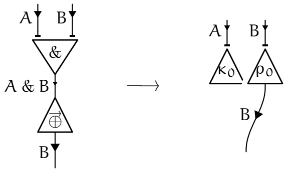
4.2.3 Réduction des opérateurs additifs actifs

Réductions logiques

Sortie-Avec$_a$ $-$ $\overleftarrow{\textit{\textbf{Plus}}}$ Lorsqu'un opérateur *plus gauche* se présente face à une sortie de boîte additive, on effectue un branchement avec le lien principal de la tranche gauche de la boîte et on commence à supprimer la tranche droite de la boîte en lançant un opérateur κ_0 sur le lien principal de la tranche droite.



Sortie-Avec$_a$ $-$ $\overrightarrow{\textit{\textbf{Plus}}}$ De façon similaire, un opérateur *plus droit* choisit la tranche droite de la boîte qu'il rencontre.



Le typage fait qu'aucun nœud ne peut rencontrer la sortie d'une construction *top*, il n'y a donc aucune réduction logique associée.

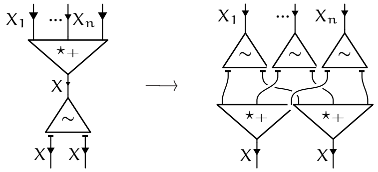
Opérateurs de boîtes

Les réductions des opérateurs de boîtes λ_i, ρ_i, et κ_i sont similaires à celles des opérateurs ε_i et o_i qui ont déjà été rencontrées en 3.2.4. Ils sont multiplexants face aux opérateurs logiques (λ_i et ρ_i sont tout de même des opérateurs à niveau décrémenté comme o_i). Ils ne peuvent rencontrer des opérateurs de leur propre niveau excepté eux-mêmes.

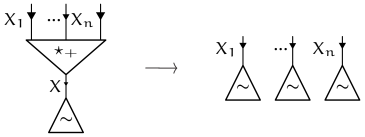
Commutations avec les opérateurs positifs simples

Dans le cadre de la restriction aux types extérieurement positifs évoquée précédemment, les règles de commutation n'interviennent que pour des opérateurs positifs. Dans le cas des opérateurs positifs simples (*unité* et *tenseur*, mais cela pourrait être généralisé plus tard) elle s'exprime localement par :

Entrée-Avec$_a$ $- \star_+$ Pour $\star_+$, opérateur logique positif simple :

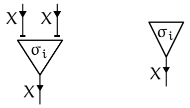


Entrée-Top$_a$ $- \star_+$ Pour $\star_+$, opérateur logique positif simple :



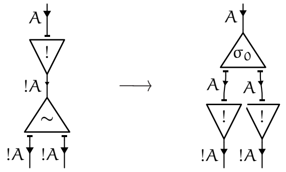
Commutations, le cas des boîtes exponentielles

On espère le même genre de commutation lorsque l'opérateur positif est une boîte exponentielle. Mais dans un modèle localisé on ne peut pas contracter (ou affaiblir) additivement une boîte en une seule étape. Il faut introduire de nouveaux opérateurs de boîtes pour effectuer progressivement cette opération :

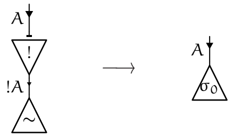


Ils sont multiplexants comme tous les opérateurs de boîtes et sont avec niveau incrémenté pour la même raison que les opérateurs ι_i l'étaient : il faut incrémenter l'altitude de la boîte exponentielle qu'on fait passer à l'intérieur de la boîte additive. À ce détail près les opérateurs σ_i dits de *copie* et d'*oubli* (on les distingue graphiquement par leur arité) se comportent opérationnellement de façon très similaire aux opérateurs de *duplication* δ_i et aux opérateurs d'*effacement* ε_i. Plus tard il sera naturel de distinguer leur sémantique (voir 12.3), et ce sont bien des opérateurs différents. Leur action est déclenchée par les règles que l'on va donner maintenant.

Entrée-Avec$_a$ $-$ ***Sortie-Promotion***$_a$ Un entrée de boîte *avec* face à une sortie de boîte *promotion* lance un opérateur σ_0 qui commence à copier le reste de la boîte exponentielle.



Entrée-Top$_a$ $-$ ***Sortie-Promotion***$_a$ Une entrée de boîte *top* face à une sortie de boîte *promotion* lance un opérateur σ_0 qui commence à oublier le reste de la boîte exponentielle.



Dans le paradigme actif, les opérateurs σ_0 n'émergeront jamais des entrées de la boîte qu'ils copient ou oublient ; ils attendront au contraire que celles-ci aient réagi avec d'autres boîtes. Les deux tranches des boîtes *avec* seront donc calculées entièrement, bien qu'une seule sera utilisée. Ce paradigme est donc assez peu raisonnable pour les constructions additives. Nous préférerons étudier le cas passif qui correspond à une gestion plus standard des constructions conditionnelles.

4.2.4 Récapitulatif pour le système localisé actif

Nous avons présenté tous les opérateurs et toutes les réductions de notre système actif comprenant les constructions additives (avec contexte extérieurement positif).

Définition 4-1. *On nomme* système additif localisé actif *le système de réseaux d'interaction (très souvent utilisé conjointement avec le système multiplicatif vu en 2.2 et le système exponentiel localisé actif vu en 3.2) qui est constitué par :*

- *les opérateurs plus gauche et plus droit*
- *les opérateurs entrées et sorties de boîtes avec actives*
- *les opérateurs entrées et sorties de boîtes top actives*
- *les opérateurs de boîtes additives* λ_i*,* ρ_i *et* κ_i
- *les opérateurs de boîtes exponentielles* σ_i*, de copie et d'oubli, qui viennent s'ajouter aux opérateurs* ε_i*,* δ_i*,* o_i *et* ι_i *lorsqu'on considère aussi les constructions exponentielles.*

et muni des réductions :

- *qui initient les opérations à effectuer sur les boîtes additives, données en 4.2.3*
- *qui effectuent les commutations additives simples, données en 4.2.3*
- *qui effectuent les commutations additives avec les constructions exponentielles lorsqu'on les considère, données en 4.2.3*
- *de multiplexage à niveau des nouveaux opérateurs de boîtes (additives et exponentielles) adaptées du principe qui a été donné en 3.2.4.*

4.3 Le système additif localisé passif

Si on se place dans un langage de programmation habituel, l'évaluation d'une construction additive passive correspond à un système qui, pour calculer l'expression « `if b then x else y` », évalue à la fois b, x et y, mais ne poursuit l'exécution de x ou y dans le contexte que pour l'un de ces deux éléments (choisi en fonction du résultat obtenu pour b, lorsque celui-ci est connu).

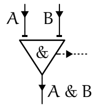
Il est possible de comparer le système auquel on aboutit ici (en se fondant sur le système des boîtes exponentielles évoqué précédemment) au système donné par *Ian Mackie* [Mac05] dans le cadre du λ-calcul ; le principe général de leurs fonctionnements sont identiques.

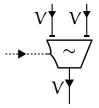
4.3.1 Opérateurs additifs passifs

Comme il a déjà été fait pour les boîtes exponentielles, on obtient un système additif passif en ajoutant un canal de contrôle entre les sorties de boîtes additives et leurs entrées. Les liens qui constituent ces canaux de contrôle sont étiquetés par deux nouveaux types (un pour les boîtes *avec* et un pour les boîtes *top*). Ils sont réduits à une simple orientation, et on ne les distinguera pas graphiquement, ni entre eux, ni du type de contrôle exponentiel.

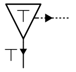
Les entrées de boîtes additives n'interagissant que lorsqu'on leur transmet des messages, le cas passif reste simple.

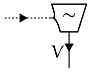
Sortie-Avec$_p$**,** ***Entrée-Avec***$_p$



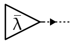


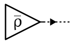
Sortie-Top$_p$**,** ***Entrée-Top***$_p$





On réutilisera les nœuds de diffusion, et on introduit les messages de contrôle additif suivants, qui parcourront les canaux de contrôle des constructions *avec* :

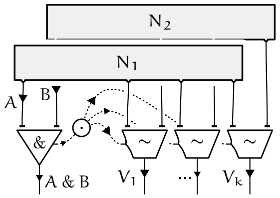




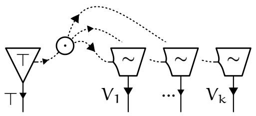
Comme aucune interaction logique n'a lieu avec les constructions *top*, aucun message ne parcourra leurs canaux de contrôle. Ceux-ci permettront néanmoins d'assurer la connexion entre les nœuds lorsque par exemple cette construction doit être dupliquée ou effacée par des opérateurs de boîtes exponentielles.

4.3.2 Constructions additives localisées passives

Avec$_p$ On remplace les boîtes *avec* par la construction suivante :



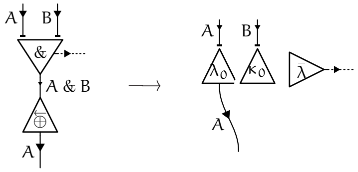
Top$_p$ De façon similaire, les boîtes *top* sont remplacées par :



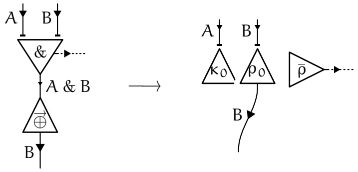
4.3.3 Réduction des opérateurs additifs passifs

Réductions d'amorçage

Sortie-Avec$_p$ **—** $\overleftarrow{\textit{Plus}}$ Comme dans le cas actif, lorsqu'un opérateur *plus gauche* se trouve face à une boîte additive, on choisit la tranche gauche de la boîte. La distinction vient du fait que l'on informe les entrées de la boîte de ce choix en envoyant un message $\bar\lambda$ le long du canal de contrôle.

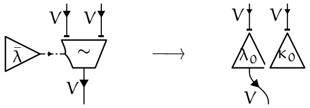


Sortie-Avec$_p$ **—** $\overrightarrow{\textit{Plus}}$ De façon similaire, on utilise un message $\bar\rho$ lorsqu'il s'agit d'un opérateur *plus droit*.

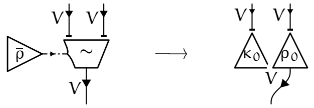


Réductions annexes

Entrée-Avec$_p$ **—** ***Message***$\bar\lambda$ Au niveau des entrées, la réception d'un message $\bar\lambda$ déclenche le même processus de branchement sur la tranche gauche de la boîte et de suppression de la tranche droite de la boîte.



Entrée-Avec$_p$ **—** ***Message***$\bar\rho$ La règle est similaire pour la réception d'un message $\bar\rho$, on branche cette fois sur la tranche droite.



4.3.4 Récapitulatif pour le système localisé passif

Nous avons présenté tous les opérateurs et toutes les réductions de notre système passif comprenant les constructions additives (sans restriction des contextes).

Définition 4-2. *On nomme* système additif localisé passif *le système de réseaux d'interaction (très souvent utilisé conjointement avec le système multiplicatif vu en 2.2 et le système exponentiel localisé passif vu en 3.3) qui est constitué par :*

- *les opérateurs plus gauche et plus droit*
- *les opérateurs entrées et sorties de boîtes avec passives*
- *les opérateurs entrées et sorties de boîtes top passives*
- *les opérateurs de diffusion*
- *les messages* $\bar{\lambda}$ *et* $\bar{\rho}$
- *les opérateurs de boîtes additives* λ_i, ρ_i *et* κ_i

et muni des réductions :

- *qui initient les opérations à effectuer sur les boîtes additives, données en 4.3.3*
- *de multiplexage des opérateurs de diffusion, qui permettent de transmettre les messages aux différentes entrées d'une boîte*
- *annexes qui interprètent les messages lorsque ceux-ci atteignent les entrées de boîtes, données en 4.3.3*
- *de multiplexage à niveau des opérateurs de boîtes additives, adaptées du principe qui a été donné en 3.2.4.*

4.3.5 Simulation

Sans surprise on obtient un théorème de simulation pour les réseaux additifs. Il s'exprime simplement dans le cas passif.

Théorème 4-3. *Le codage* $\langle \mapsto_p \rangle$ *des preuves de la logique linéaire dans les réseaux d'interaction additifs passifs simule faiblement la réduction interne sans commutations additives* $\xleftarrow{\dagger\star}$, *à l'aide de la réduction des réseaux :*

$$\xleftarrow{\dagger\star}\mapsto_p \;\subseteq\; \mapsto_p \twoheadleftarrow$$

Preuve. Raisonnement identique à celui utilisé pour les réseaux exponentiels localisés passifs : on procède par induction sur la preuve Π dont on souhaite simuler une réduction. On ne s'intéresse qu'aux coupures additives car les coupures multiplicatives se passent de façon similaire dans les deux formalismes, et on a déjà vu comment simuler les coupures exponentielles lorsqu'on les considère (les boîtes additives localisées se prêtent sans difficultés à la réplication). Dans le formalisme localisé passif une coupure additive déclenche la sélection d'une tranche : après transmission du message correspondant, entrée et sorties sont branchés sur cette tranche par l'intermédiaire d'un opérateur de boîte λ_0 ou ρ_0 (selon le cas), et chaque lien de l'interface de l'autre tranche est relié à un opérateur de boîte κ_0. En poursuivant la réduction par multiplexage à niveau on fait disparaître les opérateurs de boîtes : la tranche sélectionnée est restituée inchangée et l'autre tranche a disparu. On obtient ainsi un état qui est le traduit du résultat obtenu par la réduction globale des boîtes additives. □

4.4 Le système additif localisé contrôlé

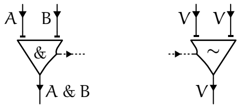
Toujours dans la perspective du calcul d'une expression conditionnelle du type « `if b then x else y` », l'évaluation d'une construction contrôlée correspond à un système qui évalue en parallèle b, x et y, poursuit l'exécution de x ou y dans les parties du contexte déjà évaluées, mais arrête — ou plus exactement, oublie — le calcul de celui qui n'importe pas dès que le résultat pour b est connu.

Nous nous contenterons ici de donner à titre indicatif les opérateurs et les règles de réductions qui manquent pour la former le système localisé contrôlé associé aux constructions additives. Le cas contrôlé est comme le cas actif soumis à la restriction des types extérieurement positifs dans les contextes de boîtes additives.

4.4.1 Opérateurs additifs contrôlés

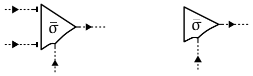
On reprend les opérateurs additifs passifs en remplaçant leurs entrées par des bi-nœuds.

Sortie-Avec$_c$, ***Entrée-Avec***$_c$



Sortie-Top$_c$, ***Entrée-Top***$_c$

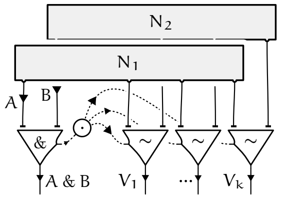
On réutilise bien sûr les messages $\bar{\lambda}$ et $\bar{\rho}$. On aura également besoin de nouveaux messages $\bar{\sigma}$ pour le contrôle additif des boîtes exponentielles :



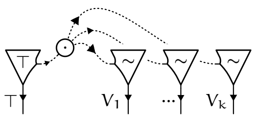
Comme les entrées de boîtes *avec* et les entrées de boîtes *top*, on les distingue par leurs arités. Le canal de contrôle de direction verticale est un canal de boîte additive *avec* alors que les canaux horizontaux sont des canaux de contrôle de boîtes exponentielles.

4.4.2 Constructions additives localisées contrôlées

Avec$_c$ On remplace les boîtes *avec* par la construction suivante :



Top$_c$ Les boîtes *top* sont remplacées par :



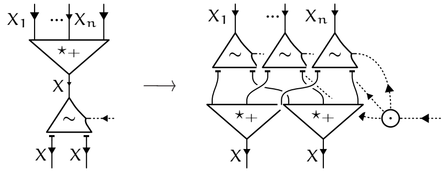
4.4.3 Réduction des opérateurs additifs contrôlés

La première partie des règles de réduction des opérateurs additifs contrôlés est similaire à celles des opérateurs passifs, il suffit de les adapter aux nœuds contrôlés. La nouveauté concerne les commutations additives qui ont lieu par interaction des entrées de boîtes additives. Celles-ci interagissent selon les règles décrites ci-dessous.

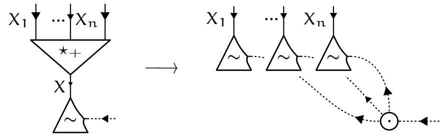
Commutations avec les opérateurs positifs simples

Les interactions avec les opérateurs logiques simples (qui ne sont pas matérialisés par des boîtes) sont données par les méta-règles suivantes lorsque $\star_+$ désigne un opérateur logique positif simple :

Entrée-Avec$_c$$-\star_+$ L'opérateur est copié et placé dans les deux tranches de la boîte additive.



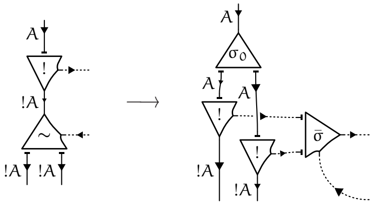
Entrée-Top$_c$$-\star_+$ L'opérateur est oublié.



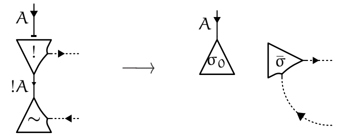
Commutations, le cas des boîtes exponentielles

C'est ici que nous aurons besoin des messages $\bar{\sigma}$, il parcourent les boîtes exponentielles pour informer leurs entrées que la boîte est en train d'être copiée afin d'être déplacée dans les deux tranches d'une boîte *avec*, ou oubliée en rentrant dans une boîte *top* vide.

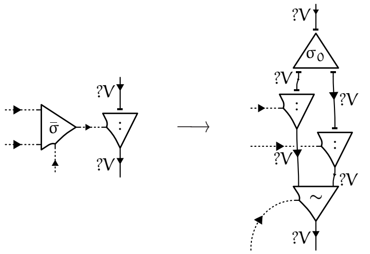
Entrée-Avec$_c$ — ***Sortie-Promotion***$_c$ On amorce une opération de copie.



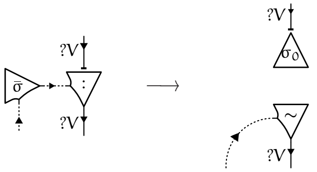
Entrée-Top$_c$ — ***Sortie-Promotion***$_c$ On amorce une opération d'oubli.



Entrée-Promotion$_c$ — ***Message*** $\bar\sigma$ On copie le contenu de la boîte exponentielle avec un opérateur σ_0 et on fait réapparaître une entrée de boîte additive sous les entrées de boîtes exponentielles obtenues.



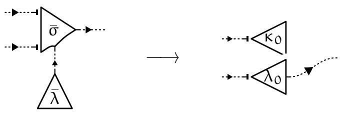
Entrée-Promotion$_c$ — ***Message*** $\bar\sigma$ On oublie le contenu de la boîte exponentielle avec un opérateur ε_0 et on fait réapparaître une entrée de boîte additive.



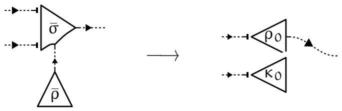
Réductions annexes au niveau des messages

Il ne reste plus que les cas un peu particuliers où des messages peuvent arriver sur d'autres messages, comme $\bar\sigma$ ou $\bar\iota$, qui sont soumis à un contrôle. Ces réductions viennent compléter les quatre réductions de ce type qui existaient déjà lorsqu'il n'y avait que des boîtes exponentielles. Celles-ci ont été données en 3.4.4.

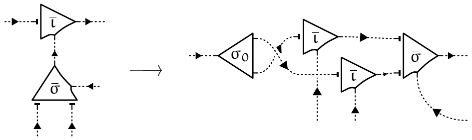
Message $\bar\lambda$ — ***Message*** $\bar\sigma$



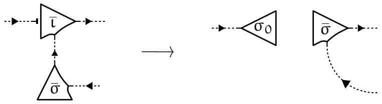
Message $\bar\rho$ — ***Message*** $\bar\rho$



Message $\bar\iota$ — ***Message*** $\bar\sigma$



Message $\bar\iota$ — ***Message*** $\bar\sigma$



Chapitre 5

Réseaux récursifs

L'introduction de la récursion a pour but de rendre possible le codage de systèmes ayant la puissance des machines de *Turing* dans nos réseaux typés. En particulier la terminaison du système de réduction n'est plus assurée dès qu'on l'introduit. Elle nous offre la possibilité d'interpréter des programmes aussi rigolos que celui-ci :

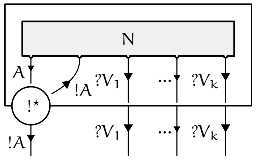
```
let rec syracuse n =
  if n = 1
  then 0
  else 1 +
    if n % 2 = 0
    then syracuse (n / 2)
    else syracuse (3 * n + 1)
```

Nous avons retenu une construction particulière, qui est en réalité une généralisation des boîtes exponentielles, pour remplir le rôle de récurseur. Nous montrerons que grâce à celle-ci il est possible de coder le langage *PCF* [Plo77].

5.1 Boîtes de récursion

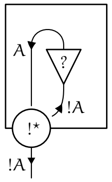
5.1.1 Construction récursive

Récursion Soit N un réseau d'interface $\vdash A, ?A^\perp, ?\Gamma$. On peut construire selon une nouvelle règle logique $\frac{\vdash A, ?A^\perp, ?\Gamma}{\vdash !A, ?\Gamma}$, un réseau d'interface $\vdash !A, ?\Gamma$. On matérialise cette construction à l'aide de la boîte récursive suivante :



Utilisées correctement, ces nouvelles boîtes vont permettre l'écriture de nouveaux programmes qui savent manipuler habilement différentes structures de données récursives telles que des entiers, des listes ou des arbres.

Cependant, on ne manquera pas de constater que la règle que nous venons d'ajouter rend la logique sous-jacente inconsistante. En effet, le réseau suivant :



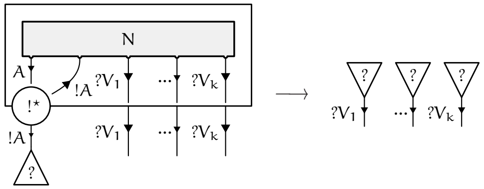
permet de prouver $!A$, et par conséquent A, pour toute formule A. Bien sûr, si on essaye de l'utiliser (en l'interrogeant grâce à une *déréliction*) la réduction que nous allons mettre en place ne terminera pas. Mal utilisée, la construction récursive permet à ce réseau de mentir : contrairement à ce que son interface annonce, il ne sait pas produire des objets de type A.

5.1.2 Réduction des boîtes récursives

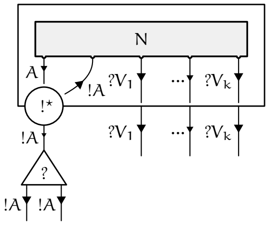
Les boîtes récursives présentent extérieurement la même interface que les boîtes exponentielles, les rédex dont elles peuvent faire partie seront donc de formes similaires.

Remarque 5-1. *Les réductions que nous allons définir pourraient tout aussi bien être formulées dans le calcul des séquents.*

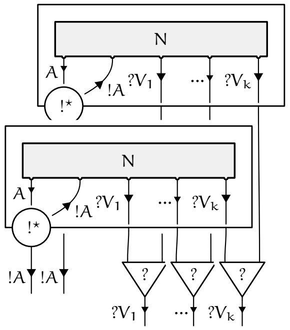
Sortie-Récursion — ***Affaiblissement*** La réduction d'une boîte récursive face à un *affaiblissement* se passe comme s'il s'agissait d'une boîte exponentielle : un *affaiblissement* efface la boîte dans les deux cas.



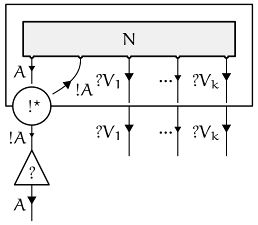
Sortie-Récursion — Contraction La réduction d'une boîte récursive face à une *contraction* se passe elle aussi comme s'il s'agissait d'une boîte exponentielle : une *contraction* duplique la boîte dans les deux cas.



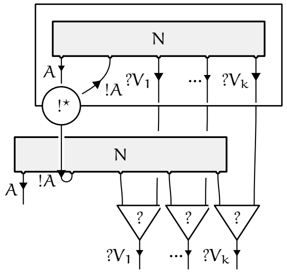
↓



Sortie-Récursion — Déréliction La différence entre une boîte exponentielle et une boîte récursive s'observe lors de l'interaction avec une *déréliction*. Une boîte exponentielle était simplement ouverte lors de la réduction. Une boîte récursive fournit une copie d'elle même au morceau de réseau qu'elle libère. Cette copie lui est accessible grâce au nouveau lien, introduit spécifiquement pour cet effet.



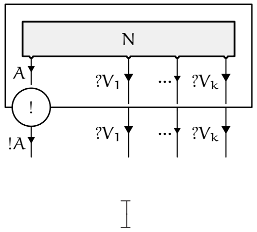
↓

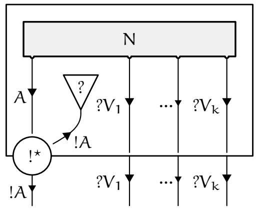


En particulier, une boîte récursive ne disparaît définitivement que lorsqu'elle se trouve face à un affaiblissement. C'est le cas lorsque le réseau N libéré ici lance un *affaiblissement* sur la copie de la boîte qu'il obtient. On dira que le réseau N ne fait aucun appel récursif. Grâce aux constructions additives, l'utilisation d'appels récursifs peut être conditionnelle. Il est d'ailleurs nécessaire que celle-ci soit conditionnelle si on veut faire quelque chose d'intéressant tout en souhaitant que la réduction termine.

Entrée-Promotion — Sortie-Récursion, Entrée-Récursion—Sortie-Promotion, Entrée-Récursion—Sortie-Récursion Précisons en dernier lieu que lorsqu'une boîte (récursive ou exponentielle) se présente face à l'entrée d'une autre boîte (récursive ou exponentielle), la réduction correspondante est similaire à celle qui à déjà été donnée dans le cas de l'interaction entre deux boîtes exponentielles (voir 3.1.2). C'est-à-dire, en faisant rentrer la première boîte dans la deuxième.

On peut également remarquer que la boîte exponentielle classique est un cas particulier d'une boîte récursive (lorsqu'on ne fait aucun appel récursif) :

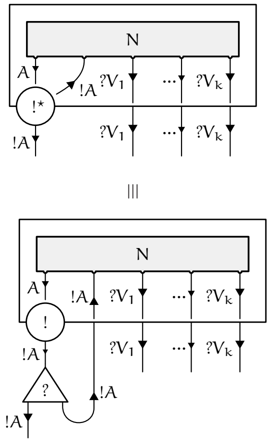




5.1.3 Récursion à l'aide de cycles

Il aurait été possible d'exprimer la récursion sans utiliser une nouvelle forme de boîte, de manière similaire à ce que fait *Raphaël Montelatici* dans [Mon03], qui travaille dans la logique linéaire polarisée. Les boîtes ‹ Y › qu'il introduit ressemblent fortement aux boîtes récursives que nous étudions, mais celles-ci n'utilisent pas la modalité exponentielle pour contrôler leur réduction.

Du point de vue de la sémantique, on constate en effet l'équivalence suivante :



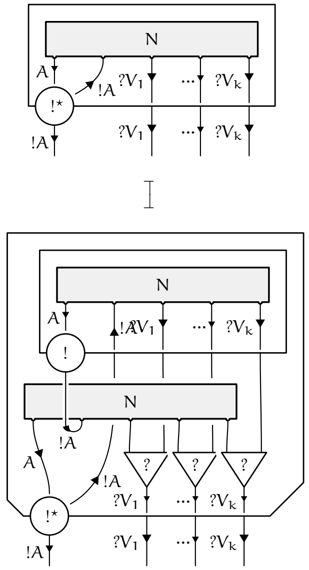
Mais cette présentation aurait deux inconvénients majeurs.

Tout d'abord on remarquera que le réseau utilisé n'est pas correct (il possède un cycle), et c'est d'ailleurs grâce à cela qu'il est capable de ne pas terminer. Mais pour la même raison, il n'est pas possible de l'écrire dans le calcul des séquents. La construction récursive spécifique que nous avons introduite, ainsi que les réductions qui lui sont associées, trouvent elles une place dans le calcul des séquents.

Le deuxième inconvénient de cette présentation avec cycles est qu'elle produit une réduction qui ne termine jamais (le réseau avec cycle qui est proposé se réduit en un réseau qui contient exactement le même type de rédex). Elle déroule indéfiniment chaque construction récursive, même si on fait une utilisation primitive de la récursion. Nous donnerons quelques exemples qui emploient nos boîtes récursives dans des cas simples (primitif récursifs), et qui terminent comme attendu. Ce sera le cas d'une opération d'addition construite par récurrence sur les entiers naturels (cf. 6.1), et d'une opération qui manipule des listes (cf. 5.3.2). Même si le système ne garantit pas la terminaison du calcul (ce n'est pas le but recherché ici), au moins il n'impose pas sa divergence.

5.1.4 Auto-réduction

On peut noter l'équivalence entre les réseaux donnés ci-dessous. Elle s'obtient en considérant la réduction cachée dans la structure d'une boîte récursive qu'on devine grâce à l'équivalence sémantique donnée précédemment.



Autoriser une telle réduction (en orientant l'équivalence vers la droite) rend tout de suite le système de réduction non terminant, étant donné que cette règle peut être appliquée à son propre résultat. C'est précisément ce que nous souhaitons éviter. Néanmoins, cette règle semble avoir un effet d'optimisation qu'il pourrait être intéressant d'exploiter.

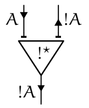
Il est envisageable d'appliquer une fois cette règle aux parties droites d'autres règles (auxquelles elle est applicable) pour obtenir une optimisation déclenchée par l'utilisation. On pourra par exemple l'appliquer aussitôt après l'interaction d'une *déréliction* avec une boîte récursive.

5.2 Récursion localisée

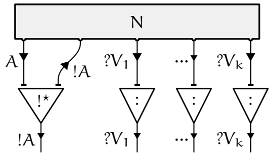
Quelques mots à propos de la localisation des boîtes récursives. On donnera l'idée en se plaçant dans un paradigme actif, mais celle-ci peut tout à fait être réutilisée dans le cas passif et le cas contrôlé.

5.2.1 Construction

Sortie-Récursion$_a$ On introduit un nouveau nœud pour les sorties de boîtes récursives. C'est un nœud binaire avec *niveau incrémenté*. Graphiquement, il sera représenté comme suit :



Récursion$_a$ Une boîte récursive se code localement de la façon suivante :

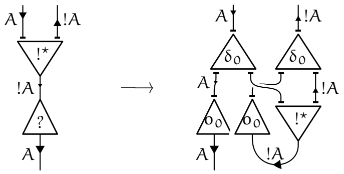


Dans cette construction les entrées de boîtes récursives sont codées à l'aide du même opérateur que celui utilisé pour les entrées de boîtes exponentielles. Nous n'aurons pas besoin d'expliciter une nouvelle fois leurs réductions.

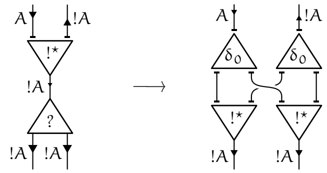
5.2.2 Réduction

Les réductions mettant en jeu les sorties de boîtes récursives sont données ci-dessous.

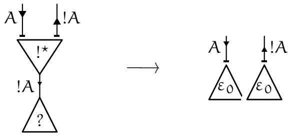
Déréliction — Sortie-Récursion$_a$ Lorsqu'une construction récursive rencontre une *déréliction*, elle s'ouvre après s'être auto-dupliquée. La copie qui n'a pas été ouverte sera utilisée par les appels récursifs. Localement, cela se traduit en la réduction de tête donnée ici.



Contraction — Sortie-Récursion$_a$ Face à une contraction, une construction récursive est répliquée comme une boîte exponentielle.

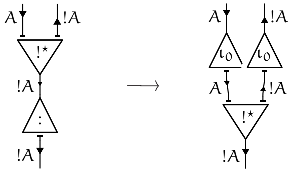


Affaiblissement — Sortie-Récursion$_a$ Face à un *affaiblissement*, une construction récursive est effacée comme une boîte exponentielle.



La dernière règle, qui régit l'interaction d'une boîte récursive face à l'entrée d'une autre boîte, fait intervenir un processus d'« ingestion » similaire au cas d'une boîte exponentielle. C'est pour cette raison que nous avons tout simplement réutilisé l'opérateur qui marque la position des entrées d'une boîte de *promotion* dans le cas des boîtes récursives.

Entrée-Promotion$_a$ ***— Sortie-Récursion***$_a$



5.3 Types récursifs

Nous introduisons ici des types de données inductifs et coinductifs qui permettent l'introduction d'objets complexes. Nous ne ferons pour nos besoins actuels aucune différence entre les deux. Néanmoins, nous ferons en sorte d'utiliser un type inductif lorsque nous souhaitons parler d'objets qui ont une structure récursive finie, et un type coinductif lorsque qu'on souhaite que cette structure soit infinie.

La construction récursive que nous venons d'introduire permet une description finie de ces objets coinductifs.

5.3.1 Notations

Si F désigne un type à une indéterminée, $\mu X.\,F(X)$ et $\nu X.\,F(X)$ désignent des types respectivement inductif et coinductif qui lui sont associés. Ces types vérifient les égalités :

$$\mu X.\,F(X) = F(\mu X.\,F(X)) \qquad \nu X.\,F(X) = F(\nu X.\,F(X))$$

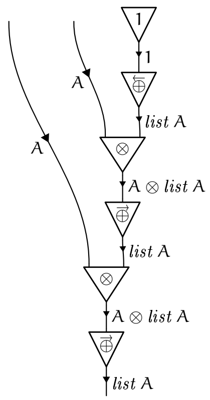
De plus, on notera que les types inductifs et coinductifs sont considérés duaux :

$$(\mu X.\,F(X))^{\perp} = \nu X.\,F^{\perp}(X)$$

Par exemple, un élément de type $\mu X.\,1 \oplus A \otimes X$ sera consommé par le biais d'un lien de type $\nu X.\,\bot \mathbin{\&} A^{\perp} \mathbin{⅋} X$.

5.3.2 Listes

Une liste d'éléments de type A peut être modélisée par un terme de type $\mathit{list}\,A = \mu X.\,1 \oplus A \otimes X$. Ce type vérifie $\mathit{list}\,A = 1 \oplus A \otimes \mathit{list}\,A$. Une liste à deux éléments est construite sur le modèle suivant :

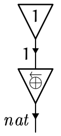


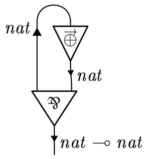
On peut alors considérer la fonction souvent appelée « `list.map` » dans les langages fonctionnels, qui applique une fonction donnée à tous les éléments d'une liste donnée. Nous en présentons une version décrite par un réseau d'interface $!(A \multimap B) \vdash\ !(\mathit{list}\,A \multimap \mathit{list}\,B)$, qui suppose que la fonction à appliquer est linéaire et qu'elle est disponible en quantité arbitraire pour pouvoir être appliquée à chaque élément présent dans la liste (dont le nombre est inconnu).

Voir Fig. 5.1 (page 63).

5.3.3 Entiers

Un entier peut être représenté par une liste d'éléments inertes de type 1, ou plus élégamment par un terme de type $\mathit{nat} = \mu X.\,1 \oplus X$. En particulier, l'élément zéro et la fonction successeur seront codés par les réseaux d'interface $\vdash \mathit{nat}$ et $\vdash \mathit{nat} \multimap \mathit{nat}$ donnés ici :





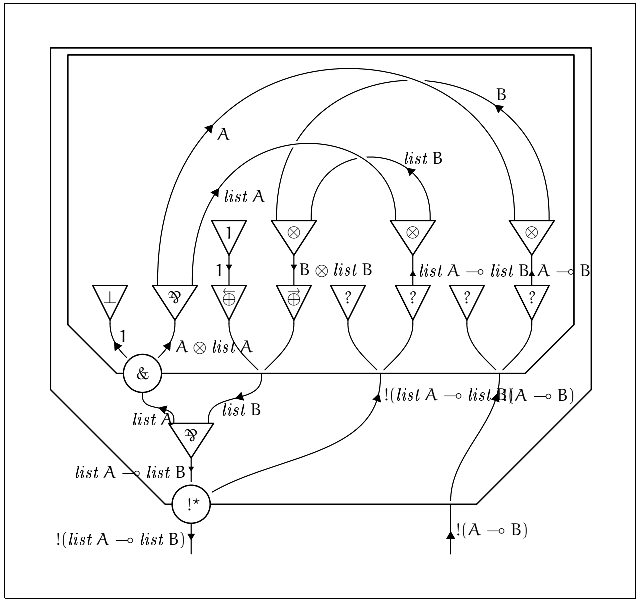


FIGURE 5.1

5.3.4 Arbres

Pour citer un dernier exemple, on peut représenter les arbres binaires purs à l'aide du type inductif $\mu X.\,1 \oplus X \otimes X$.

5.4 Codage du langage *PCF*

PCF est un langage fonctionnel minimaliste atteignant la puissance des machines de *Turing* grâce à des objets de base (des booléens et des entiers) et à une construction récursive. Il existe différentes présentations, sensiblement équivalentes, de ce langage. Nous allons utiliser une formulation adaptée à nos besoins en précisant les correspondances simples qui existent entre celle-ci et d'autres présentations.

5.4.1 Définitions

Les types de *PCF* sont donnés dans la syntaxe suivante :

$$\tau \;::=\; \mathtt{int} \mid \mathtt{bool} \mid \tau \Rightarrow \tau$$

En désignant par $s^{\tau}_{\bar{w}}$ un terme s de type τ dans un environnement $\bar{w} = z_1{:}\omega_1, \ldots, z_k{:}\omega_k$, le noyau opérationnel des termes que nous considérons repose sur la syntaxe (typée) suivante :

$$\begin{array}{rcl} t^{\tau}_{\bar{w}} & ::= & (t^{\sigma\Rightarrow\tau}_{\bar{w}})t^{\sigma}_{\bar{w}} \mid \rho x\, t^{\tau}_{x:\tau,\bar{w}} \\ t^{\tau}_{x:\tau,\bar{w}} & ::= & x^{\tau}_{x:\tau,\bar{w}} \\ t^{\sigma\Rightarrow\tau}_{\bar{w}} & ::= & \lambda x\, t^{\tau}_{x:\sigma,\bar{w}} \end{array}$$

C'est un λ-calcul tout à fait standard enrichi par une nouvelle construction. Cette construction $\rho x\, s$ permet des définitions récursives et est la source de la forte expressivité du langage. La variable liée x permet de désigner dans s le terme qu'on est justement en train de définir. D'autres présentations utilisent à la place une constante particulière Y qui s'exprime ici par $\lambda f\, \rho x\, \langle f \rangle x$.

On complète cette syntaxe par un jeu de constructions suffisamment fourni pour manipuler les booléens :

$$\begin{array}{rcl} t^{\mathtt{bool}} & ::= & \mathbf{true} \mid \mathbf{false} \\ t^{\omega} & ::= & \mathbf{cond}^{\omega}\, t^{\mathtt{bool}}\, t^{\omega}\, t^{\omega} \end{array}$$

et les entiers, par exemple :

$$\begin{array}{rcl} t^{\mathtt{int}} & ::= & \mathbf{zero} \mid \mathbf{succ}\; t^{\mathtt{int}} \\ t^{\omega} & ::= & \mathbf{match}^{\omega}\, t^{\mathtt{int}}\, t^{\omega}\, t^{\mathtt{int}\Rightarrow\omega} \end{array}$$

Ici on n'a pas précisé l'environnement, il est commun à tous les termes qui apparaissent dans une construction, et il est transmis tel quel à l'ensemble de la construction. Aussi, on se restreint à $\omega \in \mathtt{int}, \mathtt{bool}$ si on veut, puisque *PCF* n'inclut traditionnellement pas de polymorphisme.

L'utilisation de la construction **match** n'est pas tout à fait standard. Elle peut être remplacée par un test à zéro et une fonction prédécesseur dont la valeur en zéro peut être choisie arbitrairement. Ce choix a été fait car il reflète mieux la structure inductive des entiers. Pour la même raison, les entiers s'écrivent de façon structurée :

$$\begin{array}{c} \mathbf{zero} \\ \mathbf{succ\ zero} \\ \mathbf{succ\ succ\ zero} \\ \vdots \\ \mathbf{succ} \cdots \mathbf{succ\ zero} \\ \vdots \end{array}$$

Réduction

Le principal ingrédient pour réduction des termes de *PCF* reste la β-réduction usuelle du λ-calcul :

$$(\lambda x\, s)t \longrightarrow s[{}^{t}/_{x}]$$

on y rajoute la règle suivante, qui déroule les définitions récursives :

$$\rho x\, s \longrightarrow s[{}^{\rho x\, s}/_{x}]$$

La réduction n'est pas faite en profondeur dans les termes pour éviter notamment de dérouler inutilement les définitions récursives. On considère habituellement une stratégie d'appel par nom.

Enfin, il faut donner les règles de réduction qui correspondent aux manipulations des booléens et des entiers :

$$\begin{array}{rcl} \mathbf{cond}^{\omega}\ \mathbf{true}\ u\ v & \longrightarrow & u \\ \mathbf{cond}^{\omega}\ \mathbf{false}\ u\ v & \longrightarrow & v \\ \\ \mathbf{match}^{\omega}\ \mathbf{zero}\ u\ f & \longrightarrow & u \\ \mathbf{match}^{\omega}\ (\mathbf{succ}\ n)\ u\ f & \longrightarrow & (f)n \end{array}$$

5.4.2 Traduction

Le langage *PCF* admet un codage (relativement simple) dans les réseaux multiplicatifs, exponentiels, additifs et récursifs. Un type τ dans *PCF* va être interprété par un type linéaire $[\tau]$ défini comme suit :

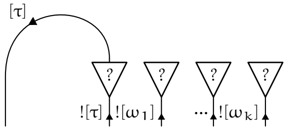
$$\begin{array}{rcl} [\mathtt{bool}] & := & 1 \oplus 1 \\ [\mathtt{int}] & := & \mu X.\,1 \oplus !X \\ [\sigma \Rightarrow \tau] & := & ![\sigma] \multimap [\tau] \\ [\alpha] & := & \alpha \end{array}$$

Notons que nous n'utilisons pas le type $nat = \mu X.\,1 \oplus X$ précédemment évoqué pour traduire le type $\mathtt{int}$! (ce sera expliqué en 5.4.3).

Un terme $t^{\tau}_{\bar{w}}$ est traduit en un réseau $[\![t^{\tau}_{\bar{w}}]\!]$ d'interface $![\omega_1], \ldots, ![\omega_k] \vdash [\tau]$ selon le procédé décrit ici.

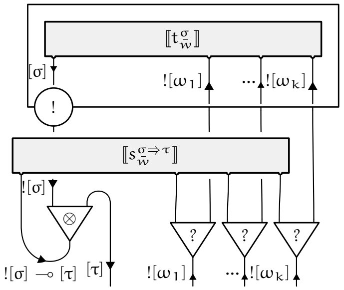
Variable Une variable se code par une simple *déréliction*, une seule instance de l'objet référencé par cette variable est demandée à l'environnement. Les autres liens sont affaiblis.

$[\![x^{\tau}_{x:\tau,\bar{w}}]\!] :=$



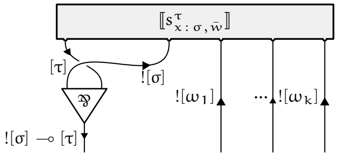
Application L'application se code à l'aide d'une *promotion*, qui rend possible la réplication de l'argument, et d'un nœud *tenseur*, qui effectue la partie linéaire de l'application. L'environnement est distribué à la fois à la fonction et à l'argument grâce à des *contractions*.

$$[\![(s^{\sigma\Rightarrow\tau}_{\bar{w}})t^{\sigma}_{\bar{w}}]\!] :=$$



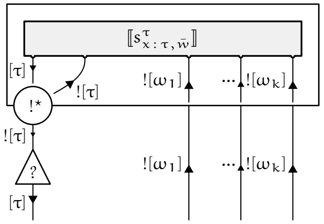
Abstraction L'abstraction se code à l'aide d'un nœud *par*, qui permet d'augmenter le contexte avec l'argument reçu. Cet argument est déjà sous forme répliquable.

$[\![\lambda x\, s^{\tau}_{x:\sigma,\bar{w}}]\!] :=$



Récursion La récursion se code à l'aide d'une boîte récursive.

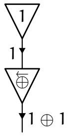
$[\![\rho x\, s^{\tau}_{x:\tau,\bar{w}}]\!] :=$



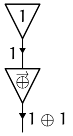
Cette construction crée immédiatement un rédex. C'est l'analogue du rédex $\rho x\, s^{\tau} = (Y^{\tau})\lambda x\, s^{\tau}$ présent dans *PCF*, qui se réduit en $s[\rho x\, s/x]$.

Constructions sur les booléens Les constructions **true** et **false** sont codées par les deux éléments canoniques de type $1 \oplus 1$.

$[\![\mathbf{true}]\!] :=$

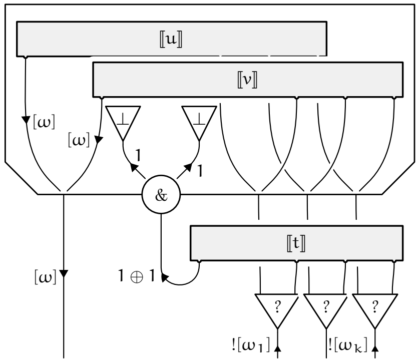


$[\![\mathbf{false}]\!] :=$



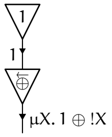
La conditionnelle repose principalement sur l'utilisation d'une boîte *avec*. Le premier argument, de type booléen, passé à la construction $\mathbf{cond}^{\omega}$ est analysé à l'aide de cette construction additive.

$[\![\mathbf{cond}^{\omega}\ t\ u\ v]\!] :=$

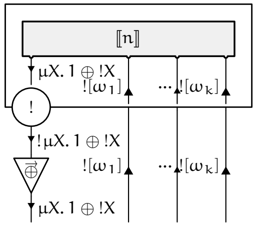


Constructions sur les entiers Les constructions **zero** et **succ** sont codées par deux constructeurs du type $\mu X.\, 1 \oplus !X$.

$[\![\mathbf{zero}]\!] :=$

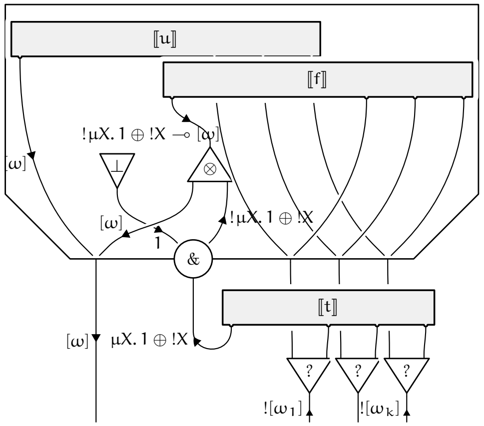


$[\![\mathbf{succ}\ n]\!] :=$



Le filtrage entier est fait à l'aide d'un nœud *avec*, de la même façon que le filtrage booléen. Si le premier argument est l'entier zéro, on renvoie le second argument, si c'est le successeur d'un autre entier, on renvoie l'application du troisième argument à ce dernier.

$[\![\mathbf{match}^{\omega}\ t\ u\ f]\!] :=$

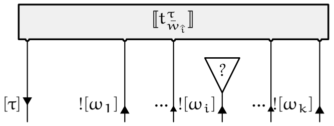


5.4.3 Alternatives à cette traduction

Affaiblissement immédiat

On peut remarquer qu'une variable z_i est dans l'environnement $\bar{w}$ d'un terme t mais que celle-ci n'est pas utilisée, il est alors possible de simplifier son codage en utilisant immédiatement un *affaiblissement* de cette façon :

$[\![t^{\tau}_{\bar{w}}]\!]' :=$



On peut en particulier forcer l'utilisation de cette simplification dès qu'elle est applicable et on retrouve une unicité du codage. Si on retarde l'affaiblissement d'une variable qui n'est pas utilisée, on obtient des nœuds *contraction* qui seront de toute façon suivis de nœuds *affaiblissement*, ça reste équivalent à la traduction canonique et à la traduction simplifiée.

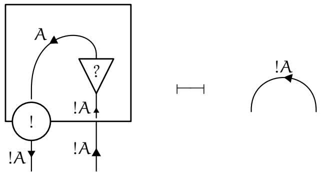
Traduction souple

L'équivalence dont nous venons de parler peut aussi permettre d'assouplir la traduction. Nous allons définir une nouvelle traduction qui va s'exprimer sous la forme d'une relation $t \mapsto N$: à un terme $t^{\tau}_{\bar{w}}$ peut correspondre plusieurs traduits N, ceux-ci sont toujours supposés d'interface $![\omega_1], \ldots, ![\omega_k] \vdash [\tau]$. Nous allons reprendre la traduction rigide $[\![\cdot]\!]$ que nous avons déjà présentée, mais on autorise de placer des arbres quelconques de *contractions* suivis d'*affaiblissements* là ou nous avions placés un seul *affaiblissement* ou une seule *contraction*. Cela va nous permettre d'énoncer plus simplement les propriétés recherchées.

Nous considérons donc une traduction souple $\mapsto$, et la notation $[\![t]\!]$ désignera à présent l'ensemble des traduits d'un terme t (c'est-à-dire les réseaux N tels que $t \mapsto N$). De cette façon, avec la convention sur les ensembles de réseaux (expliquée en 1.3), et en considérant les nœuds *affaiblissement* et *contraction* comme des arbres, la traduction que nous avons déjà donnée exprime exactement la traduction souple qui nous intéresse.

Éviter les expansions exponentielles

La traduction proposée utilise une boîte dans la traduction de chaque application, même s'il s'agit d'une simple variable. Or, lorsque l'argument d'une application est une variable, en remarquant que celle-ci est accessible dans l'environnement sous une modalité exponentielle, il est possible d'utiliser directement le lien fourni par l'environnement. On éviterait ainsi une construction *promotion* qui contient une simple *déréliction*, le tout étant équivalent à un fil :



Entiers légers

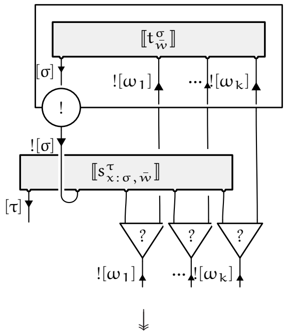
Nous avons utilisé l'interprétation $[\mathrm{int}] = \mu X.\, 1 \oplus !X$ pour traduire le type des entiers. En logique linéaire on peut préférer représenter les entiers par le type $\mu X.\, 1 \oplus X$ et le successeur par une fonction linéaire qui a pour type $\mu X.\, 1 \oplus X \multimap \mu X.\, 1 \oplus X$ (comme décrit en 5.3.3). Mais dans la traduction de notre construction $\mathbf{match}^{\omega}$, on a besoin de fournir le prédécesseur de l'entier filtré en quantité arbitraire à la fonction f. Le langage *PCF*, n'a pas une granularité de type assez fine pour imposer la linéarité de telles fonctions. Il a donc été nécessaire d'utiliser des entiers « lourds » de type $\mu X.\, 1 \oplus !X$ pour que le codage soit relativement naturel.

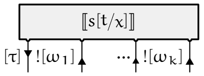
En fait, en présence de récursion, des conversions entre types $\mu X.\, 1 \oplus X$ et $\mu X.\, 1 \oplus !X$ existent, mais elles nécessitent des parcours complets du terme pour chaque conversion. Il est possible d'utiliser ce type $\mu X.\, 1 \oplus X$ si on est prêt à en payer le coût. En revanche, si nous nous intéressons à des inductions sur des listes, les conversions évoquées ne peuvent pas être mises en place aussi simplement. Pour convertir un type $\mu X.\, 1 \oplus A \otimes X$ en un type $\mu X.\, 1 \oplus A \otimes !X$ il faudrait savoir répliquer les éléments de type A. À priori rien ne nous permet de le faire.

5.4.4 Propriétés de la traduction

Comme annoncé, nous considérons ici la traduction souple des termes du langage *PCF*, et on commence par un traditionnel lemme de substitution.

Lemme 5-1. *La réduction suivante est admissible :*





Preuve. Par induction sur s, et on utilise la souplesse de la traduction. □

On prouve grâce à ce lemme un théorème de simulation.

Théorème 5-2. *La traduction $\mapsto$ de PCF qui a été définie respecte la sémantique opérationnelle. La réduction de ce langage est simulée (faiblement) par la réduction des réseaux :*

$$\longleftarrow\!\!\mapsto \;\subseteq\; \mapsto\!\twoheadleftarrow\!\!\!-\!\!\!-$$

Preuve. On traite toutes les formes de réduction présentes dans *PCF*. La β-réduction est simulée par une réduction *par – tenseur* suivie par la réduction donné dans le lemme 5-1 (elle consiste en une réplication de l'argument fourni dans une boîte, suivie de l'ouverture de chacune des copies). La ρ-réduction est simulée par une réduction *déréliction – récursion* suivie par une réplication de la boîte récursive restituée. Les réductions des constructions booléennes et entières sont simulées par des réductions additives *avec – plus gauche* ou *avec – plus droit*. □

5.4.5 Extension du codage aux types récursifs quelconques

Dans le langage *PCF* un seul type est de nature inductive : les entiers. Nous pourrions souhaiter parler d'autres langages non linéaires comprenant des types inductifs construits plus librement, comme celui-ci en *ML* :

```
type 'a tree =
  | Leaf of 'a
  | Node of 'a tree * 'a tree
```

dont le $\mathbf{match}^{\omega}$ correspondant admettrait pour arguments des termes de types $\alpha \Rightarrow \omega$ et $\alpha\,\mathfrak{tree} \Rightarrow \alpha\,\mathfrak{tree} \Rightarrow \omega$.

Nous pouvons y parvenir, mais nous serions une fois de plus tenus d'utiliser un codage « lourd » du type $\alpha\,\mathfrak{tree}$ comme $\mu X.\, ![\alpha] \oplus !X \otimes !X$.

Chapitre 6

Implantation parallèle

Une petite machine virtuelle a été réalisée pour réduire les réseaux. Constructions multiplicatives, additives, exponentielles et récursives sont disponibles. Les boîtes sont représentées sous forme localisée (ici dans le paradigme passif), on travaille donc avec de simples réseaux d'interaction.

De ce travail a émergé un certain nombre de considérations et d'idées, et même si beaucoup sont à l'heure actuelle restées à l'état d'idées, elles vont être évoquées dans ce chapitre.

6.1 Ce qui se passe sur un exemple

Considérons le code suivant, écrit dans un langage fonctionnel usuel :

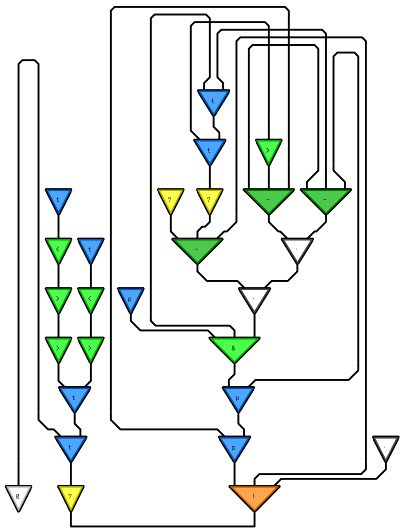
```
type int = Zero | Succ of int

let rec add x y =
  match x with
   | Zero -> y
   | Succ p -> Succ (add p y)

let one = Succ (Zero)
let two = Succ (one)

let result = add two one
```

Le réseau présenté ci-dessous correspond à un calcul similaire de l'expression $2 + 1$, dans laquelle les entiers 1 et 2 sont représentés par les objets respectifs du type $\mu X.\, 1 \oplus X$, que l'on peut nommer « entiers de *Péano* linéaires ». L'opération d'addition est codée à l'aide d'une boîte récursive, et d'une boîte *avec* qui examine le premier argument.



Les symboles « informatisés » correspondent selon l'arité des cellules aux opérateurs suivants :

"`t`" : *tenseur*, *unité*
"`p`" : *par*, *co-unité*
"`?`" : *affaiblissement*, *déréliction*, *contraction*
"`!`" : *sortie récursion (passive)*
"`<`" : *plus gauche*
"`>`" : *plus droit*
"`&`" : *sortie avec (pasive)*
"`-`" : *entrée avec (passive)*
"`.`" : *diffusion*

Dans cette disposition, les liens *coupures* sont placés en bas, et les *axiomes* se trouvent en haut. On reconnaît :

- l'application (cellule *tenseur* en bas à gauche)
- d'un couple (cellule *tenseur* juste au dessus) formé des deux entiers 2 (branche de gauche) et 1 (branche de droite),
- à une fonction (cellule *par* en bas de la partie droite) définie récursivement (à l'aide d'une boîte récursive contenant toute la partie droite de notre réseau).

Le résultat de l'application est connecté à une cellule particulière (tout à gauche) matérialisant l'unique port libre, nommé `0`, de notre réseau.

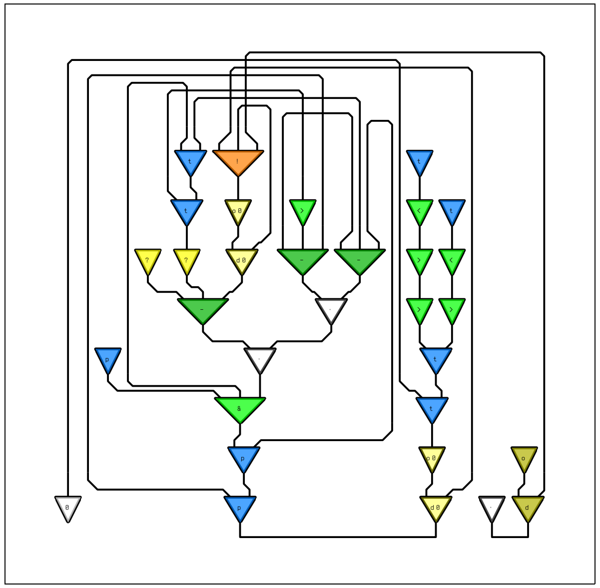


FIGURE 6.1

Réduction

On va à présent effectuer la réduction avec un degré de parallélisation maximal, c'est-à-dire en réduisant tous les rédex visibles simultanément, et mettre en évidence les différentes étapes clefs qu'on rencontre au cours de ce calcul.

La première étape de calcul réduit notre réseau de départ en le réseau décrit ci-dessous, dans lequel on voit apparaître les opérateurs de boîtes δ_0 et o_0 (ainsi que les messages $\bar{\delta}$ et $\bar{o}$) qui permettent le déroulage de la boîte récursive qui définit l'addition :

Voir FIG. 6.1 (page 69).

On passe par un nombre maximal de douze rédex à l'étape 8 (mais c'est aussi le cas à l'étape 16 et 24, et on reconnaît ici une sorte de période de duplication de la boîte) :

Voir FIG. 6.2 (page 70).

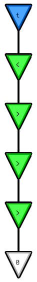
On obtient une première information sur le résultat à l'étape 10, on sait que le résultat est le successeur d'un autre entier (il est donc supérieur à 1) :

Voir FIG. 6.3 (page 70).

Le résultat est entièrement calculé à l'étape 27, c'est une représentation du nombre entier 3, qui apparaît tout à gauche sur le prochain schéma :

Voir FIG. 6.4 (page 71).

Vient ensuite une phase que l'on peut qualifier de pure « garbage collection », et la réduction termine à l'étape 36 sur :



On comprendra que si ce résultat (qui est une re-

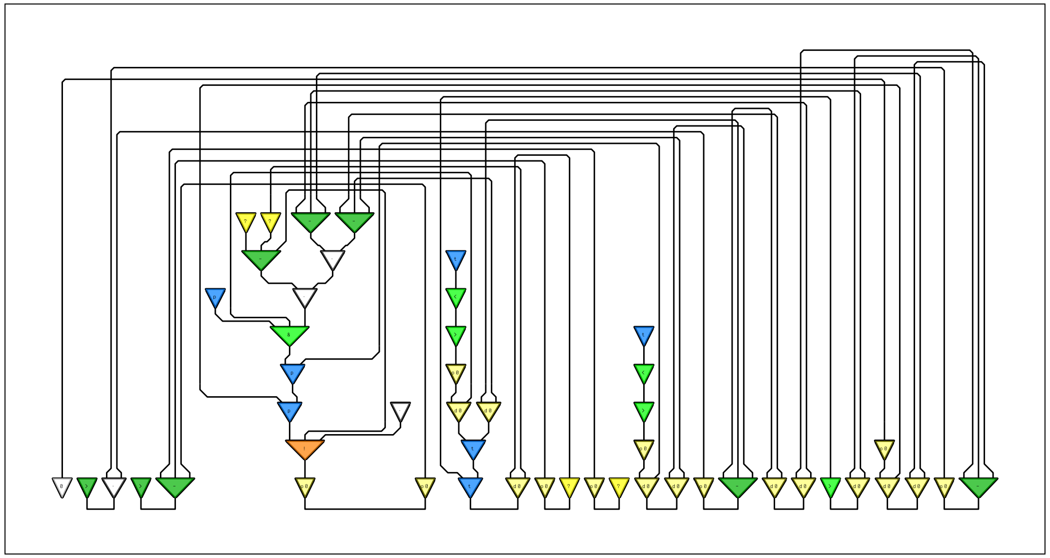


FIGURE 6.2

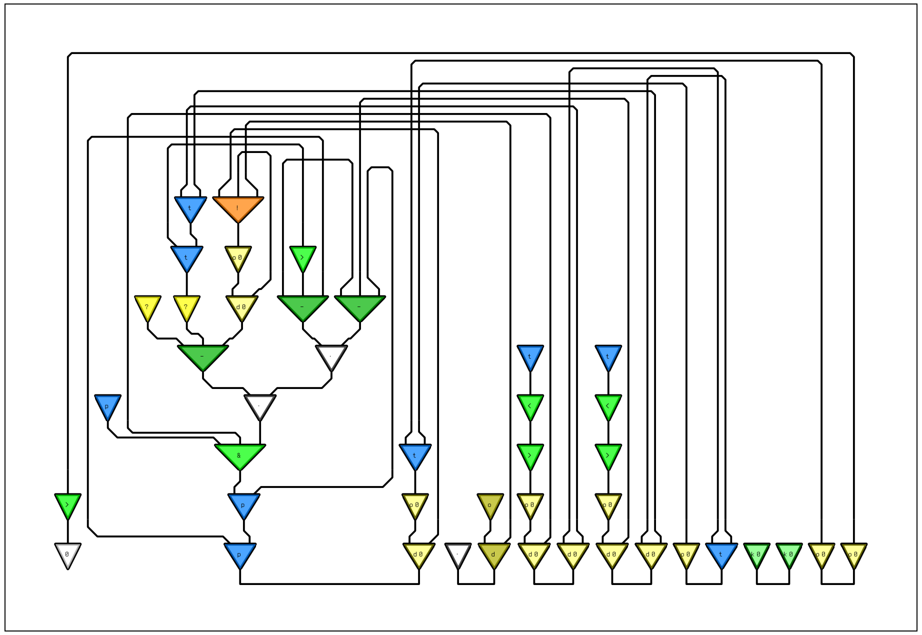


FIGURE 6.3

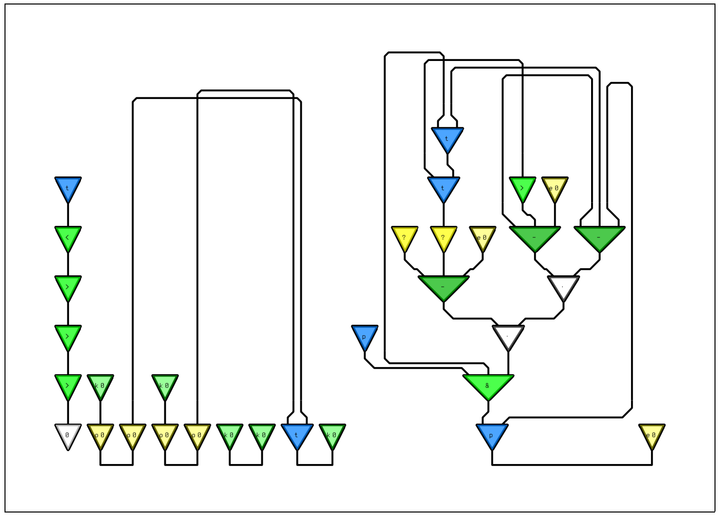


Figure 6.4

présentation du nombre 3) devait être utilisé par une autre partie du réseau, cette nouvelle partie du calcul aurait pu commencer bien avant cette dernière étape. La réduction préalable du réseau hypothétique représentant une continuation aurait pu démarrer dès l'étape 1 ; son interaction avec la valeur qu'il attend dès l'étape 10. On constate ici le grand intérêt d'une exécution parallèle.

6.2 Vers l'exécution parallèle de langages fonctionnels

De nos jours les ordinateurs sont seulement équipés de quelques processeurs (ou cœurs). Leur nombre est le plus fréquemment de 1, 2, 4, ou 8. On ne peut donc espérer atteindre le degré de parallélisation maximal considéré précédemment. On peut néanmoins essayer d'exploiter au mieux les différentes unités de calcul qui sont à notre disposition.

6.2.1 Où trouver suffisamment de parallélisme ?

À ce jour, il y a eu plusieurs tentatives, mais aucun outil n'a été adopté à grande echelle qui permette l'exécution parallèle de programmes écrits naturellement (c'est-à-dire, sans manipulations de « threads » ou autres procédures de synchronisation ad-hoc). Les langages impératifs reposent entièrement sur la notion de séquentialité, et il est très difficile de trouver quelles parties d'un programme écrit dans de tels langages peuvent être exécutés de façon parallèle. De plus, le degré de parallélisation final que l'on peut espérer est faible : deux morceaux de programme qui accèdent ou modifient le contenu d'une même variable ne pourront être exécutés en parallèle.

Par contre, dans des langages fonctionnels purs les effets de bord n'existent pas, et on peut se rendre compte que beaucoup de programmes possèdent un parallélisme caché. Considérons le programme suivant, qui définit le tri « fusion » sur des listes (on supposera que `split` et `merge` sont deux fonctions déjà écrites : la première partage une liste en deux parties, si celle-ci n'est pas atomique, et la deuxième fusionne deux listes déjà triées) :

```
let rec msort l =
  match split l with
  | Atomic l0 -> l0
  | Splited (l1, l2) ->
      merge (msort l1) (msort l2)
```

En l'absence d'effets de bord, le seul fait que « `msort l1` » et « `msort l2` » soient deux arguments distincts de l'appel à la fonction `merge` fait qu'il est possible de paralléliser totalement leur exécution. À leur tour ces appels récursifs pourront de nouveau être parallélisés. De fait, si nous avions disons quatre processeurs, très rapidement (dès le deuxième appel récursif) nous pourrions tous les faire travailler. Au final on aura rentabilisé nos quatre processeurs puisqu'en comparaison avec une exécution séquentielle, il nous faudra à peu de choses près quatre fois moins de temps pour obtenir notre résultat.

Avec des ressources de calcul infinies. . .

Les algorithmes de tri ont typiquement un coût en $O(n \log n)$. Si nous disposions d'un nombre infini de processeurs, par ce simple processus de parallélisation, nous pourrions même observer un « glissement » de complexité. Malgré sont coût en $O(n \log n)$, notre programme de tri pourrait être exécuté en temps linéaire $O(n)$.

Plus spectaculairement, si nous travaillions avec des arbres dont on a décoré les feuilles par des entiers, et que par exemple nous reherchions leur valeur maximale :

```
let rec tree_max t =
  match t with
  | Leaf i -> i
  | Node (t1, t2) ->
     max (tree_max t1) (tree_max t2)
```

une implantation séquentielle répondrait en $O(n)$ où n désigne le nombre de nœuds de l'arbre, alors qu'une implantation parallèle de ce même programme peut répondre en temps $O(h)$, où h désigne cette fois la hauteur de l'arbre. Pour des arbres équilibrés, n est exponentiellement plus grand que h, on a tout intéret à utiliser une implantation parallèle.

6.2.2 Comment exploiter le parallélisme caché ?

Les travaux décrits dans la première partie de cette thèse montrent que par traduction dans les réseaux d'interaction il est envisageable d'exécuter de façon parallèle les langages purement fonctionnels. On peut notamment obtenir un grain de parallélisation très fin grâce aux représentations localisés des boîtes.

En effet, grâce à leur structure de graphe, il est possible d'utiliser à plein temps la totalité des processeurs disponibles pour réduire différentes parties d'un même réseau. Cela ne requiert qu'une synchronisation très rudimentaire dans les cas où deux processeurs viendraient à travailler sur des parties adjacentes de ce graphe.

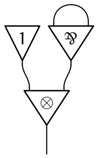
D'un point de vue technique, ces synchronisations peuvent être effectuées en mémoire partagée à l'aide d'instructions processeur atomiques dites « test and set » disponibles sur tous les processeurs actuels. Il n'est pas nécessaire (et ce serait gravement pénalisant) d'utiliser les synchronisations offertes par un système d'exploitation.

6.2.3 Essais d'exécution parallèle

Pour justifier cela, une implantation test constituée d'environ 1000 lignes de code C a été écrite. Elle permet de réduire avec un nombre arbitraire de processeurs un réseau construit avec différents opérateurs, dont ceux évoqués dans l'exemple.

Un certain nombre d'emplacements mémoire de taille fixée sont réservés et sont capables d'accueillir les données nécessaires à la description d'une cellule du réseau que l'on souhaite réduire. Chaque emplacement possède des adresses qui permettent de le référencer. Parmi les données que l'on écrit dans ces emplacements, on trouve la sorte de la cellule représentée, ainsi que des pointeurs vers d'autres emplacements qui définissent les connections qu'a cette cellule avec d'autres cellules.

Par exemple, le réseau :



est représenté en mémoire par :

⊗	q_0	r_0	—
p_0	p_1	p_2	p_3

1	—	—	—
q_0	q_1	q_2	q_3

⅋	r_2	r_1	—
r_0	r_1	r_2	r_3

Chaque cellule est stockée sur un petit bloc de mémoire contenant quatre emplacement d'adresses consécutives (dans l'exemple, ces adresses ont été notées p_0, p_1, p_2, p_3 pour la cellule *tenseur*). Le premier emplacement est utilisé pour stocker la sorte de la cellule considérée. Les autres emplacements servent à représenter les connexions entre les cellules. À l'adresse p_1 on indique où le premier port auxiliaire de la cellule *tenseur* est connecté, c'est ici q_0 (qui sert de nom pour le port principal de la cellule *unité*). À l'adresse p_2 on indique où le deuxième port auxiliaire est connecté, c'est ici r_0 (pour le port principal de la cellule *par*). Une cellule *tenseur* n'ayant pas de troisième port auxiliaire on laisse l'emplacement d'adresse p_3 vide.

Quant aux coupures du réseau, celles-ci sont stockées dans une liste séparée qui contient des couples d'adresses construits avec les adresses des cellules qui sont liées par leur ports principaux. Pour effectuer la réduction on utilise des processus (créés de façon à ce que chacun puisse être animé par un processeur) qui viennent piocher un couple dans cette liste et effectuent les modifications adéquates sur la représentation mémoire décrite pour réduire le rédex correspondant, avant de recommencer.

Nous ne rentrerons pas dans les détails techniques d'implantation, le lecteur intéressé trouvera le code source de la machine disponible sur le web.

Remarque 6-1. *D'autres approches ont été envisagées. Une implantation différente de systèmes de réseaux d'interaction à été proposée par Jorge Sousa Pinto [Sou00], elle est basée sur une machine abstraite nommée MPINE. Une autre de ses approches pour implanter de façon parallèle le λ-calcul utilise la géométrie de l'interaction [Sou01].*

6.3 Un protocole pour le calcul parallèle

Notre machine virtuelle repose sur le modèle client – serveur, et en réalité une grosse partie du code sert à décrire la partie serveur, seule une petite partie sert à la réduction proprement dite des réseaux. Lors d'une utilisation typique, un client souhaite déléguer le calcul d'un certain réseau à notre machine virtuelle. Il l'envoie donc à travers un canal de communication (et selon un certain protocole) à cette machine. Celle-ci effectue la réduction avec tous les processeurs dont elle dispose et retransmet progressivement en direction du client (selon le même protocole) la forme normale du réseau qui lui a été transmis.

Une chose est intéressante à souligner : le protocole de transmission d'un réseau est fondé sur la structure même des réseaux d'interaction. À chaque sorte de cellule considérée correspond un message qui peut être transmis sur un canal de communication. Pour faire fonctionner le tout il suffit d'introduire des couples de cellules particulières *in* et *out* pour matérialiser un lien distant, plus trois messages particuliers qui permettent la création d'une coupure, l'envoi d'un axiome, et le changement d'orientation d'un canal.

Sans donner de véritables spécifications techniques, nous allons illustrer ici le fonctionnement de ce protocole.

Situation initiale

Prenons l'exemple d'un client qui souhaite utiliser la machine virtuelle pour obtenir le résultat d'un calcul. Symboliquement, on va supposer que ce client souhaite connaître le résultat obtenu lorsqu'on fait l'application de la fonction identité à l'élément unité. Du point de vue du client, la coupure qui relie ces deux éléments est représentée à l'aide un nœud *coupure* explicite, car ce n'est pas à lui d'effectuer la réduction.

`asy/6c202081-std5`

Une connexion est initialisée, et un message demandant la création d'une coupure mentionnant deux noms frais a et b est envoyé sur le canal de communication qui relie le client au serveur. Le système passe alors dans l'état décrit ci-dessous, dans lequel deux morceaux de réseau (situés côté client) sont reliés par deux liens orientés (nommés a et b) à une coupure (située côté serveur).

`asy/5e2aa10a-std5`

Envoi des données

Les données du calcul n'ont pas encore été envoyées au serveur. Cette transmission va être effectuée de façon asynchrone dans un ordre arbitraire, fondé sur la structure des réseaux d'interaction. Les cellules *out* interagissent en « envoyant » les cellules sur lesquelles elles sont coupées en direction du serveur à l'aide de messages appropriés qui référencent la sorte de la cellule à transmettre, le nom du lien auquel elle est attachée et un certain nombre de noms de liens (fraîchement générés) supplémentaires. Après réception du message par le serveur, ces cellules sont reconstruites au niveau de la cellule *in* située de l'autre côté du lien désigné.

Supposons que la coupure du haut, impliquant la cellule *par*, soit effectuée en premier. La transaction aboutit au résultat suivant :

`asy/4287c43d-std5`

Les nouveaux noms frais c et d choisis par le client font partie du message envoyé et ont servi à instaurer deux nouveaux liens qui permettront la transmission ultérieure des parties du réseaux anciennement connectés aux ports auxiliaires de la cellule *par* qui vient d'être transmise.

L'envoi de la cellule *tenseur* sur le lien nommé b est effectuée de la même manière.

`asy/ff074fee-std5`

On constate que dès à présent le serveur a la possibilité d'effectuer le calcul souhaité par le client, c'est-à-dire de réduire la coupure entre les cellules *par* et *tenseur* qu'il vient de recevoir. On va néanmoins finir la transmission des données avant d'effectuer cette vraie étape de calcul.

Supposons donc que la prochaine étape sera celle qui permet d'envoyer l'axiome qui est censé relier les deux ports auxiliaires de la cellule *par*. Un message particulier est utilisé lorsque deux cellules *out* forment un rédex, il contient les noms des liens sur lesquels sont connectés l'axiome (ici c et d), et la réduction s'effectue comme suit :

`asy/4f3d1655-std5`

Le serveur a connecté les liens branchés sur les deux cellules *in* correspondantes. On transmet ensuite la cellule *unité* sur le lien e :

`asy/6606a324-std5`

Puis, lors d'une dernière étape d'envoi, la cellule *get* (qui délimite la partie du réseau à envoyer au serveur) vient demander l'inversion du sens de transmission des données pour le dernier lien nommé ici f :

`asy/dfe370b4-std5`

Calcul effectif

Vient ensuite le calcul effectif qui normalise notre réseau de départ, à présent situé coté serveur. Nous n'avons dans cet exemple simple besoin que d'une étape de calcul, mais s'il s'agissait d'un exemple réaliste, le temps de calcul effectif serait typiquement d'un ordre de grandeur largement supérieur au temps pris pour la transmission des données. On effectue donc notre réduction côté serveur :

`asy/9d8c0107-std5`

Réception des données

Dans une dernière phase, il convient d'informer le client du résultat calculé. Pour cela le protocole déjà utilisé pour l'envoi des données depuis le client vers le serveur va être réutilisé. On utilise le nouveau lien qui a pour nom g, et cette fois les rôles sont inversés : le serveur envoie et le client réceptionne.

`asy/e0801306-std5`

On atteint l'état final, et le client sait désormais que le résultat qu'il attendait est l'élément *unité*. Le résultat de notre exemple est simple, et une seule étape de communication a suffit pour la réception du résultat. Plus aucun lien n'est partagé entre client et serveur, le canal de communication peut alors être physiquement rompu.

Remarque 6-2. *Les différentes phases (envoi des données, calcul, réception des données) ont été expliquées séparément et décrites dans un ordre séquentiel. Ces trois phases peuvent cependant être entrelacées : le calcul effectif côté serveur, et même la réception d'un résultat partiel côté client, peuvent commencer* *avant la fin de l'envoi du réseau à réduire. On comprend pourquoi un tel protocole est adapté à la réduction parallèle.*

Bien sûr, rien n'empêche le serveur de servir plusieurs clients en même temps, et de tout calculer dans le même espace. Ces calculs s'effectueront dans des composantes connexes distinctes sans interférer.

On notera cependant que rien n'empêche un client de transmettre des réseaux mal formés (mal typés, ou contentant des cycles par exemple) qui peuvent provoquer des erreurs lors de la réduction. Une solution à ce problème n'est pas forcement évidente lorsque celui-ci se pose, c'est-à-dire dans un cadre où le client n'est pas considéré fiable. Un problème similaire apparaît lorsqu'un client provoque (sans nécessairement avoir de mauvaises intentions) l'exécution d'un réseau qui ne termine pas. Le serveur n'a dans ce cas aucun moyen de s'en rendre compte.

6.4 Perspectives

6.4.1 Répartition de charge

Aucune étude n'a été faite pour l'instant dans cette direction, mais il est envisageable de compléter le protocole pour parvenir à faire fonctionner ensemble plusieurs machines virtuelles du type décrit précédemment, qui seraient déployées sur des machines physiques distinctes.

Il serait possible que le serveur, se rendant compte qu'il est surchargé, décide de réduire une coupure (ici *la* coupure) de la façon suivante :

Voir Fig. 6.5 (page 76).

en délèguant le calcul à une troisième machine qu'il connaît, et qu'il sait disponible.

La difficulté réside dans la décision de déléguer la réduction d'une certaine coupure (parce qu'il ne reste plus de processeurs inoccupés) à une autre machine distante (qui elle est inoccupée ou partiellement inoccupée). Il n'est pas possible d'évaluer la quantité de calcul que représente cette coupure et, s'il s'avère que celle-ci était triviale à réduire, on risque de payer au prix fort la latence de communication avec la machine distante.

Il est possible que des mécanismes de répartition de charge d'un ordre totalement différent puissent être mis en place, mais des considérations en lien avec les études de complexité implicite pourraient constituer une aide à ce type de décisions, et peuvent constituer une voie de recherche. Une coupure qui peut être écrite dans la logique linéaire multiplicative *MLL* sera probablement réduite en peu d'étapes (complexité linéaire), alors que la réduction d'une coupure faisant intervenir des constructions de la logique linéaire exponentielle *MELL* prendra vraissemblablement plus de temps. Les études de complexité implcite sont nées avec l'apparition de systèmes intermédiaires *LLL* [Gir98], *SLL* [Laf04] qui correspondent à une complexité polynomiale, et *ELL* [DJ03] qui caractérise la complexité élémentaire.

6.4.2 Communication symétrique

Les liens de communication qui ont été utilisés sont orientés, et cela leur confère une propriété utile d'asynchronisme. En effet, à tout instant, seul l'un des deux acteurs est susceptible d'envoyer un (et un unique) message mentionnant le nom d'un lien donné.

Protocoles spécialisés

Si on veut utiliser le méta-protocole décrit ici pour faire communiquer deux programmes différents, qui souhaitent s'échanger des données selon un protocole spécialisé (fixé par le type d'un lien initialement partagé), on peut envisager d'utiliser ces liens orientés, formés par des couples de cellules *in* et *out*, mais cela présente des inconvénients. Si on ne change jamais l'orientation de ces liens, tout le calcul sera effectué par la machine relié au côté *in*. Et en particulier le programme censé s'exécuter côté *out* est obligé de dévoiler le code qui correspond à sa part de l'interaction (puisqu'elle sera calculée de l'autre côté).

Pour palier à ces inconvénients on peut penser à faire changer l'orientation des liens à l'aide de cellules *get* correctement insérées. Cela peut poser de lourdes complications. Une modification du système de types semble nécessaire si on souhaite assurer une cohérence dans la communication.

Liens synchrones

Une alternative consiste à concevoir des liens symétriques de la forme suivante :

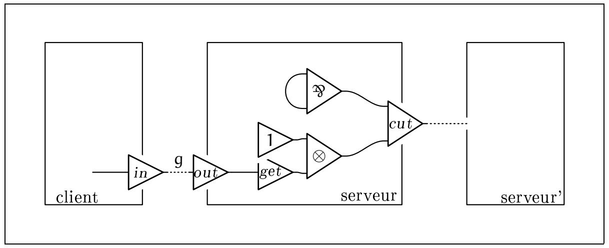


Figure 6.5

`asy/3eb405cb-std5`

Il faut alors prendre des précautions pour gérer correctement les paires critiques qui peuvent apparaître (comme dans l'exemple donné). Deux messages risquent d'être transmis dans des sens opposés, et il faut faire en sorte de résoudre cette éventualité et d'aboutir invariablement dans un état cohérent.

Au choix, on peut faire migrer la cellule *par* vers la machine M_2, ou la cellule *tenseur* vers la machine M_1. Dans le premier cas, la machine M_1 devra se charger de la réduction multiplicative qui apparaît, dans le deuxième cas se sera à la machine M_2 de s'en charger. En s'autorisant l'intervention de certains facteurs aléatoires, on arrive alors facilement à répartir la charge sur les deux machines. Cela dit, on n'est toujours pas en mesure de garantir la propriété de sécurité évoquée.

Interaction « sécurisée »

Une approche plus intéressante serait de décider que chaque machine reconstruit la partie du réseau correspondant à la réduction du rédex transmis. La mise en place concrète de cette alternative semble un peu délicate, car il faut marquer certaines cellules du statut « transmise » tout en les conservant pour interpréter correctement les messages envoyés par le correspondant ; c'est néanmoins envisageable.

De plus amples études seraient nécessaires, mais cette spécification symétrique semble conférer à l'interaction les propriétés de sécurité recherchées. Étant donné que le contenu d'une fonction (qui se trouve derrière une cellule *par*) n'est pas transmis au correspondant, celui-ci n'a accès qu'au résultat de l'interaction avec les arguments qu'il fournit. De tels liens permettent une communication tout en honorant la frontière spatiale qui sépare deux programmes d'identités distinctes dont les exécutions sont menées de façon concurrente.

Remarque 6-3. *L'utilisation d'un tel protocole est possible, et éventuellement intéressante, indépendamment du modèle d'exécution choisi (parallèle ou non, à base de réseaux d'interaction ou pas...).*

6.4.3 Matériel dédié à la réduction de réseaux d'interaction

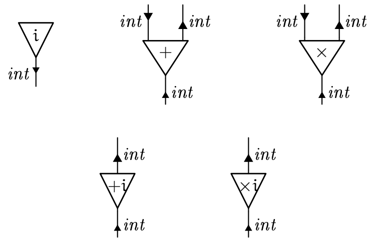
On a envisagé l'exploitation du parallélisme implicite de programmes fonctionnels à l'aide de matériel existant. Peut on envisager l'utilisation de matériel spécifique pour exploiter plus profondément ce parallélisme ?

De petits processeurs ?

Les processeurs actuels savent exécuter des milliers d'instructions complexes. Pourtant, la machine virtuelle qui a été réalisée utilise une infime partie de ces instructions. Il serait intéressant d'utiliser de tout petits processeurs (de moindre coût) en grande quantité pour espérer réduire encore plus efficacement les réseaux d'interaction.

Les processeurs sachant faire de grosses opérations arithmétiques seront toujours utiles si on a besoin d'accélérer les calculs sur des structures de données particulieres telles que les entiers (pour le traitement de vidéos par exemple) ou les flotants (traitement audio, 3d...). Si on utilise de petits processeurs pour réduire la majorité des coupures correspondant au flot d'exéctution d'un programme (qui est décrit avec des opérateurs de la logique linéaire), on peut laisser aux processeurs « arithmétiques » le traitement de la réduction des coupures impliquant certaines cellules « arithmétiques » spéciales qui permetent un calcul accéléré.

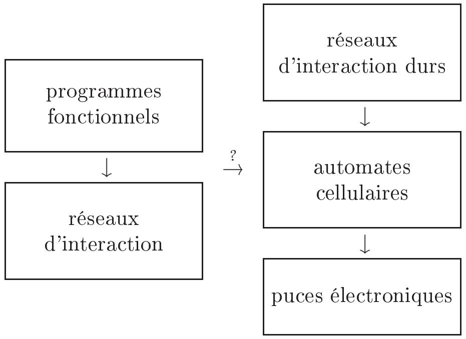
Par exemple, on pourra utiliser les cellules qui ont été présentées dans le premier chapitre :



Des puces électroniques dédiées ?

Une étape supplémentaire pourrait être de coder la réduction des réseaux d'interaction à l'intérieur même d'une puce électronique.

Yves Lafont à proposé une variante des réseaux d'interaction appelées réseaux d'interaction durs [BL08]. À la différence des réseaux d'interaction usuels, ces derniers sont contraints à une géométrie fixe : seules les sortes des cellules et la position du port principal des cellules d'un rédex peuvent être modifiées. Il ne semble pas difficile de réduire des réseaux d'interaction durs à l'aide d'automates cellulaires (et donc de puces électroniques). Mais coder les réseaux d'interaction (généraux) dans les réseaux d'interaction durs semble poser de grosses difficultés, et cela forme le maillon manquant :



Je n'ai connaissance d'aucun travail de recherche allant dans cette direction. Une idée consisterait à choisir une cellule dure pour coder chaque cellule d'un réseau d'interaction donné et de maintenir des liens de communication distante entre deux cellules dures qui codent des cellules originales voisines. Les problèmes majeurs qui apparaissent alors sont, non seulement, le routage qu'il est nécessaire de mettre en place pour établir ces liens, mais surtout l'allocation et la désallocation d'espace pour stocker de nouvelles cellules issues d'une réduction. En particulier sur une puce en deux dimensions, un programme dont la taille croît exponentiellement (en nombre de cellules) ne pourra probablement pas être déployé avec un débit qui suit cette croissance sur l'intégralité de la puce.

Deuxième partie

Approche différentielle

Chapitre 7

Ressources

L'idée était déjà présente dans les travaux de *Gerard Boudol* sur le λ-calcul avec multiplicités [Bou93]. On souhaite définir un calcul intermédiaire entre le λ-calcul linéaire dont l'expressivité est trop limitée et le λ-calcul ordinaire pour lequel on n'a aucun contrôle sur les effets de réplication qui peuvent intervenir.

7.1 Le λ-calcul avec ressources

Le λ-calcul avec ressources Δ défini par *Thomas Ehrhard* et *Laurent Regnier* dans [ER08] est un λ-calcul modifié de façon à ce que toute variable apparaisse en position linéaire. Contrairement à ce qui se fait dans le λ-calcul linéaire, un lieur va servir à lier plusieurs variables de même nom. Lors d'une application on va alors fournir, non plus une seule ressource (comme dans le λ-calcul linéaire), ni une ressource répliquable (comme dans le λ-calcul ordinaire) mais des paquets de ressources. Au cours d'une réduction, chacune des ressources contenues dans un paquet est destinée à être consommée, mais elle ne sera ni effacée ni dupliquée.

7.1.1 Syntaxe

Définition 7-1. *Le λ-calcul avec ressources Δ est défini grâce à la syntaxe suivante :*

$$t \;::=\; \begin{array}{c|c|c} x & \lambda x\, t & \langle t\rangle T \\ 0 & \multicolumn{2}{c}{t+t} \end{array} \qquad T \;::=\; \begin{array}{c|c|c} 1 & TT & t \\ 0 & \multicolumn{2}{c}{T+T} \end{array}$$

Le symbole t *représente un terme, et le symbole* T *désignera ce que l'on appelle un paquet.*

La somme $\langle + \rangle$ et $\langle 0 \rangle$ son élément neutre vérifient les propriétés qui confèrent à Δ (l'ensemble des termes) et à $\Delta^!$ (l'ensemble des paquets), des structures de monoïdes commutatifs. Les paquets seront de plus quotientés par commutativité, associativité et neutralité du paquet vide $\langle 1 \rangle$ pour l'opération produit. Si u_i désigne une famille de termes, un paquet pourra donc être décrit grâce à la notation suivante :

$$T = u_1 \cdots u_n$$

Enfin, on quotiente par (bi-)linéarité de toutes les constructions :

$$\begin{array}{rcl@{\qquad}rcl} \lambda x\,(u+v) &=& \lambda x\, u + \lambda x\, v & \lambda x\, 0 &=& 0 \\ \langle u+v\rangle T &=& \langle u\rangle T + \langle v\rangle T & \langle 0\rangle T &=& 0 \\ \langle t\rangle (U+V) &=& \langle t\rangle U + \langle t\rangle V & \langle t\rangle 0 &=& 0 \\ (U+V)T &=& UT + VT & 0T &=& 0 \end{array}$$

Les termes qui peuvent être construits sans somme sont appelés *termes simples*. Un terme du λ-calcul avec ressources est une somme de termes simples.

Substitution linéaire

Dans ce calcul on souhaite substituer les variables d'un terme occurrence par occurrence. Une variable et une ressource étant choisies, il reste à décider de l'occurrence de variable qui va recevoir cette ressource. L'opération que nous allons définir fait la somme de tous les termes obtenus selon ce dernier choix.

Définition 7-2. *On considère une opération de substitution linéaire d'une variable* z *par un terme* u *dans un terme* t*. Elle sera notée* $\frac{\partial}{\partial z} t \cdot u$*, et définie par :*

$$\frac{\partial}{\partial z} x \cdot u \;:=\; 0 \qquad \frac{\partial}{\partial z} z \cdot u \;:=\; u$$

$$\frac{\partial}{\partial z} (\lambda x\, s) \cdot u \;:=\; \lambda x\, \frac{\partial}{\partial z} s \cdot u$$

$$\frac{\partial}{\partial z} (\langle r\rangle R) \cdot u \;:=\; \left\langle \frac{\partial}{\partial z} r \cdot u \right\rangle R + \langle r\rangle \frac{\partial}{\partial z} R \cdot u$$

où l'opération de substitution linéaire sur les paquets étend la précédente opération de la façon suivante :

$$\frac{\partial}{\partial z} 1 \cdot u \;:=\; 0$$

$$\frac{\partial}{\partial z} ST \cdot u \;:=\; \left(\frac{\partial}{\partial z} S \cdot u\right) T + S \left(\frac{\partial}{\partial z} T \cdot u\right)$$

La substitution linéaire sur les paquets est bien définie de manière unique ; on peut vérifier qu'elle est compatible avec la commutativité et l'associativité du produit.

Propriété 7-3. *Les opérateurs* $\frac{\partial}{\partial x}$ *et* $\frac{\partial}{\partial y}$ *commutent, on donne une forme généralisée de la propriété de Schwarz :*

$$\frac{\partial}{\partial y}\left(\frac{\partial}{\partial x} t \cdot u\right) \cdot v + \frac{\partial}{\partial y} t \cdot \left(\frac{\partial}{\partial x} v \cdot u\right)$$

$$=$$

$$\frac{\partial}{\partial x} t \cdot \left(\frac{\partial}{\partial y} u \cdot v\right) + \frac{\partial}{\partial x}\left(\frac{\partial}{\partial y} t \cdot v\right) \cdot u$$

On remarquera que dans le cas habituel où $x \notin v$ (on utilise cette notation pour indiquer que x n'apparaît pas libre dans v) et $y \notin u$, on retrouve :

$$\frac{\partial}{\partial y}\left(\frac{\partial}{\partial x} t \cdot u\right) \cdot v \;=\; \frac{\partial}{\partial x}\left(\frac{\partial}{\partial y} t \cdot v\right) \cdot u$$

Preuve. Par induction sur t, voir [ER08]. On traite le cas TT :

$$\frac{\partial}{\partial x} ST \cdot \left(\frac{\partial}{\partial y} u \cdot v\right) + \frac{\partial}{\partial x}\left(\frac{\partial}{\partial y} ST \cdot v\right) \cdot u \;=$$

$$+\left[\begin{array}{l} \left(\frac{\partial}{\partial x} S \cdot \left(\frac{\partial}{\partial y} u \cdot v\right) + \frac{\partial}{\partial x}\left(\frac{\partial}{\partial y} S \cdot v\right) \cdot u\right) T \\ S\left(\frac{\partial}{\partial x} T \cdot \left(\frac{\partial}{\partial y} u \cdot v\right) + \frac{\partial}{\partial x}\left(\frac{\partial}{\partial y} T \cdot v\right) \cdot u\right) \\ \left(\frac{\partial}{\partial x} S \cdot u\right)\left(\frac{\partial}{\partial y} T \cdot v\right) + \left(\frac{\partial}{\partial y} S \cdot v\right)\left(\frac{\partial}{\partial x} T \cdot u\right) \end{array}\right.$$

or les deux premiers termes sont symétriques en (x, u) et (y, v) par hypothèse d'induction et la somme des deux autres termes est symétrique par permutation. □

7.1.2 Sémantique opérationnelle

La substitution linéaire nous permet de définir une réduction sur les λ-termes avec ressources.

Définition 7-4. *On définit la réduction* $\longrightarrow$ *par clôture contextuelle de deux réductions élémentaires sur les termes simples :*

$$\langle \lambda x\, s\rangle 1 \;\longrightarrow\; s\left[{}^{0}/{}_{x}\right]$$

$$\langle \lambda x\, s\rangle rR \;\longrightarrow\; \left\langle \lambda x\, \frac{\partial}{\partial x} s \cdot r \right\rangle R$$

Il est tout de même nécessaire de préciser le sens de clôture contextuelle ici. On ne demande que la réduction soit stable que par ajout de contextes à une seule inconnue. En particulier, pour la construction *somme* qui fait ici partie de la syntaxe cela signifie que $t_1 + t_2 \longrightarrow t_1' + t_2$ dès que $t_1 \longrightarrow t_1'$ (et que $t_1 + t_2 \longrightarrow t_1 + t_2'$ dès que $t_2 \longrightarrow t_2'$).

La propriété de *Schwarz* nous assure la confluence des paires critiques que suppose cette définition.

Propriété 7-5. *La réduction* $\longrightarrow$ *est localement confluente :*

$$\longrightarrow\longleftarrow \;\subseteq\; \twoheadleftarrow\!\!\longrightarrow\!\!\!\!\twoheadrightarrow$$

Preuve. On dresse une liste exhaustive des paires critiques. Les termes contenant un rédex de la forme $\langle \lambda x\, s\rangle r_1 r_2 R$ présentent une paire critique qui se réduit ainsi :

$$\left\langle \lambda x\, \frac{\partial}{\partial x} s \cdot r_1 \right\rangle r_2 R \;\longrightarrow\; \left\langle \lambda x\, \frac{\partial}{\partial x}\left(\frac{\partial}{\partial x} s \cdot r_1\right) \cdot r_2 \right\rangle R$$

$$\left\langle \lambda x\, \frac{\partial}{\partial x} s \cdot r_2 \right\rangle r_1 R \;\longrightarrow\; \left\langle \lambda x\, \frac{\partial}{\partial x}\left(\frac{\partial}{\partial x} s \cdot r_2\right) \cdot r_1 \right\rangle R$$

or les deux termes sur lesquels on aboutit sont égaux. Une paire critique peut également apparaître lors de la réduction de deux rédex distincts dans un terme de la forme $\langle \lambda y\, c[\langle \lambda x\, s\rangle uU]\rangle vV$, elle se réduit de la façon suivante :

$$\left\langle \lambda y\, c\left[\left\langle \lambda x\, \frac{\partial}{\partial x} s \cdot u \right\rangle U\right]\right\rangle vV \;\longrightarrow$$

$$\left\langle \lambda y\, \frac{\partial}{\partial y}\left(c\left[\left\langle \lambda x\, \frac{\partial}{\partial x} s \cdot u \right\rangle U\right]\right) \cdot v \right\rangle V$$

$$= \;+\left[\begin{array}{l} \left\langle \lambda y \left(\frac{\partial}{\partial y} c \cdot v\right)\left[\left\langle \lambda x\, \frac{\partial}{\partial x} s \cdot u \right\rangle U\right]\right\rangle V \\ \left\langle \lambda y\, c\left[\left\langle \lambda x\, \frac{\partial}{\partial y}\left(\frac{\partial}{\partial x} s \cdot u\right) \cdot v \right\rangle U\right]\right\rangle V \end{array}\right.$$

et

$$\left\langle \lambda y\, \frac{\partial}{\partial y}\left(c[\langle \lambda x\, s\rangle uU]\right) \cdot v \right\rangle V \;=$$

$$+\left[\begin{array}{l} \left\langle \lambda y \left(\frac{\partial}{\partial y} c \cdot v\right)[\langle \lambda x\, s\rangle uU] \right\rangle V \\ \left\langle \lambda y\, c\left[\left\langle \lambda x\, \frac{\partial}{\partial y} s \cdot v \right\rangle uU\right]\right\rangle V \end{array}\right. \quad \longrightarrow^2$$

$$+\left[\begin{array}{l} \left\langle \lambda y \left(\frac{\partial}{\partial y} c \cdot v\right)\left[\left\langle \lambda x\, \frac{\partial}{\partial x} s \cdot u \right\rangle U\right]\right\rangle V \\ \left\langle \lambda y\, c\left[\left\langle \lambda x\, \frac{\partial}{\partial x}\left(\frac{\partial}{\partial y} s \cdot v\right) \cdot u \right\rangle U\right]\right\rangle V \end{array}\right.$$

une fois encore la propriété de *Schwarz* nous dit qu'on aboutit sur les même termes. Les réductions des paires critiques que l'on trouve dans les termes de la forme $\langle\lambda y\, c[\langle\lambda x\, s\rangle 1]\rangle \nu V$, $\langle\lambda y\, c[\langle\lambda x\, s\rangle u U]\rangle 1$ ou $\langle\lambda y\, c[\langle\lambda x\, s\rangle 1]\rangle 1$ confluent de la même manière. □

On remarque aussi qu'on aurait obtenu une confluence forte si on avait considéré une réduction telle que $t_1 + t_2 \longrightarrow t_1' + t_2'$ lorsque $t_1 \longrightarrow t_1'$ et $t_2 \longrightarrow t_2'$, et $0 \longrightarrow 0$.

Normalisation

On peut montrer que la réduction normalise fortement par un simple argument portant sur la taille des termes [ER08].

7.1.3 Calcul typé

On donne en dernier lieu la version typée de ce calcul, c'est celle que l'on souhaite traduire dans les réseaux.

On se donne préalablement une syntaxe pour des types simples :

$$\tau \quad ::= \quad \alpha \;\mid\; \tau \to \tau$$

puis on introduit le λ-calcul avec ressources typé $\overrightarrow{\Delta}$ en restreignant la définition des termes de Δ à la grammaire ci-dessous, où $t^{\tau}_{\bar{w}}$ représente un terme t qui possède le type τ dans un environnement $z_1 : \omega_1, \ldots, z_n : \omega_n$:

$$\begin{aligned}
t^{\tau}_{x:\tau,\bar{w}} &::= x \\
t^{\tau}_{\bar{w}} &::= \langle t^{\sigma\to\tau}_{\bar{w}}\rangle T^{\sigma}_{\bar{w}} \\
t^{\sigma\to\tau}_{\bar{w}} &::= \lambda x\, t^{\tau}_{x:\sigma,\bar{w}} \\
T^{\sigma}_{\bar{w}} &::= 1 \;\mid\; T^{\sigma}_{\bar{w}} T^{\sigma}_{\bar{w}} \;\mid\; t^{\sigma}_{\bar{w}}
\end{aligned}$$

Un terme $t \in \Delta$ est dit typable (noté $t \in \bar{\Delta}$) lorsqu'il existe au moins un type τ et un environnement $\bar{w}$ qui lui permettent d'être construit selon cette grammaire. On notera $t^{\tau}_{\bar{w}} \in \overrightarrow{\Delta}$ le terme typé associé à t correspondant. Par exemple, le terme $\eta = \langle\lambda x\, \langle y\rangle x\rangle z$ est typable puisqu'il peut être typé $\eta^{\alpha}_{y:\beta\to\alpha,z:\beta}$. Notons que nous pourrions lui associer d'autres termes typés. En revanche le terme $\lambda x\, \langle x\rangle x$ ne l'est pas.

Propriété 7-6. *La réduction* $\longrightarrow$ *et compatible avec les types. Lorsque un terme* t *peut être typé* $t^{\tau}_{\bar{w}}$*, dans toute réduction* $t \longrightarrow s$*, le réduit* s *peut être typé de la même façon :* $s^{\tau}_{\bar{w}}$.

Preuve. Un ingrédient principal : si t et u peuvent être typés $t^{\tau}_{x:\sigma}$ et u^{σ}, alors $t' = \frac{\partial}{\partial x} t \cdot u$ peut être typé $t'^{\tau}_{x:\sigma}$. □

7.2 Réseaux différentiels

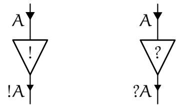
Introduits dans [ER06b], les nœuds différentiels peuvent être compris comme des opérateurs d'acheminement de ressources. Ces opérateurs rendent possible l'acheminement d'un nombre indéterminé de ressources de type A sur un canal de type $!A$.

7.2.1 Opérateurs différentiels

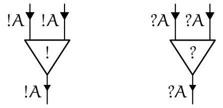
On introduit trois nouveaux nœuds notés avec un symbole ‹ ! ›. Ils vont jouer les rôles duaux (producteurs de ressources) aux trois opérateurs consommateurs de ressources déjà rencontrés, qui étaient notés avec un symbole ‹ ? ›.

Co-affaiblissement, Affaiblissement Ces nœuds correspondent respectivement à une production et une consommation nulles.

Co-déréliction, Déréliction Ces nœuds correspondent respectivement à une production et une consommation unitaires.



Co-contraction, Contraction Ces nœuds permettent de former des productions et des consommations multiples en combinant respectivement deux productions et deux consommations.



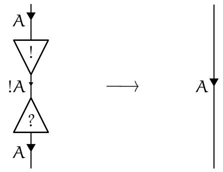
Les opérateurs exponentiels d'*affaiblissement* de *déréliction* et de *contraction* étaient introduits négativement (cf. 1.2.4) dans le cadre de la logique linéaire, ils le restent dans le cadre de la logique linéaire différentielle. Les opérateurs exponentiels duaux, le *co-affaiblissement*, la *co-déréliction* et la *co-contraction* sont dits opérateurs différentiels et sont introduits positivement. Nous allons voir juste après qu'ils permettent d'exprimer le λ-calcul avec ressource.

Sortis du cadre logique (on considère des réseaux d'interaction quelconques, pas nécessairement valides) le système constitué par ces six opérateurs est appelé *DIN*, ou « réseaux d'interaction différentiels ». Ce système est intéressant car il crée un pont entre algèbres de processus et logique (linéaire). Il permet d'interpréter un π-calcul finitaire [EL07].

7.2.2 Réductions

La première règle de réduction que nous allons évoquer est la règle la plus fondamentale, elle correspond à la transmission d'une ressource atomique.

Déréliction — Co-déréliction Le problème d'une production unitaire face à une consommation unitaire se résout en une liaison d'acheminement atomique. C'est celle que l'on peut souhaiter observer pour spécifier le comportement du système.

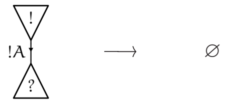


Lorsqu'un producteur et un consommateur de ressources de type A (qui communiquent donc par un lien de type $!A$) souhaitent se transmettre n ressources, notre système va faire en sorte que la règle que nous venons de présenter soit utilisée n fois. Il reste a développer une stratégie d'acheminement de ressources.

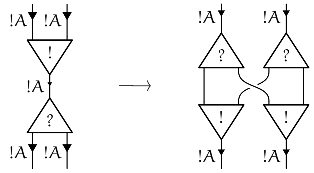
Réductions de propagation

Un premier jeu de règles de réduction participe à la résolution de l'acheminement de ressources sans nécessiter de faire un choix. Il est constitué des quatre règles présentées ici.

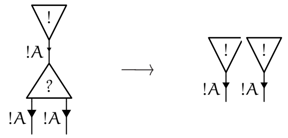
Affaiblissement — Co-affaiblissement Le problème d'une production nulle face à une consommation nulle se résout silencieusement. Le rédex disparait simplement du réseau qui le contient.



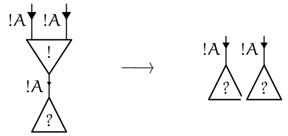
Contraction — Co-contraction Le problème d'une production multiple face à une consommation multiple se propage en laissant des canaux capables d'effectuer tout transfert entre l'un des points de production et l'un des points de consommation.



Contraction — Co-affaiblissement Lorsqu'il n'y a pas de producteurs on ne peut produire pour aucune des composantes d'une consommation multiple.



Affaiblissement — Co-contraction Lorsqu'il n'y a pas de consommateurs on ne peut consommer en provenance d'aucune des composantes d'une production multiple.



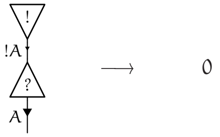
Réductions indéterministes

Les quatre réductions restantes sont des cas où le problème de l'acheminement des ressources est impossible, ou bien des cas où l'on est obligés d'envisager deux possibilités pour trouver une solution.

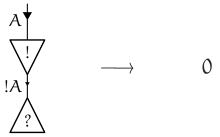
Pour travailler avec un tel indéterminisme, nous allons devoir généraliser la notion de réseau. Comme nous l'avons fait pour les λ-termes, nous allons ajouter une somme ‹ + › et un élément neutre ‹ 0 › en se plaçant dans le monoïde commutatif libre, qu'on notera $\mathcal{N}^+$, engendré par l'ensemble $\mathcal{N}$ des réseaux d'interaction que nous avons considérés jusque là. On utilisera dorénavant l'appellation réseau pour désigner un élément du monoïde, et on parlera de *réseau simple* pour désigner l'un des éléments générateurs.

Les règles de réduction que nous avons donné jusque là pour l'ensemble $\mathcal{N}$ sont étendues à tout réseau de $\mathcal{N}^+$: si $N_1 \longrightarrow N_1'$ alors $N_1 + N_2 \longrightarrow N_1' + N_2$. Les nouvelles règles, présentées ci-dessous, ne peuvent elles être exprimées que dans ce cadre étendu.

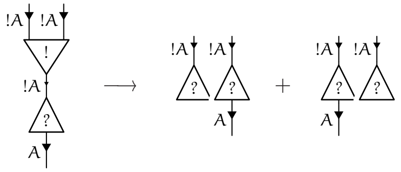
Déréliction — Co-affaiblissement Le problème d'un consommateur unitaire face à un producteur nul n'a pas de solutions. On exprime cela en disant que le rédex se réduit en un réseau nul, il faut comprendre que le réseau entier qui contient ce rédex se réduit au réseau nul.



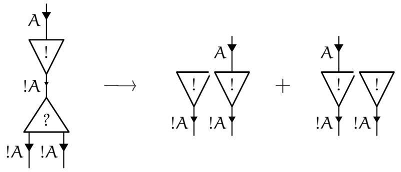
Affaiblissement — Co-déréliction Le problème d'un producteur unitaire face à un consommateur nul n'a pas de solutions.



Déréliction — Co-contraction Dans le cas d'un consommateur unitaire face à un producteur multiple, il est possible de trouver des solutions en le routant vers l'une ou l'autre des composantes du producteur.



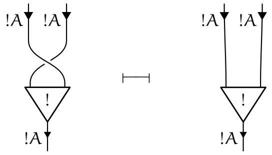
Contraction — Co-déréliction Dans le cas d'un producteur unitaire face à un consommateur multiple, il est possible de trouver des solutions en le routant vers l'une ou l'autre des composantes du consommateur.

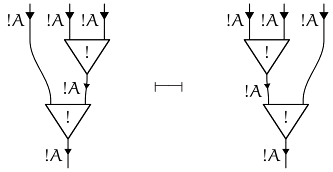


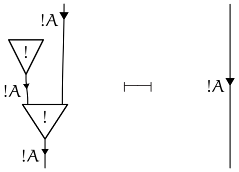
L'introduction de la somme nous a permis d'internaliser la notion de tentative de réduction. On fait échouer de telles tentatives en les réduisant au réseau nul, celles-ci n'auront alors plus aucune influence sur le résultat.

7.2.3 Équivalences

Pour les opérateurs *contraction* et *affaiblissement*, des équivalences exprimaient les propriétés de *commutativité*, d'*associativité* et de *neutralité* pour le produit correspondant, c'est toujours le cas en présence des opérateurs différentiels. Par symétrie du système, des équivalences similaires existent pour les opérateurs duaux *co-contraction* et *co-affaiblissement*.





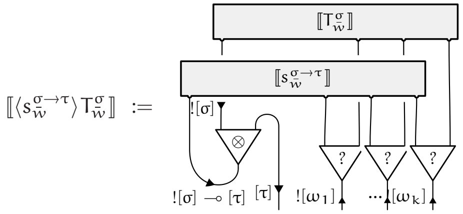


7.3 Codage du λ-calcul avec ressources

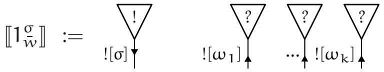
Nous allons à présent expliquer comment le λ-calcul avec ressources peut être exprimé dans le cadre de la logique linéaire différentielle.

7.3.1 Traduction

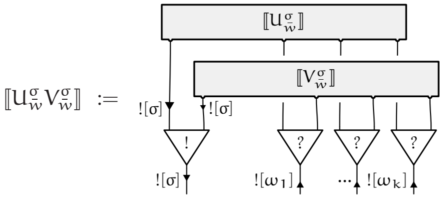
Le λ-calcul sous-jacent se traduit de façon habituelle, et les paquets vont pouvoir être exprimés à l'aide des nouveaux opérateurs. Pour cela, on reprend traduction usuelle $[\![\cdot]\!]$ du λ-calcul dans les réseaux donnée en 5.4.2, à ceci près que la traduction d'une application est faite sans boîte exponentielle, elle est remplacée par la traduction du paquet correspondant :



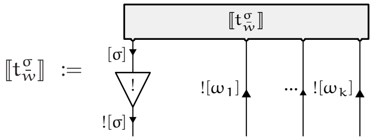
L'argument dans l'application est construit par traduction des paquets. Le paquet vide est exprimé par :



Un paquet composé est exprimé grâce à :



Et un paquet constitué d'un seul terme t s'exprime :



7.3.2 Idées pour une bissimulation

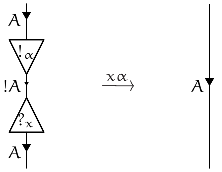
Pour établir un lien précis entre λ-calcul avec ressources et réseaux d'interaction différentiels, on peut penser à étiqueter par des symboles α, β… à la fois les ressources du λ-calcul avec ressources et les nœuds *co-déréliction* qui servent à les traduire. Pendant la traduction, on étiquette aussi les nœuds *déréliction* par les variables auxquelles ils correspondent.

Suite à cela on peut étiqueter les règles de réduction et définir des systèmes de transitions étiquetés. La réduction du λ-calcul avec ressources est étiquetée par $x\alpha$ lorsque la variable du rédex est x et la ressource choisie dans le paquet correspondant est étiquetée par α :

$$\langle \lambda x\, s\rangle 1 \xrightarrow{\tau} s\left[{}^{0}\!/_{x}\right]$$

$$\langle \lambda x\, s\rangle r^{\alpha} R \xrightarrow{x\alpha} \left\langle \lambda x\, \frac{\partial}{\partial x} s \cdot r \right\rangle R$$

Concernant les réseaux différentiels, toutes les réductions sont étiquetées par τ, sauf celle qui définit l'interaction entre *co-déréliction* et *déréliction* :



Ces réductions étiquetées sont étendues aux sommes : si $N_1 \xrightarrow{x\alpha} N_1'$ et $N_2 \xrightarrow{x\alpha} N_2'$ alors $N_1 + N_2 \xrightarrow{x\alpha} N_1' + N_2'$. On étend les τ-réductions aux sommes de façon différente : si $N_1 \xrightarrow{\tau} N_1'$ alors $N_1 + N_2 \xrightarrow{\tau} N_1' + N_2$.

Mais pour établir un théorème de bissimulation [SM92 ; Pou07] entre le λ-calcul avec ressources et réseaux différentiels ainsi étiquetés, il faudrait d'une part considérer les λ-termes à σ-équivalence près, et d'autre part faire attention à certains choix effectués par des τ-réductions côté réseaux. Pour contourner ce dernier point on pense pouvoir utiliser une notion un peu plus faible que la bissimulation (faible) appelée *simulation couplée*, et originellement introduite par *Uwe Nestmann* et *Benjamin C. Pierce* [NC00].

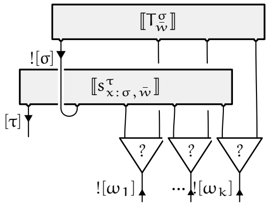
Nous nous contenterons ici d'un théorème de simulation, de même nature que celui donné pour le langage *PCF* dans la première partie.

7.3.3 Simulation

Nous considérons une traduction souple, de la même manière qu'en 5.4.3 : une relation $t \mapsto N$ permet de traduire un terme $t^{\tau}_{\bar w}$ en différents réseaux N d'interface $![\omega_1], \dots, ![\omega_k] \vdash [\tau]$ qui se comporteront de façon similaire. La notation $[\![t]\!]$ désigne l'ensemble des traduits du terme t.

Nous reprenons l'idée qui consiste à remplacer les *affaiblissements* et les *contractions* de la traduction rigide par des arbres. Cela est en réalité nécessaire pour que la traduction soit compatible avec le quotient fait sur la structure des paquets.

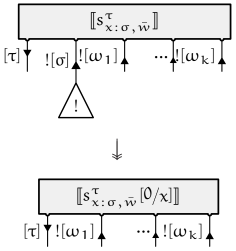
Nous rajoutons une seconde souplesse : la possibilité de traduire un terme qui est sous forme de rédex par un réseau dans lequel la coupure multiplicative correspondante a déjà été réduite :



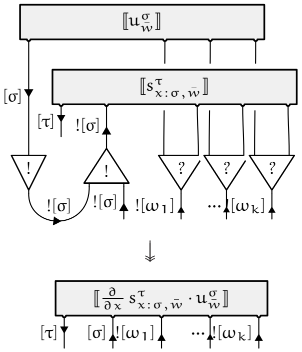
$$\in [\![\langle \lambda x\, s^{\tau}_{x:\sigma,\bar w}\rangle T^{\sigma}_{\bar w}]\!]$$

Cela va nous permettre de contourner la difficulté introduite par la réintroduction d'une *abstraction* et d'une *application* dans la deuxième forme de réduction du λ-calcul avec ressources.

Lemme 7-7. *Les réductions suivantes sont admissibles :*



et :

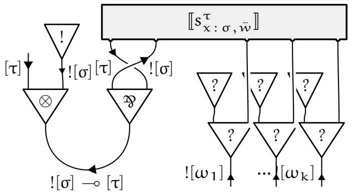


Preuve. Par induction sur s, dans les deux cas on utilise la souplesse de la traduction. □

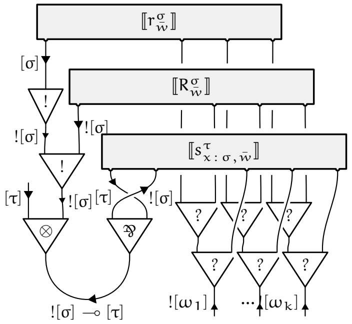
Théorème 7-8. *La traduction ↦ du λ-calcul avec ressources dans les réseaux différentiels respecte la sémantique opérationnelle. La réduction de ce calcul est simulée (faiblement) par la réduction des réseaux :*

$$\longleftarrow\!\mapsto \;\subseteq\; \mapsto\!\twoheadleftarrow\!\!-$$

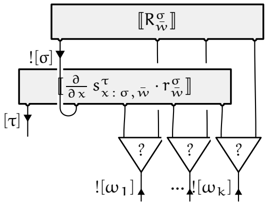
Preuve. Les deux réductions sont contextuelles, et il n'y a que deux types de réduction à simuler. Le terme de gauche dans la première réduction $\langle \lambda x\, s\rangle 1 \xrightarrow{\tau} s[0/x]$ s'exprime du côté des réseaux par :



Il se réduit grâce au premier point du lemme 7-7 (et la souplesse de la traduction) vers un terme de $[\![s[0/x]]\!]$. En ce qui concerne la deuxième réduction $\langle \lambda x\, s\rangle r R \xrightarrow{x\alpha} \left\langle \lambda x\, \frac{\partial}{\partial x} s \cdot r \right\rangle R$, le membre de gauche se traduit en :



Il se réduit par réduction multiplicative et grâce au deuxième point du lemme 7-7 en :



Et ce réseau appartient bien à l'ensemble $[\![\left\langle \lambda x\, \frac{\partial}{\partial x} s \cdot r \right\rangle R]\!]$. □

Chapitre 8

Ressources et promotion

Ce chapitre s'attache à combiner les opérateurs différentiels et la construction *promotion* présentée dans la première partie.

Nous allons dans un premier temps réintroduire la notion de promotion dans le λ-calcul avec ressources, afin de pouvoir exprimer ressources et réplication au sein d'un même calcul. Le calcul obtenu de cette façon est très proche du λ-calcul différentiel, leurs différences seront évoquées. Nous passerons alors dans le monde des réseaux dans lequel existe un système analogue que nous décrirons. C'est une étape nécessaire avant d'introduire les véritables nouveautés qui seront développées dans le chapitre suivant. Une fois de plus, la manière naturelle d'exprimer la dynamique des réseaux s'écarte significativement des procédés de réduction que l'on trouve dans les divers calculs fondés sur des substitutions de variables.

Plusieurs systèmes présentés dans ce chapitre ont été étudiés conjointement avec *Paolo Tranquilli*. Les travaux qu'il présente dans sa thèse [Tra08] offrent des résultats sur le système de réseaux restreint à un cadre intuitionniste. Ceux-ci permettent en particulier de déduire un résultat de confluence sur l'un des λ-calculs qu'on va introduire (muni d'une réduction dite *groupée*). Néanmoins cela ne fonctionne que dans un cadre typé puisqu'il dépend d'un résultat de normalisation.

8.1 Le λ-calcul avec ressources et promotion

8.1.1 Syntaxe

Définition 8-1. *On définit le λ-calcul avec ressources et promotion $\Delta_!$ grâce à la syntaxe suivante :*

$$t \ ::= \ \begin{array}{l} x \ \mid \ \lambda x\, t \ \mid \ \langle t\rangle T \\ 0 \ \mid \ t+t \end{array}$$

$$T \ ::= \ \begin{array}{l} 1 \ \mid \ TT \ \mid \ t \ \mid \ t^! \\ 0 \ \mid \ T+T \end{array}$$

On a repris la syntaxe du λ-calcul avec ressources (sans promotion) et on a introduit une nouvelle construction de paquet : la promotion $t^!$ d'un terme t. L'ensemble des termes $\Delta_!$ et l'ensemble des paquets $\Delta_!^!$ sont toujours munis d'une structure de monoïde commutatif.

Les paquets sont quotientés par commutativité et associativité et neutralité pour la somme et le produit. Et cette fois, on quotiente par (bi-)linéarité de toutes les constructions sauf de la *promotion*, pour laquelle l'équivalence qu'il convient d'utiliser est la suivante :

$$(u + v)^! \ = \ u^! v^! \qquad 0^! \ = \ 1$$

On rappelle que les termes qui peuvent être construits sans somme sont appelés *termes simples*. Un terme du λ-calcul avec ressources et promotion est une somme de termes simples.

8.1.2 Réduction atomique

La dynamique de la réduction atomique que nous allons définir repose une fois de plus sur une opération de substitution linéaire.

Substitution linéaire

L'opération de substitution linéaire d'une variable z par un terme u dans un terme t est toujours notée $\frac{\partial}{\partial z} t \cdot u$. Elle est définie en étendant la définition inductive donnée en 7.1.1 pour la nouvelle construction de paquets :

$$\frac{\partial}{\partial z} t^! \cdot u \ := \ \left(\frac{\partial}{\partial z} t \cdot u\right) t^!$$

Cette opération est toujours définie de manière unique et compatible avec la commutativité et l'associativité du produit. Ses propriétés sont conservées.

Propriété 8-2. *Les opérateurs $\frac{\partial}{\partial x}$ et $\frac{\partial}{\partial y}$ commutent, c'est la propriété de Schwarz :*

$$\frac{\partial}{\partial y}\left(\frac{\partial}{\partial x} t \cdot u\right) \cdot v + \frac{\partial}{\partial y} t \cdot \left(\frac{\partial}{\partial x} v \cdot u\right)$$

$$=$$

$$\frac{\partial}{\partial x} t \cdot \left(\frac{\partial}{\partial y} u \cdot v\right) + \frac{\partial}{\partial x}\left(\frac{\partial}{\partial y} t \cdot v\right) \cdot u$$

Preuve. Par induction sur t. La construction $t^!$ ne casse pas la propriété :

$$\frac{\partial}{\partial x} t^! \cdot \left(\frac{\partial}{\partial y} u \cdot v\right) + \frac{\partial}{\partial x}\left(\frac{\partial}{\partial y} t^! \cdot v\right) \cdot u \ =$$
$$+\left[\begin{array}{l} \left(\frac{\partial}{\partial x} t \cdot \left(\frac{\partial}{\partial y} u \cdot v\right) + \frac{\partial}{\partial x}\left(\frac{\partial}{\partial y} t \cdot v\right) \cdot u\right) t^! \\ \left(\frac{\partial}{\partial x} t \cdot u\right)\left(\frac{\partial}{\partial y} t \cdot v\right) t^! \end{array}\right.$$

le premier terme est symétrique par hypothèse d'induction, le second est symétrique par construction. □

Propriété 8-3. *La commutation entre substitution linéaire et substitution habituelle se passe de la façon suivante, dans le cas où $x \notin v$ et $y \notin u$:*

$$\left(\frac{\partial}{\partial x} t \cdot u\right)[{}^{v}\!/_{y}] \ = \ \frac{\partial}{\partial x} t[{}^{v}\!/_{y}] \cdot u$$

Preuve. Par induction sur t :

(Variable) Lorsque $t = x$, l'égalité n'est autre que $u = u$, et est par conséquent vérifiée. Lorsque $t = z (\neq x)$, l'égalité devient $0 = 0$, elle est également vérifiée.

(Abstraction) Dans le cas où $t = \lambda z\, s$, après calcul, l'égalité s'écrit $\lambda x \left(\frac{\partial}{\partial z} s \cdot u\right)[v/y] = \lambda z \frac{\partial}{\partial x} s[y/v] \cdot u$, qui est équivalente à l'hypothèse d'induction $\left(\frac{\partial}{\partial z} s \cdot u\right)[v/y] = \frac{\partial}{\partial x} s[y/v] \cdot u$ que satisfait le sous terme s.

(Application) Puis, si $t = \langle r\rangle R$, alors il faut montrer que $\left\langle \left(\frac{\partial}{\partial x} r \cdot u\right)[y/v] \right\rangle R[y/v] + \langle r[y/v]\rangle \left(\frac{\partial}{\partial x} R \cdot u\right)[y/v] = \frac{\partial}{\partial x}\left(\langle r[y/z]\rangle R[y/v]\right) \cdot u$. On l'obtient grâce aux hypothèses d'induction sur r et R.

(Paquets) Le raisonnement est similaire à celui qui vient d'être fait pour l'*application* dans le cas des constructions produit. Si $t = s^!$ alors il faut montrer que $\left(\frac{\partial}{\partial x} s \cdot u\right)[y/v]\left(s[v/y]\right)^! = \left(\frac{\partial}{\partial x} s[v/y] \cdot u\right)\left(s[v/y]\right)^!$. C'est immédiat grâce à l'hypothèse d'induction sur le sous terme s. □

Réduction atomique

La substitution linéaire nous permet de définir une réduction à petits pas dite *atomique*, qui est engendrée par les réductions suivantes sur les termes simples :

$$\langle \lambda x\, s\rangle 1 \ \xrightarrow{a} \ s[{}^{0}\!/_{x}]$$

$$\langle \lambda x\, s\rangle r R \ \xrightarrow{a} \ \left\langle \lambda x \frac{\partial}{\partial x} s \cdot r \right\rangle R$$

$$\langle \lambda x\, s\rangle r^! R \ \xrightarrow{a} \ \langle \lambda x\, s\rangle R + \left\langle \lambda x \frac{\partial}{\partial x} s \cdot r \right\rangle r^! R$$

Nous allons préciser l'intuition derrière cette dernière règle. Présent dans le rédex, $r^!$ est un paquet qui peut fournir un nombre arbitraire de ressources r. Les possibilités qui s'offrent à nous peuvent être résumées en un seul choix : n'utiliser aucune ressource r (d'où le premier terme dans l'expression du réduit) ou en utiliser au moins une (et cela correspond au deuxième terme). Dit autrement, la réduction est inspirée de la similarité $t^! \approx 1 + t t^!$.

Cette réduction ne termine pas. Par exemple, le terme $\left\langle \lambda x \langle y\rangle x^! \right\rangle z^!$ admet pour réduction infinie :

$$\begin{array}{cc} & \left\langle \lambda x \langle y\rangle x^! \right\rangle z^! \\ \xrightarrow{a} & \langle y\rangle 1 + \left\langle \lambda x \langle y\rangle z x^! \right\rangle z^! \\ \xrightarrow{a} & \langle y\rangle 1 + \langle y\rangle z + \left\langle \lambda x \langle y\rangle z z x^! \right\rangle z^! \\ \xrightarrow{a} & \langle y\rangle 1 + \langle y\rangle z + \langle y\rangle z z + \left\langle \lambda x \langle y\rangle z z z x^! \right\rangle z^! \\ & \vdots \end{array}$$

On reconnaît une forme de développement limité dans ce terme. En poursuivant cette réduction nous reconnaîtrions le développement de *Taylor–Ehrhard* (qualitatif) normalisé de notre terme de départ [ER06a].

Réduction atomique parallèle

On définit également la réduction parallèle $\overset{a}{\Longrightarrow}$, qui autorise à réduire simultanément plusieurs rédex qui sont visibles :

$$\frac{}{x \overset{a}{\Longrightarrow} x} \qquad \frac{s \overset{a}{\Longrightarrow} s^*}{\lambda x\, s \overset{a}{\Longrightarrow} \lambda x\, s^*} \qquad \frac{r \overset{a}{\Longrightarrow} r^* \quad R \overset{a}{\Longrightarrow} R^*}{\langle r\rangle R \overset{a}{\Longrightarrow} \langle r^*\rangle R^*}$$

les cas correspondant à des applications qui sont des rédex peuvent également être traités comme suit :

$$\frac{s \overset{a}{\Longrightarrow} s^*}{\langle \lambda x\, s\rangle 1 \overset{a}{\Longrightarrow} s^*[0/x]} \qquad \frac{s \overset{a}{\Longrightarrow} s^* \quad r \overset{a}{\Longrightarrow} r^* \quad R \overset{a}{\Longrightarrow} R^*}{\langle \lambda x\, s\rangle r R \overset{a}{\Longrightarrow} \left\langle \lambda x \frac{\partial}{\partial x} s^* \cdot r^* \right\rangle R^*}$$

$$\frac{s \overset{a}{\Longrightarrow} s^* \quad r \overset{a}{\Longrightarrow} r^* \quad R \overset{a}{\Longrightarrow} R^*}{\langle \lambda x\, s\rangle r^! R \overset{a}{\Longrightarrow} \langle \lambda x\, s^*\rangle R^* + \left\langle \lambda x \frac{\partial}{\partial x} s^* \cdot r^* \right\rangle {r^*}^! R^*}$$

Lorsqu'on réduit un paquet, tous ses termes peuvent être réduits en même temps. La réduction des sommes peut également être effectuée de façon simultanée :

$$\frac{t_1 \Longrightarrow t_1' \quad t_2 \Longrightarrow t_2'}{t_1 + t_2 \Longrightarrow t_1' + t_2'} \qquad \frac{}{0 \Longrightarrow 0}$$

Propriété 8-4. *La réduction* $\overset{a}{\Longrightarrow}$ *est encadrée ainsi :*

$$\xrightarrow{a} \;\subseteq\; \overset{a}{\Longrightarrow} \;\subseteq\; \overset{a}{\twoheadrightarrow}$$

Preuve. Montrer que $t \overset{a}{\Longrightarrow} t'$ implique $t \overset{a}{\twoheadrightarrow} t'$ se fait facilement par induction sur t. L'autre inclusion est vérifiée par définition après avoir remarqué que la réduction $\overset{a}{\Longrightarrow}$ est réflexive. □

Nous aurons besoin par la suite de réduire des termes définis à l'aide de substitutions. C'est pourquoi nous utiliserons fréquemment le lemme présenté ci-dessous dans les preuves de divers théorèmes.

Lemme 8-5. *Soient* t *et* u *deux termes tels que* $t \overset{a}{\Longrightarrow} t^*$ *et* $u \overset{a}{\Longrightarrow} u^*$*. On a :*

$$\frac{\partial}{\partial z} t \cdot u \;\overset{a}{\Longrightarrow}\; \frac{\partial}{\partial z} t^* \cdot u^*$$

Preuve. Le lemme se généralise aux paquets, et on procède par induction sur la dérivation de $t \overset{a}{\Longrightarrow} t^*$, les cas difficiles sont ceux où t est un rédex (il y en a trois) :

(i) Lorsque $t = \langle \lambda x\, s\rangle 1$ et $t^* = s^*[0/x]$, on a :

$$\frac{\partial}{\partial z} \left(\langle \lambda x\, s\rangle 1\right) \cdot u \;=\; \lambda x \left\langle \frac{\partial}{\partial z} s \cdot u \right\rangle 1$$

et, puisque $x \notin u^*$:

$$\frac{\partial}{\partial z} s^* \left[{}^{0}\!/_{x}\right] \cdot u^* \;=\; \left(\frac{\partial}{\partial z} s^* \cdot u^*\right)\left[{}^{0}\!/_{x}\right]$$

or par hypothèse d'induction $\frac{\partial}{\partial z} s \cdot u \overset{a}{\Longrightarrow} \frac{\partial}{\partial z} s^* \cdot u^*$, et le premier terme se réduit donc en le second.

(ii) Lorsque $t = \langle \lambda x\, s\rangle r R$ et $t^* = \left\langle \lambda x \frac{\partial}{\partial x} s^* \cdot r^* \right\rangle R^*$, on a :

$$\frac{\partial}{\partial z} \left(\langle \lambda x\, s\rangle r R\right) \cdot u \;=\; +\left[\begin{array}{l} \left\langle \lambda x \frac{\partial}{\partial z} s \cdot u \right\rangle r R \\ \langle \lambda x\, s\rangle \left(\frac{\partial}{\partial z} r \cdot u\right) R \\ \langle \lambda x\, s\rangle r \left(\frac{\partial}{\partial z} R \cdot u\right) \end{array}\right.$$

et

$$\frac{\partial}{\partial z} \left(\left\langle \lambda x \frac{\partial}{\partial x} s^* \cdot r^* \right\rangle R^*\right) \cdot u^* \;=$$

$$+\left[\begin{array}{l} \left\langle \lambda x \frac{\partial}{\partial x} \left(\frac{\partial}{\partial z} s^* \cdot u^*\right) \cdot r^* \right\rangle R^* \\ \left\langle \lambda x \frac{\partial}{\partial x} s^* \cdot \left(\frac{\partial}{\partial z} r^* \cdot u^*\right) \right\rangle R^* \\ \left\langle \lambda x \frac{\partial}{\partial x} s^* \cdot r^* \right\rangle \frac{\partial}{\partial z} R^* \cdot u^* \end{array}\right.$$

où on a utilisé la propriété 8-2 (*Schwarz*) dans le cas $x \notin u^*$. Les hypothèses d'induction permettent de montrer que chaque terme de la première somme se réduit en un terme de la deuxième.

(iii) Lorsque $t = \langle \lambda x\, s\rangle r^! R$ et $t^* = \langle \lambda x\, s^*\rangle R^* + \left\langle \lambda x \frac{\partial}{\partial x} s^* \cdot r^* \right\rangle r^{*!} R^*$, on a :

$$\frac{\partial}{\partial z} \left(\langle \lambda x\, s\rangle r^! R\right) \cdot u \;=\; +\left[\begin{array}{l} \left\langle \lambda x \frac{\partial}{\partial z} s \cdot u \right\rangle r^! R \\ \langle \lambda x\, s\rangle \left(\frac{\partial}{\partial z} r \cdot u\right) r^! R \\ \langle \lambda x\, s\rangle r^! \left(\frac{\partial}{\partial z} R \cdot u\right) \end{array}\right.$$

et

$$\frac{\partial}{\partial z} \left(\langle \lambda x\, s^*\rangle R^* + \left\langle \lambda x \frac{\partial}{\partial x} s^* \cdot r^* \right\rangle r^{*!} R^*\right) \cdot u^*$$

$$= \;+\left[\begin{array}{l} \left\langle \lambda x \frac{\partial}{\partial z} s^* \cdot u^* \right\rangle R^* \\ \langle \lambda x\, s^*\rangle \frac{\partial}{\partial z} R^* \cdot u^* \\ \left\langle \lambda x \frac{\partial}{\partial x} \left(\frac{\partial}{\partial z} s^* \cdot u^*\right) \cdot r^* \right\rangle r^{*!} R^* \\ \left\langle \lambda x \frac{\partial}{\partial x} s^* \cdot \left(\frac{\partial}{\partial z} r^* \cdot u^*\right) \right\rangle r^{*!} R^* \\ \left\langle \lambda x \frac{\partial}{\partial x} s^* \cdot r^* \right\rangle \left(\frac{\partial}{\partial z} r \cdot u^*\right) r^{*!} R^* \\ \left\langle \lambda x \frac{\partial}{\partial x} s^* \cdot r^* \right\rangle r^{*!} \left(\frac{\partial}{\partial z} R^* \cdot u^*\right) \end{array}\right.$$

une fois de plus, on vérifie qu'il y a réduction des termes de la première somme vers ceux de la deuxième. □

À l'aide de ce lemme, on prouve le théorème de confluence suivant :

Théorème 8-6. *La réduction atomique parallèle* $\overset{a}{\Longrightarrow}$ *est fortement confluente :*

$$\overset{a}{\Longleftarrow}\overset{a}{\Longrightarrow} \;\subseteq\; \overset{a}{\Longrightarrow}\overset{a}{\Longleftarrow}$$

Preuve. On doit montrer que pour tout termes t, t_1 et t_2, si $t \overset{a}{\Longrightarrow} t_1$ et $t \overset{a}{\Longrightarrow} t_2$, alors il existe un terme t' tel que $t_1 \overset{a}{\Longrightarrow} t'$ et $t_2 \overset{a}{\Longrightarrow} t'$. On procède par induction sur t ; il y a beaucoup de cas à considérer :

(Variable) Lorsque $t = x$, la seule possibilité est $t_1 = x$ et $t_2 = x$. On prend $t' = x$.

(Abstraction) Lorsque $t = \lambda x\, s$, les seules possibilités sont $t_1 = \lambda x\, s_1$ et $t_2 = \lambda x\, s_2$ avec $s \overset{a}{\Longrightarrow} s_1$ et $s \overset{a}{\Longrightarrow} s_2$. Par hypothèse d'induction, il existe s' tel que $s_1 \overset{a}{\Longrightarrow} s'$ et $s_2 \overset{a}{\Longrightarrow} s'$. On prend $t' = \lambda x\, s'$. Dans la suite on notera par l'expression plus compacte $s \overset{a}{\Longrightarrow} s_1/s_2 \overset{a}{\Longrightarrow} s'$ le diagramme de confluence de s_1 et s_2 formé par les quatre hypothèses dont on a hérité ici.

(Application) Lorsque $t = \langle r\rangle R$ n'est pas un rédex, on a nécessairement $t_1 = \langle r_1\rangle R_1$ et $t_2 = \langle r_2\rangle R_2$ avec $r \overset{a}{\Longrightarrow} r_1/r_2 \overset{a}{\Longrightarrow} r'$ et $R \overset{a}{\Longrightarrow} R_1/R_2 \overset{a}{\Longrightarrow} R'$. On prend alors $t' = \langle r'\rangle R'$.

(Rédex (pas de ressource)) Lorsque $t = \langle \lambda x\, s\rangle 1$, les seules possibilités sont ***(a)*** le cas où $t_1 = s_1[0/x]$ et $t_2 = s_2[0/x]$ avec $s \overset{a}{\Longrightarrow} s_1/s_2 \overset{a}{\Longrightarrow} s'$, et on prend $t' = s'[0/x]$; ***(b)*** le cas traité précédemment, ou bien ***(c)*** le cas mixte où $t_1 = s_1[0/x]$ et $t_2 = \langle s_2\rangle 1$ avec $s \overset{a}{\Longrightarrow} s_1/s_2 \overset{a}{\Longrightarrow} s'$, pour lequel on prend également $t' = s'[0/x]$. On oubliera dans la suite les cas ou on n'a pas réduit le rédex, ils ne posent pas de difficultés particulières.

(Rédex (ressource simple)) Il y a deux cas principaux selon que les deux réductions ont été effectué en privilégiant la même ressource, ou en privilégiant des ressources différentes. ***(a)*** Lorsque $t = \langle \lambda x\, s\rangle r R$, avec $s \overset{a}{\Longrightarrow} s_1/s_2 \overset{a}{\Longrightarrow} s'$, $r \overset{a}{\Longrightarrow} r_1/r_2 \overset{a}{\Longrightarrow} r'$, $R \overset{a}{\Longrightarrow} R_1/R_2 \overset{a}{\Longrightarrow} R'$, et $t_1 = \left\langle \lambda x \frac{\partial}{\partial x} s_1 \cdot r_1 \right\rangle R_1$, $t_2 = \left\langle \lambda x \frac{\partial}{\partial x} s_2 \cdot r_2 \right\rangle R_2$: on prend $t' = \left\langle \lambda x \frac{\partial}{\partial x} s' \cdot r' \right\rangle R'$. ***(b)*** Lorsque $t = \langle \lambda x\, s\rangle u v R$, avec $s \overset{a}{\Longrightarrow} s_1/s_2 \overset{a}{\Longrightarrow} s'$, $u \overset{a}{\Longrightarrow} u_1/u_2 \overset{a}{\Longrightarrow} u'$, $v \overset{a}{\Longrightarrow} v_1/v_2 \overset{a}{\Longrightarrow} v'$, $R \overset{a}{\Longrightarrow} R_1/R_2 \overset{a}{\Longrightarrow} R'$, et $t_1 = \left\langle \lambda x \frac{\partial}{\partial x} s_1 \cdot u_1 \right\rangle v_1 R_1$, $t_2 = \left\langle \lambda x \frac{\partial}{\partial x} s_2 \cdot v_2 \right\rangle u_2 R_2$: on prend $t' = \left\langle \lambda x \frac{\partial}{\partial x} \left(\frac{\partial}{\partial x} s' \cdot u'\right) \cdot v' \right\rangle R'$. Étant donné que $x \notin u', v'$ la propriété de *Schwarz* s'applique dans sa forme simple.

(Rédex (ressource promue)) Toujours deux cas principaux. ***(a)*** Lorsque $t = \langle \lambda x\, s\rangle r^!$, avec $s \overset{a}{\Longrightarrow} s_1/s_2 \overset{a}{\Longrightarrow} s'$, $r \overset{a}{\Longrightarrow} r_1/r_2 \overset{a}{\Longrightarrow} r'$, $R \overset{a}{\Longrightarrow} R_1/R_2 \overset{a}{\Longrightarrow} R'$, et $t_1 = \langle \lambda x\, s_1\rangle R_1 + \left\langle \lambda x \frac{\partial}{\partial x} s_1 \cdot r_1 \right\rangle {r_1}^! R_1$, $t_2 = \langle \lambda x\, s_1\rangle R_2 + \left\langle \lambda x \frac{\partial}{\partial x} s_2 \cdot r_2 \right\rangle {r_2}^! R_2$: on prend $t' = \langle \lambda x\, s'\rangle R' + \left\langle \lambda x \frac{\partial}{\partial x} s' \cdot r' \right\rangle R' + \left\langle \lambda x \frac{\partial}{\partial x} \left(\frac{\partial}{\partial x} s' \cdot r'\right) \cdot r' \right\rangle r'^! R'$. ***(b)*** Lorsque $t = \langle \lambda x\, s\rangle u^! v^! R$, avec $s \overset{a}{\Longrightarrow} s_1/s_2 \overset{a}{\Longrightarrow} s'$, $u \overset{a}{\Longrightarrow} u_1/u_2 \overset{a}{\Longrightarrow} u'$, $v \overset{a}{\Longrightarrow} v_1/v_2 \overset{a}{\Longrightarrow} v'$, $R \overset{a}{\Longrightarrow} R_1/R_2 \overset{a}{\Longrightarrow} R'$, et $t_1 = \langle \lambda x\, s_1\rangle {v_1}^! R_1 + \left\langle \lambda x \frac{\partial}{\partial x} s_1 \cdot u_1 \right\rangle {u_1}^! {v_1}^! R_1$, $t_2 = \langle \lambda x\, s_1\rangle {u_2}^! R_2 + \left\langle \lambda x \frac{\partial}{\partial x} s_2 \cdot v_2 \right\rangle {u_2}^! {v_2}^! R_2$: on prend $t' = \langle \lambda x\, s'\rangle R' + \left\langle \lambda x \frac{\partial}{\partial x} s' \cdot u' \right\rangle u'^! R' + \left\langle \lambda x \frac{\partial}{\partial x} s' \cdot v' \right\rangle v'^! R' + \left\langle \lambda x \frac{\partial}{\partial x} \left(\frac{\partial}{\partial x} s' \cdot u'\right) \cdot v' \right\rangle u'^! v'^! R'$.

(Rédex (ressource simple et ressource promue)) Lorsque $t = \langle \lambda x\, s\rangle u^! v R$, avec $s \overset{a}{\Longrightarrow} s_1/s_2 \overset{a}{\Longrightarrow} s'$, $u \overset{a}{\Longrightarrow} u_1/u_2 \overset{a}{\Longrightarrow} u'$, $v \overset{a}{\Longrightarrow} v_1/v_2 \overset{a}{\Longrightarrow} v'$, $R \overset{a}{\Longrightarrow} R_1/R_2 \overset{a}{\Longrightarrow} R'$, et $t_1 = \langle \lambda x\, s_1\rangle v_1 R_1 + \left\langle \lambda x \frac{\partial}{\partial x} s_1 \cdot u_1 \right\rangle {u_1}^! v_1 R_1$, $t_2 = \left\langle \lambda x \frac{\partial}{\partial x} s_2 \cdot v_2 \right\rangle {u_2}^! R_2$: on prend $t' = \left\langle \lambda x \frac{\partial}{\partial x} s' \cdot v' \right\rangle R' + \left\langle \lambda x \frac{\partial}{\partial x} \left(\frac{\partial}{\partial x} s' \cdot v'\right) \cdot u' \right\rangle u'^! R'$. □

À son tour, ce théorème nous informe sur la confluence de la réduction atomique non parallèle $\xrightarrow{a}$.

Corollaire 8-7. *La réduction atomique* $\xrightarrow{a}$ *est confluente :*

$$\overset{a}{\twoheadleftarrow}\overset{a}{\twoheadrightarrow} \;\subseteq\; \overset{a}{\twoheadrightarrow}\overset{a}{\twoheadleftarrow}$$

Preuve. La réduction parallèle $\overset{a}{\Longrightarrow}$ possède la propriété de confluence forte, et par conséquent la propriété de confluence globale. La propriété 8-4 fait que la réduction $\xrightarrow{a}$ hérite de cette confluence. □

8.1.3 Réduction groupée

Substitution par paquets

On considère une opération de substitution groupée d'une variable z par un paquet U dans un terme t notée $t\{U/z\}$, définie comme suit (cette définition ne fonctionne que pour $z \notin U$) :

$$t\left\{{}^{uU'}\!/_{z}\right\} \;=\; \left(\frac{\partial}{\partial z} t \cdot u\right)\left\{{}^{U'}\!/_{z}\right\}$$

$$x\left\{{}^{u^!}\!/_{z}\right\} \;=\; x \qquad z\left\{{}^{u^!}\!/_{z}\right\} \;=\; u$$

$$(\lambda x\, s)\left\{{}^{u^!}\!/_{z}\right\} \;=\; \lambda x\, s\left\{{}^{u^!}\!/_{z}\right\}$$

$$(\langle r\rangle R)\left\{{}^{u^!}\!/_{z}\right\} \;=\; \left\langle r\left\{{}^{u^!}\!/_{z}\right\}\right\rangle R\left\{{}^{u^!}\!/_{z}\right\}$$

sur les paquets cette opération de substitution étend la précédente de la façon suivante :

$$T\left\{{}^{uU'}\!/_{z}\right\} \;=\; \left(\frac{\partial}{\partial z} T \cdot u\right)\left\{{}^{U'}\!/_{z}\right\}$$

$$1\left\{{}^{u^!}\!/_{z}\right\} \;=\; 1 \qquad ST\left\{{}^{u^!}\!/_{z}\right\} \;=\; S\left\{{}^{u^!}\!/_{z}\right\}T\left\{{}^{u^!}\!/_{z}\right\}$$

$$t^!\left\{{}^{u^!}\!/_{z}\right\} \;=\; \left(t\left\{{}^{u^!}\!/_{z}\right\}\right)^!$$

La propriété 8-2 (*Schwarz*) assure que cette opération est correctement définie car, lorsque $U = u_1 u_2 U'$, les deux façons d'utiliser la définition donnent le même résultat :

$$\left(\frac{\partial}{\partial z} T \cdot u_1\right)\left\{{}^{u_2 U'}\!/_{z}\right\} \;=\; \left(\frac{\partial}{\partial z}\left(\frac{\partial}{\partial z} T \cdot u_1\right) \cdot u_2\right)\left\{{}^{U'}\!/_{z}\right\}$$

et de façon similaire :

$$\left(\frac{\partial}{\partial z} T \cdot u_2\right)\left\{{}^{u_1 U'}\!/_{z}\right\} \;=\; \left(\frac{\partial}{\partial z}\left(\frac{\partial}{\partial z} T \cdot u_2\right) \cdot u_1\right)\left\{{}^{U'}\!/_{z}\right\}$$

Réduction groupée

La substitution par paquets nous permet de définir la réduction groupée, qui s'exprime de la façon suivante sur les termes simples :

$$\langle \lambda x\, s\rangle R \;\xrightarrow{g}\; s\left\{{}^{R}\!/_{x}\right\}$$

Le terme $\langle \lambda x\, \langle y\rangle x^{!}\rangle z^{!}$ qui ne terminait pas avec la réduction atomique est maintenant réduit et normalisé lors d'une unique étape :

$$\langle \lambda x\, \langle y\rangle x^{!}\rangle z^{!} \;\xrightarrow{g}\; \langle y\rangle z^{!}$$

Rien ne nous empêche par contre de considérer un terme similaire au λ-terme δδ comme $\langle \lambda x\, \langle x\rangle x^{!}\rangle \left(\lambda x\, \langle x\rangle x^{!}\right)^{!}$, et on s'aperçoit qu'en l'absence de typage la terminaison de la réduction groupée n'est pas encore assurée.

Cette réduction groupée est la même que celle que décrit *Paolo Tranquilli* sous l'appellation « giant-step β-reduction ». En revanche, celle qui est nommée « baby-step β-reduction » est différente de la réduction atomique que nous avons vu précédemment : elle applique bien les ressources uniques séparément, mais elle ne décompose pas les ressources promues.

Réduction groupée parallèle

La réduction groupée parallèle $\xRightarrow{g}$ permet de réduire plusieurs rédex visibles dans un terme simultanément.

$$\frac{}{x \xRightarrow{g} x} \qquad \frac{s \xRightarrow{g} s^*}{\lambda x\, s \xRightarrow{g} \lambda x\, s^*} \qquad \frac{r \xRightarrow{g} r^* \quad R \xRightarrow{g} R^*}{\langle r\rangle R \xRightarrow{g} \langle r^*\rangle R^*}$$

telle que la réduction d'une application sous forme de rédex peut également être traitée comme suit :

$$\frac{s \xRightarrow{g} s^* \quad R \xRightarrow{g} R^*}{\langle \lambda x\, s\rangle R \xRightarrow{g} s^*\{R^*/x\}}$$

Les termes contenus dans les paquets ou les sommes sont réduits simultanément. Cette réduction a été définie de façon à être une parallélisée de la réduction groupée originelle.

Théorème 8-8. *La réduction groupée parallèle* $\xRightarrow{g}$ *est fortement confluente.*

$$\xLeftarrow{g}\xRightarrow{g} \;\subseteq\; \xRightarrow{g}\xLeftarrow{g}$$

Preuve. On doit montrer que pour tous termes t, t_1 et t_2, si $t \xRightarrow{g} t_1$ et $t \xRightarrow{g} t_2$, alors il existe un terme t' tel que $t_1 \xRightarrow{g} t'$ et $t_2 \xRightarrow{g} t'$. On procède par induction sur t. Les constructions qui n'impliquent pas de rédex ne posent pas de problème, on se référera à la preuve correspondante pour la réduction atomique. Lorsque t est un rédex, disons $t = \langle \lambda x\, s\rangle R$, il faut faire attention au cas où $t_1 = s_1\{R_1/x\}$ et $t_2 = s_2\{R_2/x\}$ avec par hypothèse d'induction $s \xRightarrow{a} s_1/s_2 \xRightarrow{a} s'$ et $R \xRightarrow{a} R_1/R_2 \xRightarrow{a} R'$. On prend $t' = s'\left\{R'/x\right\}$. Le cas mixte où $t_1 = s_1\{R_1/x\}$ et $t_2 = \langle \lambda x\, s_2\rangle R_2$ conflue en $t = s\{R/x\}$ en ne réduisant que les termes s_1 et R_1 dans le premier terme et en ne réduisant que le rédex dans le second. □

Corollaire 8-9. *La réduction groupée* $\xrightarrow{g}$ *est confluente.*

8.1.4 Rapports entre réduction groupée et réduction atomique

Une réduction groupée correspond potentiellement à une infinité de réductions atomiques. Le termes obtenus par réduction groupée ont l'avantage d'exprimer de façon finie ce que la réduction atomique ne pouvait exprimer sans décomposition infinie. Le développement de *Taylor–Ehrhard* effectué par la réduction atomique peut être ainsi évité.

Par chance, on remarque que les deux réductions peuvent coexister sans engendrer de problèmes de confluence.

Théorème 8-10. *Les réductions* $\xRightarrow{a}$ *et* $\xRightarrow{g}$ *sont fortement inter-confluentes :*

$$\xLeftarrow{a}\xRightarrow{g} \;\subseteq\; \xRightarrow{g}\xLeftarrow{a}$$

Preuve. On veut montrer que pour tous termes t, t_1 et t_2, si $t \xRightarrow{a} t_1$ et $t \xRightarrow{g} t_2$, alors il existe un terme t' tel que $t_1 \xRightarrow{g} t'$ et $t_2 \xRightarrow{a} t'$. Toujours par induction sur t. Les cas ou t n'est pas un rédex (ou n'est pas réduit par l'une des réductions en tenant compte de ce fait) se passent une fois de plus sans problème. Sinon, dans le cas ***(a)*** où $t = \langle \lambda x\, s\rangle 1$, on a $t_1 = s_1[0/x]$ et $t_2 = s_2[0/x]$ avec $s \xRightarrow{a} s_1 \xRightarrow{g} s'$ et $s \xRightarrow{g} s_2 \xRightarrow{a} s'$, on prend simplement $t' = s'[0/x]$. Dans le cas ***(b)*** où $t = \langle \lambda x\, s\rangle rR$, on a $t_1 = \left\langle \lambda x\, \frac{\partial}{\partial x} s_1 \cdot r_1 \right\rangle R_1$ et $t_2 = s_2\{r_2 R_2/x\}$ avec les hypothèses d'induction habituelles, on prend $t' = \left(\frac{\partial}{\partial x} s' \cdot r'\right)\left\{R'/x\right\} = s'\left\{r'R'/x\right\}$. Dans le cas ***(c)*** où $t = \langle \lambda x\, s\rangle r^{!}R$, on a $t_1 = \langle \lambda x\, s_1\rangle R_1 + \left\langle \lambda x\, \frac{\partial}{\partial x} s_1 \cdot r_1 \right\rangle r_1{}^{!} R_1$ et $t_2 = s_2\left\{r_2{}^{!}R_2\big/x\right\}$ avec les hypothèses d'induction habituelles, on prend $t' = s'\left\{R'/x\right\} + \left(\frac{\partial}{\partial x} s' \cdot r'\right)\left\{r'^{!}R'\big/x\right\} = s'\left\{r'^{!}R'\big/x\right\}$, pour montrer l'égalité on prend un R' générique de la forme $u_1 \cdots u_n u^{!}$ et on explicite les substitutions par leur définitions. □

Corollaire 8-11. *Les réductions* $\xrightarrow{a}$ *et* $\xrightarrow{g}$ *sont inter-confuentes :*

$$\overset{a}{\twoheadleftarrow}\overset{g}{\twoheadrightarrow} \;\subseteq\; \overset{g}{\twoheadrightarrow}\overset{a}{\twoheadleftarrow}$$

8.1.5 Calcul typé

La syntaxe des types simples utilisée reste la même en présence de promotion :

$$\tau \quad ::= \quad \alpha \;\mid\; \tau \to \tau$$

Le λ-calcul avec ressources et promotion typé $\vec{\Delta}_{!}$ s'obtient en restreignant la définition des termes de $\Delta_{!}$ à la grammaire ci-dessous, où $t^{\tau}_{\bar{w}}$ représente un terme t qui possède le type τ dans un environnement $\bar{w} = z_1{:}\omega_1, \ldots, z_n{:}\omega_k$:

$$\begin{aligned} t^{\tau}_{x:\tau,\bar{w}} &::= x \\ t^{\tau}_{\bar{w}} &::= \langle t^{\sigma\to\tau}_{\bar{w}}\rangle T^{\sigma}_{\bar{w}} \\ t^{\sigma\to\tau}_{\bar{w}} &::= \lambda x\, t^{\tau}_{x:\sigma,\bar{w}} \\ T^{\tau}_{\bar{w}} &::= 1 \;\mid\; T^{\sigma}_{\bar{w}} T^{\sigma}_{\bar{w}} \;\mid\; t^{\sigma}_{\bar{w}} \;\mid\; \left(t^{\sigma}_{\bar{w}}\right)^{!} \end{aligned}$$

Comme précédemment on définit l'ensemble des termes typables $\bar{\Delta}_{!}$ et $\vec{\Delta}_{!}$ l'ensemble des termes typés. Les types sont toujours conservés par réduction.

Propriété 8-12. *Les réductions atomiques* $\xrightarrow{a}$ *et groupée* $\xrightarrow{g}$ *sont compatibles avec les types. Lorsque un terme* t *peut être typé* $t^{\tau}_{\bar{w}}$*, dans toute réduction* $t \xrightarrow{a/g} s$*, le réduit* s *peut être typé de la même façon :* $s^{\tau}_{\bar{w}}$*.*

Preuve. On utilise la même propriété que dans la preuve déjà donnée pour le calcul sans promotion (voir la propriété 7-6) dans le cas de la réduction atomique. Pour la réduction groupée on remarque de la même façon que si t et U peuvent être typés $t^{\tau}_{x:\sigma}$ et U^{σ}, alors $t' = t\{U/x\}$ peut être typé t'^{τ}. □

On ne peut espérer que la réduction $\xrightarrow{a}$ termine, même sur l'ensemble restreint des termes typables (η en fait partie et ne normalise pas). En revanche nous allons pouvoir montrer la terminaison de la réduction $\xrightarrow{g}$ dans le cas typé (δδ n'en faisant pas partie). Dans un premier temps, nous allons montrer la normalisation faible de cette réduction par un procédé combinatoire similaire à celui qu'on peut utiliser pour le λ-calcul ordinaire (méthode *Tait–Martin-Löf*).

Théorème 8-13. *La réduction groupée* $\xrightarrow{g}$ *des termes typés normalise faiblement.*

Preuve. La réduction groupée parallèle forcée $\overset{g}{\Longmapsto}$ (incluse dans $\xRightarrow{g}$, on force la réduction de tout rédex visible) d'un terme t réduit strictement le maximum de la taille des types associés aux rédex de t. Tout rédex de type $\sigma \to \tau$ est de la forme $\langle \lambda x\, s^{\tau}_{x:\sigma}\rangle T^{\sigma}$; chacun est réduit en un terme de type τ de la forme $s^{*\tau}_{x:\sigma}\{T^{*\sigma}/x\}$ où le maximum de la taille des types associés aux rédex de s^* et T^* sont strictement inférieurs à ceux de s et T. Puis on fait la remarque qu'un terme normal pour $\overset{g}{\Longmapsto}$ est normal pour $\xrightarrow{g}$. □

8.1.6 Codage du λ-calcul différentiel

On remarquera que le λ-calcul différentiel [ER04 ; Vau07a], ainsi que ses réductions, peut être codé dans le λ-calcul avec ressources et promotion muni de la réduction groupée (on donne ici une traduction à partir de la syntaxe de *Lionel Vaux*) :

$$[\![x]\!] \;:=\; x$$

$$[\![\lambda x\, t]\!] \;:=\; \lambda x\, [\![t]\!] \qquad [\![(s)t]\!] \;:=\; \langle [\![s]\!]\rangle [\![t]\!]^{!}$$

$$[\![D\, t \cdot u]\!] \;:=\; \lambda z\, \langle [\![t]\!]\rangle [\![u]\!] z^{!}$$

La β-réduction est bissimulée de façon triviale par la réduction groupée ; la réduction différentielle aussi ! En effet, la règle D $(\lambda x\, s)\cdot u \longrightarrow \lambda x \frac{\partial}{\partial x} s \cdot u$ devient :

$$\lambda z \,\langle \lambda x \,[\![s]\!]\rangle\, [\![u]\!] z^! \;\xrightarrow{g}\; \lambda x \frac{\partial}{\partial x} [\![s]\!] \cdot [\![u]\!]$$

Cela dit, on peut également s'intéresser à ce codage, mais en considérant la réduction atomique. Il n'y a plus bissimulation (pour les même raisons qui font que le λ-calcul ordinaire ne peut être codé avec ce genre de réduction). On notera cependant, qu'en un certain sens, cette réduction décompose la β-réduction mais également la réduction différentielle. En effet, on peut choisir une ressource z pour effectuer la réduction dans le cas précédent :

$$\begin{gathered}\lambda z \,\langle \lambda x \,[\![s]\!]\rangle\, [\![u]\!] z^! \;\xrightarrow{a}\\ \lambda z \,\langle \lambda x \,[\![s]\!]\rangle\, [\![u]\!] \;+\; \lambda z \left\langle \lambda x \frac{\partial}{\partial x} [\![s]\!] \cdot z \right\rangle [\![u]\!] z^!\end{gathered}$$

et on obtient des termes qui n'ont pas d'équivoque dans le λ-calcul différentiel.

8.1.7 Réduction à la *Krivine*

Nous avons jusqu'à présent substitué des ressources dans un terme selon leur disponibilité. Nous allons maintenant les substituer selon la demande. On va s'intéresser à une réduction $\xrightarrow{k}$ dont le principe peut être illustré par un exemple :

$$\langle \lambda x \,\langle x\rangle x\rangle u v^! \;\xrightarrow{k}\; \langle \lambda x\, \langle u\rangle x\rangle v^! + \langle \lambda x\, \langle v\rangle x\rangle u v^!$$

Ici, on remarque que la variable x est active dans le terme $\lambda x \,\langle x\rangle x$ à la position matérialisée par le symbole $\square$ dans le contexte $\lambda x \,\langle \square\rangle x$. On substitue alors cette occurrence par l'une des ressources disponibles.

Rédex généralisé

La définition formelle de cette réduction nécessite l'introduction de la notion de *contexte applicatif* :

$$c \;\;::=\;\; \square \;\mid\; \lambda y\, c \;\mid\; \langle c\rangle T$$

Une occurrence de variable x sera dite en *position de tête* d'un terme t lorsque ce dernier s'écrit $t = c[x]$, pour un certain contexte applicatif c. Un terme possède une unique occurrence de variable en position de tête.

Nous pourrions ne nous intéresser qu'à cette occurrence de tête, mais nous souhaitons un peu plus de généralité. Nous allons pour cela considérer les termes à σ-équivalence près [Reg94].

Définition 8-14. *On définit la σ-équivalence sur les λ-termes avec ressources (et promotion), notée $\equiv_\sigma$, comme la congruence engendrée par les équivalences élémentaires suivantes :*

$$\begin{aligned}\langle\langle \lambda x\, s\rangle U\rangle T \;&\equiv_\sigma\; \langle \lambda x\,\langle s\rangle T\rangle U\\ \lambda y\,\langle \lambda x\, t\rangle U \;&\equiv_\sigma\; \langle \lambda x\, \lambda y\, t\rangle U\end{aligned}$$

toujours avec la convention qui impose que les variables liées ne sont jamais réutilisées à l'extérieur de leur lieur.

C'est une astuce qui permet d'apparier simplement une occurrence d'une variable x d'un terme t avec l'éventuel paquet R qui contient les ressources destinées à lui être substitué. Si ce paquet est présent dans t, on pourra par σ-équivalence trouver un représentant de t qui contient un rédex de la forme $\langle \lambda x\, c[x]\rangle R$, on dit alors que l'occurrence de x en question est en *position active.*

Syntaxiquement, la σ-équivalence est une transformation linéaire (i.e. dans sa définition toute méta-variable apparaît une et une seule fois à gauche et à droite des équivalences qui l'engendrent). La notion d'occurrence de variable a donc un sens même lorsqu'on considère les termes à σ-équivalence près. On note également que les contextes applicatifs restent des contextes applicatifs par σ-équivalence. Leur structure impose le fait qu'une seule une occurrence d'une même variable peut être en position active.

Extraction d'une ressource

Pour définir formellement notre réduction nous avons besoin d'une nouvelle forme de substitution. Les réductions précédentes utilisaient la substitution linéaire. Cette opération consistait à placer une ressource particulière à un emplacement désigné par une occurrence quelconque d'une variable donnée. Une somme était faite sur l'ensemble des occurrences possibles. Nous souhaitons définir une opération duale : une occurrence particulière d'une variable est choisie, on souhaite la remplacer par une ressource quelconque d'un paquet donné. La somme sera faite cette fois sur l'ensemble des ressources possibles.

Pour définir formellement cette opération, nous allons considérer des contextes (au sens habituel, pas applicatif) à deux emplacements dont le premier emplacement attend un terme et le deuxième un paquet. L'exemple le plus simple est le suivant :

$$t[\square_1, \square_2] \;=\; \langle \square_1\rangle \square_2$$

Pour éviter les problèmes de capture de variable on supposera comme toujours que les variables liées ne sont pas réutilisées à l'extérieur de leurs lieurs.

On notera $\int_R t[dU, U]$ l'opération qui consiste à sommer sur toutes les possibilités d'extraire une ressource atomique d'un paquet R, et placer à chaque fois dans le contexte t la ressource sélectionnée à l'emplacement désigné par dU et les ressources restantes à l'emplacement désigné par U. Les symboles dU et U sont des variables muettes liées au signe intégral qui permettent facilement de désigner les emplacements du contexte t qui nous intéressent. Attention, dU prend la place occupée habituellement par un terme et pas par un paquet.

Définition 8-15. *L'opération d'extraction linéaire $\int_R t[dU, U]$ est définie par induction sur R :*

$$\begin{gathered}\int_1 t[dU, U] \;=\; 0\\ \int_{ST} t[dU, U] \;=\; \int_S t[dU, UT] + \int_T t[dU, SU]\\ \int_r t[dU, U] \;=\; t[r, 1] \qquad \int_{r^!} t[dU, U] \;=\; t\left[r, r^!\right]\end{gathered}$$

Illustrons cela grâce au contexte que nous avons pris pour exemple. Nous souhaitons le remplir avec les ressources d'un paquet contenant exactement trois ressources u, v, w. Cela s'écrit :

$$\int_{uvw} \langle dU\rangle U \;=\; \langle u\rangle vw + \langle v\rangle uw + \langle w\rangle uv$$

Pour cette nouvelle opération, on obtient une propriété duale à celle de *Schwarz*, qui s'apparente à celle de *Fubini*. D'autres raisons peuvent justifier l'emploi du signe intégral pour cette notation, l'une d'elles sera donnée en 9.4.4.

Propriété 8-16. *Les opérateurs d'extraction linéaire commutent, on appellera cette propriété la propriété de Fubini. Lorsque $U_1 \notin R_2$ et $U_2 \notin R_1$, on a :*

$$\begin{gathered}\int_{R_1}\int_{R_2} t[dU_1, U_1, dU_2, U_2]\\ =\\ \int_{R_2}\int_{R_1} t[dU_1, U_1, dU_2, U_2]\end{gathered}$$

Dynamique

On peut enfin définir la réduction qui nous intéresse. Comme pour les réductions précédentes, on va réduire un terme après avoir identifié une variable x et un paquet R qui se correspondent. Les réductions précédentes plaçaient une ressource choisie dans le paquet R à un emplacement désigné par une occurrence quelconque de la variable x. La substitution linéaire utilisée pour cela faisant la somme sur l'ensemble des occurrences de x qu'on peut choisir. La nouvelle réduction utilise une approche duale, elle va s'intéresser à une occurrence particulière de la variable x, et la remplacer par une ressource quelconque du paquet R. L'opération d'extraction linéaire va être utilisée pour cela, elle nous permet de faire une somme sur l'ensemble des ressources de R qu'on peut choisir.

Définition 8-17. *La réduction à la Krivine est la clôture contextuelle de la réduction suivante sur les λ-termes avec ressources simples considérés à σ-équivalence près :*

$$\langle \lambda x\, c[x]\rangle R \;\xrightarrow{k}\; \int_R \langle \lambda x\, c[dU]\rangle U$$

le nom c désigne ici un contexte applicatif.

Cette réduction est déterministe sur un rédex donné. À la différence des réductions précédentes où on laissait libre le choix de la ressource utilisée pour la substitution, ici on fixe l'occurence de variable à substituer en choisissant celle qui est active. Laisser libre le choix de la variable à substituer en utilisant un contexte c quelconque aurait également un sens.

Remarque 8-1. *La réduction opérée dans une machine de Krivine [Kri08 ; ER06a], dite* réduction linéaire de tête, *s'exprime de façon similaire sur les* λ*-termes ordinaires si on les considère à* σ*-équivalence près. C'est l'instance de la règle suivante qui opère systématiquement sur l'occurrence de variable qui est en position de tête si celle-ci est active dans le terme qu'il faut réduire :*

$$(\lambda x\, c[x])\, t \longrightarrow (\lambda x\, c[t])\, t$$

Cette règle correspond à notre réduction lorsque R *est comme tous les paquets, de la forme* $t^!$.

Plusieurs occurrences de variables peuvent être en position active, or pour chaque occurrence de variable en position active dans un terme t, il existe une réduction de t associée. Si nous avions défini une telle réduction dans le λ-calcul avec ressources sans promotion, en l'absence de duplication, on pourrait montrer que cette réduction est fortement confluente. En présence de promotion on va tout de même montrer qu'elle est confluente.

La réduction que nous venons de définir est intéressante car elle est atomique (dans le sens où elle transmet les ressources une à une) et possède tout de même une propriété de normalisation en présence de promotion (pour les termes typés).

Réduction parallèle

Ici, une fois de plus on peut obtenir un théorème de confluence en passant par une réduction à la *Krivine* parallèle $\xRightarrow{k}$. La réduction d'un terme qui peut être écrit sous forme de rédex (par σ-équivalence) est traitée comme suit (on étend naturellement la réduction aux contextes) :

$$\frac{c \xRightarrow{k} c^* \quad R \xRightarrow{k} R^*}{\langle \lambda x\, c[x]\rangle R \xRightarrow{k} \int_{R^*} \langle \lambda x\, c^*[dU]\rangle U}$$

en complément des règles habituelles :

$$\frac{}{x \xRightarrow{k} x} \qquad \frac{s \xRightarrow{k} s^*}{\lambda x\, s \xRightarrow{k} \lambda x\, s^*} \qquad \frac{r \xRightarrow{k} r^* \quad R \xRightarrow{k} R^*}{\langle r\rangle R \xRightarrow{k} \langle r^*\rangle R^*}$$

On vérifie que cette définition ne dépend pas du représentant choisi.

Lemme 8-18. *Soient* t *un contexte tel que* $t \xRightarrow{k} t^*$ *et* R *un paquet tel que* $R \xRightarrow{k} R^*$*. On a :*

$$\int_R t[dU, U] \xRightarrow{k} \int_{R^*} t^*[dU, U]$$

Preuve. Par induction sur R. Le cas $R = 1$ est évident. Le cas $R = ST$ s'obtient directement avec les hypothèses d'induction. Les cas $R = r$ et $R = r^!$ s'obtiennent par réduction parallèle d'un contexte : si $t \xRightarrow{k} t^*$, $r \xRightarrow{k} u^*$, et $S \xRightarrow{k} S^*$ alors $t[r, S] \xRightarrow{k} t^*[r^*, S^*]$. □

Avec ce lemme on prouve le théorème de confluence attendu.

Théorème 8-19. *La réduction* $\xRightarrow{k}$ *est fortement confluente :*

$$\xLeftarrow{k}\xRightarrow{k} \;\subseteq\; \xRightarrow{k}\xLeftarrow{k}$$

Preuve. On veut montrer que pour tous termes t, t_1 et t_2, si $t \xRightarrow{k} t_1$ et $t \xRightarrow{k} t_2$, alors il existe un terme t' tel que $t_1 \xRightarrow{k} t'$ et $t_2 \xRightarrow{k} t'$. On procède par récurrence sur la taille d'un représentant de t (ils ont tous la même taille). Une fois de plus, les constructions simples ne posent pas de problème, on se référera aux preuves déjà établies. Lorsque t s'écrit sous la forme d'un rédex, disons $t = \langle \lambda x\, c[x]\rangle R$, il faut faire attention au cas où $t_1 = \int_{R_1} \langle \lambda x\, c_1[dU]\rangle U$ et $t_2 = \int_{R_2} \langle \lambda x\, c_2[dU]\rangle U$ avec par hypothèse de récurrence $c \xRightarrow{k} c_1/c_2 \xRightarrow{k} c'$ et $R \xRightarrow{k} R_1/R_2 \xRightarrow{k} R'$. Grâce au lemme on sait que $t' = \int_{R'} \langle \lambda x\, c'[dU]\rangle U$ fait l'affaire. □

Corollaire 8-20. *La réduction* $\xrightarrow{k}$ *est confluente :*

$$\overset{k}{\twoheadleftarrow}\overset{k}{\twoheadrightarrow} \;\subseteq\; \overset{k}{\twoheadrightarrow}\overset{k}{\twoheadleftarrow}$$

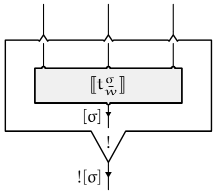
On pense également que cette réduction possède une propriété d'inter-confuence avec les deux autres, la réduction atomique et la réduction groupée.

8.2 Réseaux différentiels et boîtes exponentielles

Nous souhaitons présenter ici le système qui correspond aux réseaux de preuve d'une logique linéaire différentielle complétée. C'est-à-dire combinant *promotion* et opérateurs différentiels.

C'est un système qui permet d'exprimer le λ-calcul avec ressources et promotion. En effet, la traduction du λ-calcul avec ressources donnée en 7.3.1 s'étend tout simplement aux promotions de la manière suivante (on change légèrement de notation graphique pour les boîtes exponentielles, la raison sera expliquée plus tard) :

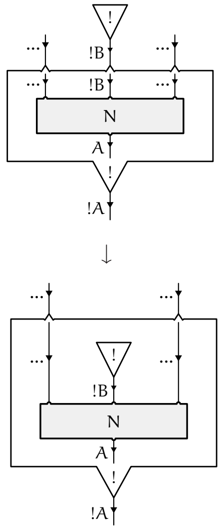
$$[\![(t^{\sigma}_{w})^{!}]\!] :=$$



8.2.1 Réductions des boîtes face aux opérateurs différentiels

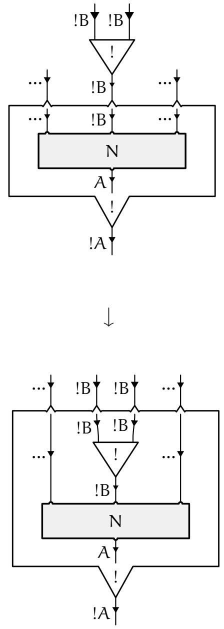
Se rajoutent aux règles de réduction déjà rencontrées en 3.1.2 pour les boîtes, et en 7.2.2 pour ce qui concerne les opérateurs différentiels, les trois règles présentées ici. Ces règles sont déclenchées par l'arrivée d'un opérateur différentiel devant l'entrée d'une boîte exponentielle.

Entrée-Promotion — Co-affaiblissement Un *co-affaiblissement* arrivant sur l'entrée d'une boîte rentre dans cette boîte.



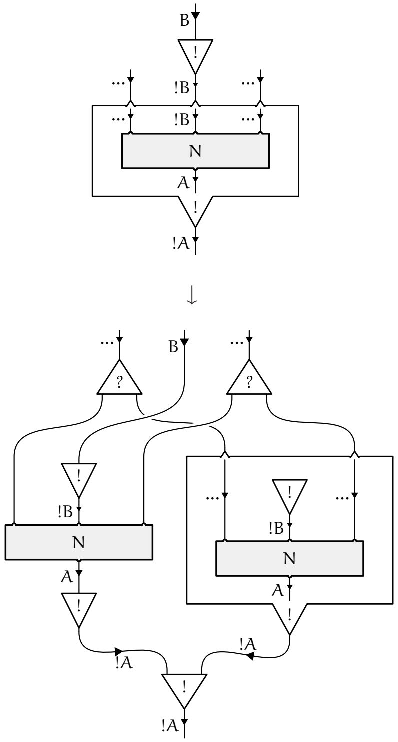
En termes de ressources, cela correspond au fait qu'une usine n'ayant accès à aucune ressource pour un certain type d'approvisionnement ne peut générer que des produits n'utilisant aucune ressource de ce type.

Entrée-Promotion — Co-contraction Une *co-contraction* arrivant sur l'entrée d'une boîte rentre dans cette boîte.



Cela correspond au fait qu'une usine ayant deux sous-traitants qui la fournissent en ressources au travers d'un même lien d'approvisionnement peut générer des produits qui accéderont eux-mêmes aux ressources de ces deux fournisseurs (à la manière de services).

Entrée-Promotion — Co-déréliction Lorsqu'une *co-déréliction* rencontre l'entrée d'une boîte, elle réagit selon la règle de la chaîne.

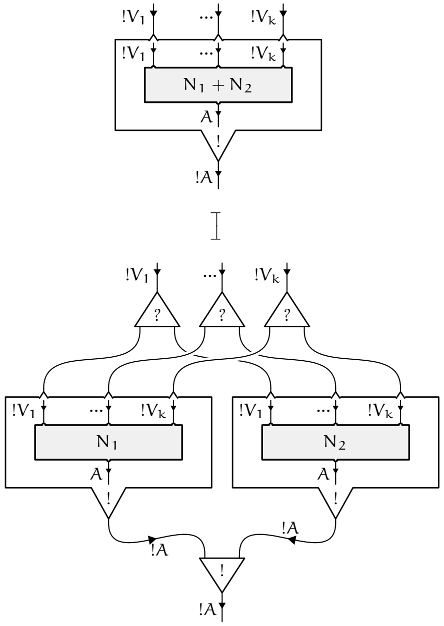


L'interprétation est ici qu'une usine que l'on fournit avec une ressource atomique unique, pour un certain type d'approvisionnement (ici B), doit générer exactement un produit utilisant cette ressource, et éventuellement d'autres produits n'utilisant pas ce type de ressource.

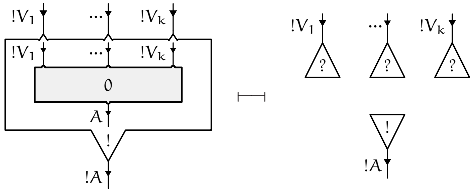
8.2.2 Boîtes et somme

Il est important de remarquer que chaque boîte est paramétrée par un élément du monoïde considéré. Les sommes générées par les interactions différentielles n'ont pas une portée globale, mais une portée restreinte à la première boite qui les contient.

Une boîte exponentielle contenant la somme de deux réseaux est cependant remarquablement équivalente à un réseau simple :



Et il en est de même pour une boîte exponentielle contenant une somme nulle de réseaux :



On remarque aussi que ces équivalences pourraient être orientées (vers la droite) dans le but d'être intégrées au processus de réduction. La réduction préserverait alors sa propriété de confluence.

8.2.3 Propriétés

Ce système est localement confluent modulo les équivalences données en 3.1.4. Une preuve de normalisation a été formulée par *Paolo Tranquilli* dans un cadre intuitionniste. Une preuve par réalisabilité de la normalisation faible de ce système est donnée en 11.4.

8.3 Interprétation des réductions du λ-calcul avec ressources et promotion

Nous avons présenté en 7.3.2 une simulation du λ-calcul avec ressources dans le système des les réseaux différentiels. Nous n'avons pas écrit la preuve, mais des simulations du même genre existent vraisemblablement pour les réductions $\xrightarrow{a}$, $\xrightarrow{g}$ et $\xrightarrow{k}$ du λ-calcul avec ressources et promotion dans les réseaux différentiels avec boîtes de promotion. À chacune de ces réductions correspond une stratégie de réduction particulière dans les réseaux.

Chapitre 9

Super-promotion et réplication

La *super-promotion* est une nouvelle construction qui permet l'effacement et la duplication de données. Contrairement à la *promotion*, elle conserve la symétrie des modalités qui est présente dans la logique différentielle de base. Elle semble incontournable pour étendre la traduction [EL07] du π-calcul finitaire vers les réseaux d'interaction différentiels, qui a été proposée par *Thomas Ehrhard* et *Olivier Laurent*, à la réplication. En contrepartie, avec cette généralisation certaines réductions ne terminent pas.

On va dans un premier temps introduire directement la règle *super-promotion*. Elle est proche de l'intuition que l'on peut trouver dans le π-calcul. On peut également la faire apparaître dans un λ-calcul avec ressource. La règle alternative dite *réplication* sera présentée ensuite. Elle est dans un certain sens plus atomique, et le système de réduction associé est plus satisfaisant dans les réseaux pour cette variante fonctorielle.

Le critère de correction pour ces nouvelles constructions n'est devenu clair qu'après la considération d'un typage plus expressif que celui que nous avons considéré jusqu'alors pour les réseaux. Ce formalisme ne sera présenté que dans le chapitre suivant, il repose sur une présentation de la logique linéaire différentielle dans un calcul où les séquents sont remplacés par des structures plus flexibles.

9.1 Motivations

On adjoint souvent au π-calcul une construction de réplication gardée $!a.P$, voire une réplication plus générale $!P$, pour un processus P quelconque. Cette construction vérifie $!P \approx P \mid !P$. On notera la similarité avec la correspondance $t^! \approx 1 + tt^!$ citée lorsqu'on a parlé de λ-calcul avec ressources et promotion. Si on considère le π-calcul sous un angle « linéaire » où l'on souhaite que tous les processus soient consommés avant d'obtenir un résultat, l'équivalence structurelle qu'il faudrait adopter serait effectivement $!P \approx 1 + P \mid !P$ où même $!P \approx \Sigma_{n=0}^{+\infty} \frac{1}{n!}(P \mid \cdots \mid P)$ si on souhaite utiliser une interprétation quantitative.

Cependant, contrairement à la promotion qui agit sur une ressource unique, la réplication des processus du π-calcul se fait sur des processus qui contiennent plusieurs entités. Par exemple si $P = a.R \mid b.S$, l'expression développée de sa réplication devient :

$$!P \;\approx\; 1 + a.R \mid b.S + a.R \mid b.S \mid a.R \mid b.S + \cdots$$

Il n'était pas possible d'exprimer la promotion de deux ressources $(uv)^!$ dans le λ-calcul avec ressources. On se propose donc de remplacer la construction *promotion* par une *super-promotion* qui agit sur les paquets.

Définition 9-1. *La syntaxe du λ-calcul avec ressources et super-promotion s'exprime ainsi :*

$$\begin{array}{lcl} t & ::= & x \;\mid\; \lambda x\, t \;\mid\; \langle t\rangle T \\ & & \quad 0 \;\mid\; t+t \\[1ex] T & ::= & 1 \;\mid\; TT \;\mid\; t \;\mid\; T^! \\ & & \quad 0 \;\mid\; T+T \end{array}$$

La *promotion* permettait de former un paquet en répliquant un terme, la *super-promotion* crée elle un paquet en répliquant un paquet. Mais dans le λ-calcul la réduction devient quelque peu problématique. On pourrait en effet l'exprimer ainsi lorsqu'un paquet promu est présent dans un rédex :

$$\langle \lambda x\, s\rangle U^! R \;\xrightarrow{a}\; \langle \lambda x\, s\rangle R + \langle \lambda x\, s\rangle U U^! R$$

en utilisant comme précédemment une similarité, ici $U^! \approx 1 + UU^!$. Mais cette réduction « explose ». Contrairement à la réduction qu'on utilisait pour une promotion simple :

$$\langle \lambda x\, s\rangle r^! R \;\xrightarrow{a}\; \langle \lambda x\, s\rangle R + \left\langle \lambda x \,\frac{\partial}{\partial x}\, s \cdot r \right\rangle r^! R$$

l'absence de substitution linéaire fait qu'on peut poursuivre indéfiniment la réduction du terme à droite de la somme (dans lequel le rédex originel est réapparu), sans nécessairement obtenir un terme nul.

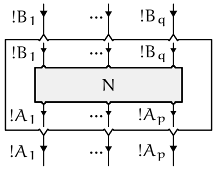
9.2 Super-promotion

Dans le monde différentiel, il est possible de retarder le moment où l'on se rend compte que le nombre de ressources qu'on possède ne permet pas de faire interagir chacune d'entre elle. Cette facilité fait qu'il est possible de généraliser la règle de *promotion*, on y parvient dans la syntaxe des réseaux.

9.2.1 Boîtes de super-promotion

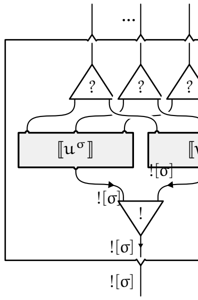
Les boîtes de *super-promotion* vont permettre d'isoler un morceau de réseau qui pourra être répliqué à volonté. Pour que cela ait un sens il faut que tout les liens accessibles soient sous une modalité exponentielle puisque la quantité de ressources transmises par ces liens peut varier.

Super-promotion



Jusqu'à présent on a utilisé une *promotion* qui jouait ce rôle en proposant une seule sortie sous la modalité ‹ ! › et un nombre variable d'entrées sous la modalité ‹ ? ›. Le nombre de ressources désiré sur l'unique sortie commandait le nombre de copies du contenu de la boîte qu'il faut générer, et en même temps le nombre de ressources qu'il faut demander sur les liens associés aux entrées. Dans le monde différentiel rien ne nous empêche de proposer plusieurs sorties de modalité ‹ ! › et de décorréler complètement le nombre de ressource demandé et le nombre de copies nécessaire. Tant pis si ce nombre de copies est différent lorsque l'on veut satisfaire indépendamment des demandes faite sur chacune des sorties. La dynamique du calcul différentiel fait en sorte qu'on s'en aperçoive plus tard, car la réduction terminera en fin de compte sur un réseau nul.

On peut en particulier exprimer la *super-promotion* telle qu'on a tenté de l'introduire dans le λ-calcul, et on parviendra cette fois à définir réduction satisfaisante. Par exemple la *super-promotion* d'un paquet de deux ressources $(uv)^!$ est écrite :



Sorties multiples et critère de validité

On ne perçoit pas encore l'intérêt d'avoir des boîtes munies de plusieurs sorties, c'est en fait la dynamique de la réduction (que nous décrirons juste après) qui nécessite cela. Cependant, à cause de cette généralisation forcée par la dynamique, nous allons devoir renoncer temporairement à toute notion de validité pour un réseau.

La règle du calcul des séquents que nous avons besoin d'ajouter pour coder le λ-calcul avec *super-promotion*, et qui correspond à des boîtes de *super-promotion* avec une seule sortie de type $!A$, est la suivante :

$$\frac{!\Gamma \vdash !A}{!\Gamma \vdash !A}\ \textit{super-promotion}$$

Et jusqu'à présent, le critère de validité le plus naturel pour un réseau était l'existence d'une forme séquentialisée de celui-ci dans le calcul des séquents. Cependant, dans le monde différentiel, la syntaxe des séquents n'est pas assez souple pour pouvoir exprimer une élimination des coupures. Cela pose déjà quelques difficultés en présence d'une simple *co-contraction*, c'est un véritable problème en présence de *super-promotion*.

La syntaxe des réseaux est elle très souple (et même trop souple car elle ne donne justement pas l'indication de validité recherchée). Nous donnerons les réductions de la *super-promotion* dans cette syntaxe, et nous nous contenterons dans un premier temps de justifications sémantiques, à défaut de justifications logiques. Nous introduirons en 10.2 un formalisme qui sera à la fois séquentiel et suffisamment souple pour exprimer la réduction. C'est ce formalisme qui nous donnera un critère de correction clair pour exprimer la validité d'un réseau. En attendant, on sera rassuré de savoir que lorsqu'un réseau est construit avec la construction donnée ici (qui permet d'introduire des boîtes à une seule sortie), il est valide au sens de ce futur critère. Et bien que les réductions que nous proposerons ci-devant sont susceptibles de faire apparaître des réseaux qui ne peuvent être construits avec cette construction restreinte, la véritable notion de validité sera elle préservée par réduction.

En particulier la règle du calcul des séquents à laquelle on peut penser pour introduire des boîtes à plusieurs sorties :

$$\frac{!\Gamma \vdash !\Delta}{!\Gamma \vdash !\Delta}\ (\textit{mauvaise-super-promotion})$$

est incorrecte vis-à-vis de la bonne notion de validité.

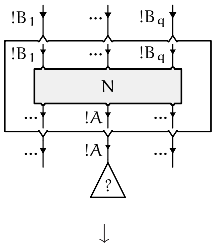
9.2.2 Réduction des *super-promotions*

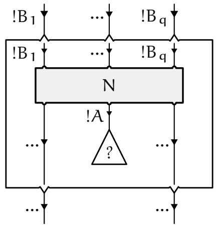
Comme dans le cas des autres constructions avec boîte, les ports externes d'une *super-promotion* seront tous considérés principaux.

Interactions structurelles

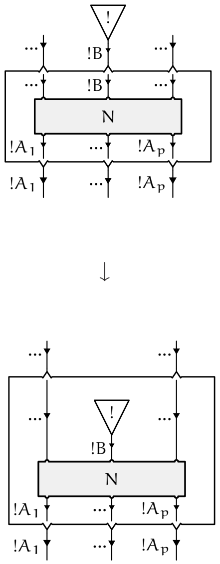
Un premier jeu de réductions vient compléter l'ensemble de réductions structurelles des réseaux différentiels.

Affaiblissement* — *Sortie-Super-Promotion Les opérateurs *affaiblissement* sont ingérés par les boîtes de *super-promotion*. Les boîtes produites par de telles interactions ont une sortie de moins que les boîtes d'origine.

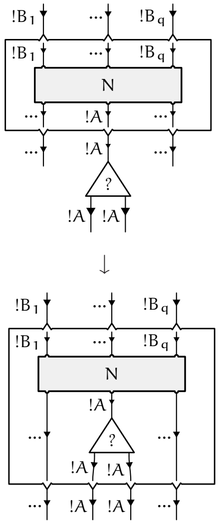




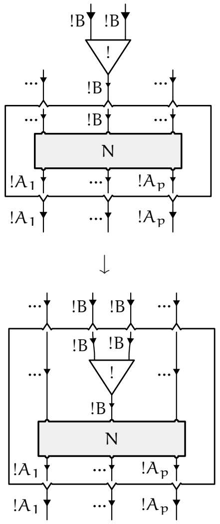
Entrée-Super-Promotion* — *Co-affaiblissement Cette règle est similaire à la règle précédente et à celle que l'on connaissait déjà pour la promotion. Les opérateurs *co-affaiblissement* sont ingérés par les boîtes de *super-promotion*.



Contraction* — *Sortie-Super-Promotion Les opérateurs *contraction* sont également ingérés par les boîtes de *super-promotion*. Les boîtes produites par de telles interactions ont une sortie de plus que les boîtes d'origine.



Entrée-Super-Promotion* — *Co-contraction Bien sûr, cette règle est de nouveau similaire à la règle précédente. Les opérateurs *co-contraction* sont ingérés par les boîtes de *super-promotion*.

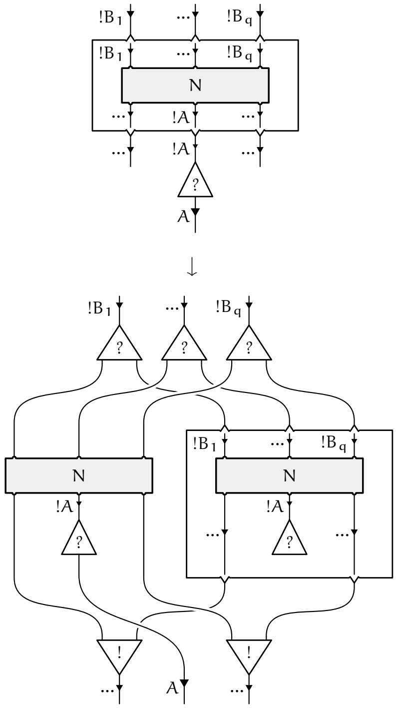


Notons que les règles qui modifient le nombre de sorties d'une boîtes sont celles qui génèrent potentiellement des réseaux qu'il n'est pas possible de construire directement selon les règles admises.

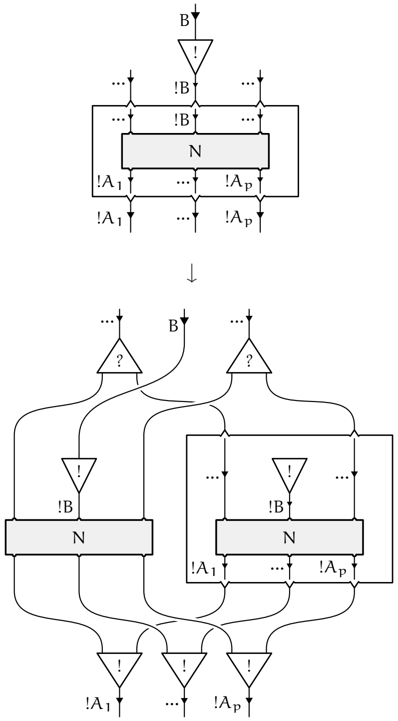
Instanciation

Dans les réseaux différentiels, lorsqu'une *déréliction* ou une *co-déréliction* rencontre un opérateur structurel d'*affaiblissement* ou de *contraction*, des réductions indéterministes ont lieu. Face à une boîte de *super-promotion* cela ne va pas être le cas, il suffira de distinguer une instance du contenu de la boîte. On va même pouvoir identifier dans les règles suivantes le noyau opérationnel qui était à l'origine de la règle de la chaîne (présentée dans le chapitre précédent, cf. 8.2.1).

Sortie-Super-Promotion* — *Déréliction La réduction d'une boîte de *super-promotion* face à une *déréliction* se fait de la façon décrite ci-dessous. Une certaine copie du contenu de la boîte va fournir l'unique ressource de type A demandée par l'environnement.



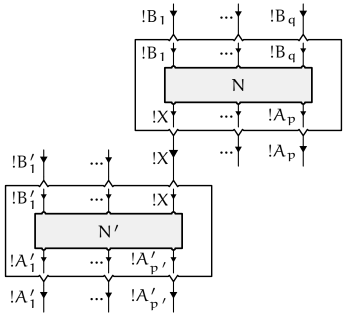
Entrée-Super-Promotion — Co-déréliction Cette règle est similaire à la règle précédente. Une certaine copie du contenu de la boîte va consommer l'unique ressource de type B fournie par l'environnement.



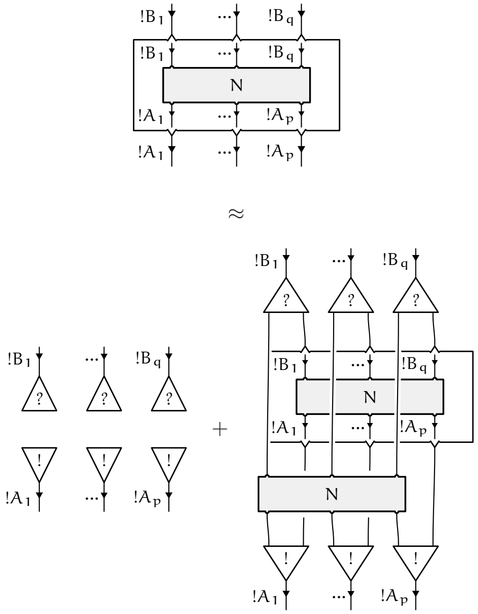
Inter-instanciation

Une dernière règle régit l'interaction entre deux boîtes. Cette règle est de nature complexe du point de vue computationnel et est à l'origine de la non terminaison du système de réduction.

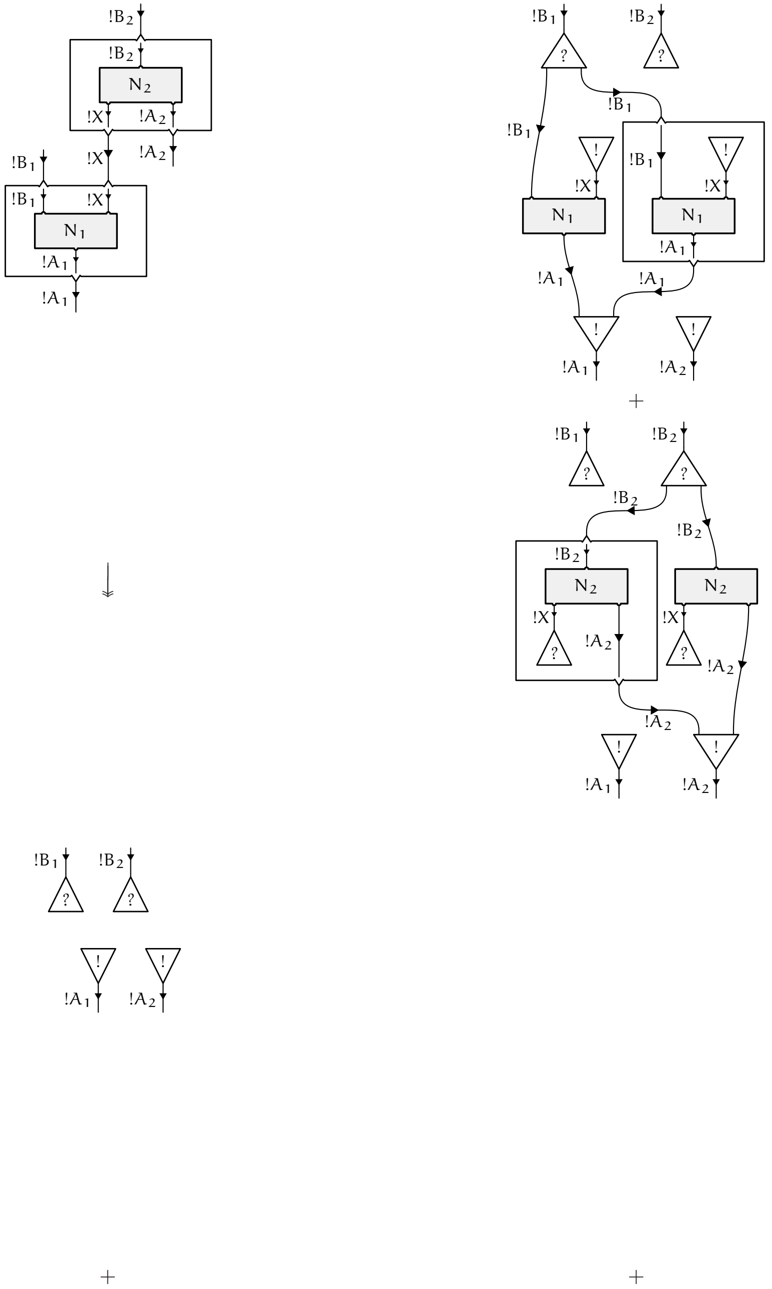
Entrée-Super-Promotion—Sortie-Super-Promotion Lorsque deux boîtes de *super-promotion* sont liées, elles peuvent interagir pas à pas après développements successif. En effet, lorsqu'un rédex tel que celui présenté ici :

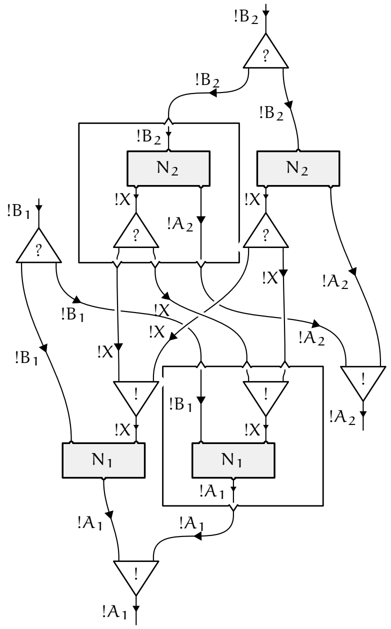


apparaît, on peut utiliser l'expansion suivante sur les deux boîtes pour deviner la règle de réduction correspondante :



Cette règle est celle qui réécrit le rédex originel en une seule étape vers la superposition de quatre réseaux obtenue après développement des deux boîtes et réduction. On donne ici cette règle pour des boîtes qui possèdent, outre la porte qui forme le rédex, une entrée et une sortie supplémentaires :



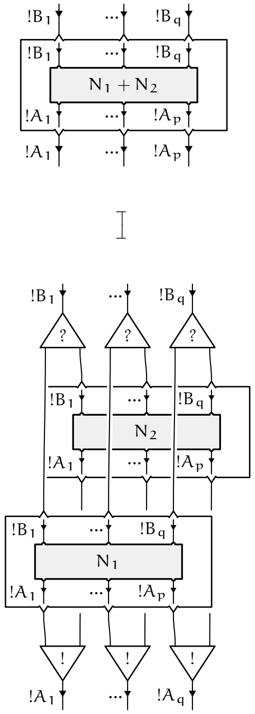


Un seul rédex est gardé dans un état non réduit dans cette superposition de réseaux, celui qui correspond à l'interaction entre deux nouvelles boîtes de *super-promotion*.

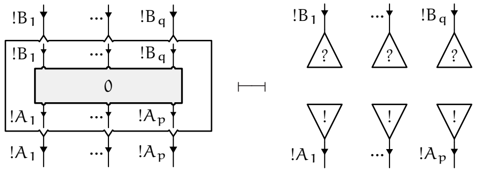
On voit ici apparaître le caractère récursif (non terminant) de l'interaction entre deux boîtes. Ce n'est pas le seul problème que l'on peut reprocher à cette règle. Elle ne respecte pas les sémantiques quantitatives telle que la sémantique finitaire [Ehr05] : dans l'expansion proposée si la boîte originelle pouvait être développée selon une série exponentielle habituelle, la boîte qui apparaît dans l'autre terme ne correspond pas à la même série. Le lecteur trouvera d'avantage de détails à ce propos dans [ER06b, page 28].

9.2.3 Équivalences

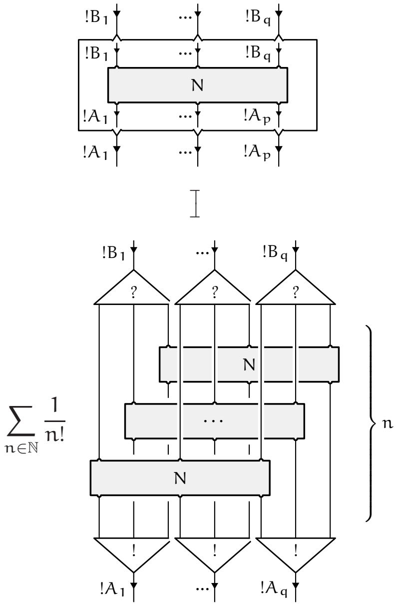
Comme précédemment, une somme possède une portée limitée à la boîte immédiatement englobante. Néanmoins, une boîte de *super-promotion* contenant la somme de deux réseaux est équivalente à un réseau simple :



On remarquera que la terminologie a été choisie de façon à ce que cela corresponde avec la propriété bien connue qui dit que l'exponentielle d'une somme est égale au produit des exponentielles. Ce même morphisme, du point de vue des éléments neutres, est justifié par une équivalence similaire pour une boîte contenant une somme nulle de réseaux :



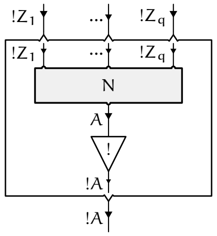
On obtient plus généralement le développement de *Taylor–Ehrhard* suivant :



Appliquée récursivement, cette équivalence permet d'écrire tout réseau sans construction utilisant une boîte. Cela a pour coût la présence de réseaux arbitrairement grands dans la somme obtenue.

9.2.4 Codage de la promotion

Une promotion habituelle se code grâce à une *super-promotion* et une *co-déréliction* placée derrière son unique sortie.



On pourra vérifier que selon ce codage, on retrouve certaines réductions habituelles des boîtes exponentielles, modulo les équivalences énoncées plus haut.

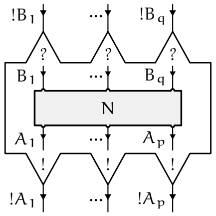
La réduction obtenue pour l'interaction entre deux promotions est néanmoins décevante. Le système présenté dans la section suivante présente une version fonctorielle des boîtes de *super-promotion*. Il a l'avantage de ne pas introduire d'η-expansions inutiles lors des réductions, et on pourra coder les promotions habituelles dans ce système de façon plus satisfaisante.

9.3 Réplication

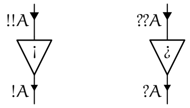
On préfère parfois une version fonctorielle de la *promotion* à une *promotion* classique. De la même façon on pourra préférer travailler avec une version fonctorielle de de la *super-promotion*. On utilisera le nom *réplication* pour cette nouvelle construction.

9.3.1 Boîtes de réplication

Réplication On matérialise cette construction à l'aide d'une boîte représentée comme ceci :



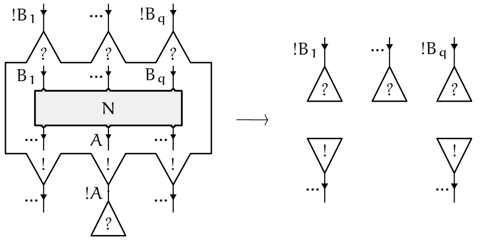
Enfouissement*, *Co-enfouissement Pour retrouver l'expressivité du système précédent nous considérons en plus des *affaiblissements*, des *contractions*, des *déréliction*s (et des opérateurs duaux), les opérateurs *enfouissement* et *co-enfouissement* qui seront typés et représentés comme suit :



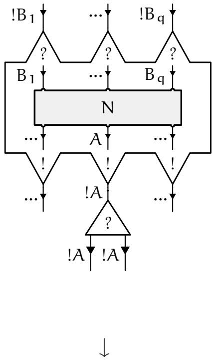
9.3.2 Réduction des réplications fonctorielles

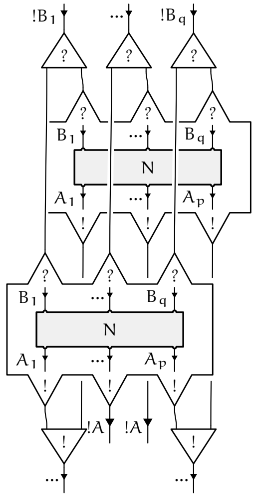
Quatre opérateurs produisent la modalité exponentielle, et leurs quatre opérateurs duaux la consomment. Avec les boîtes, nous devrons considérer un système de réduction des exponentielles qui possède au total $5 \times 5 = 25$ règles. Les neuf interactions entre *affaiblissement*, *contraction*, *déréliction* et opérateurs duaux ont été déjà étudiées et restent inchangées. Nous allons expliciter maintenant les seize règles manquantes.

Affaiblissement — Sortie-Réplication Dans sa version fonctorielle, une boîte de réplication se fait effacer par un *affaiblissement*.

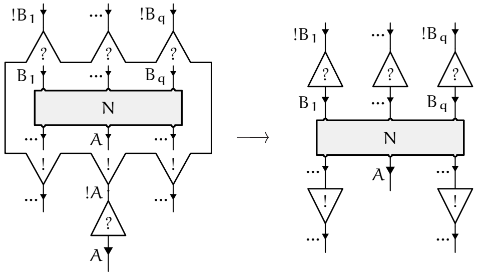


Contraction — Sortie-Réplication Dans sa version fonctorielle, une boîte de réplication se fait dupliquer par une *contraction*.

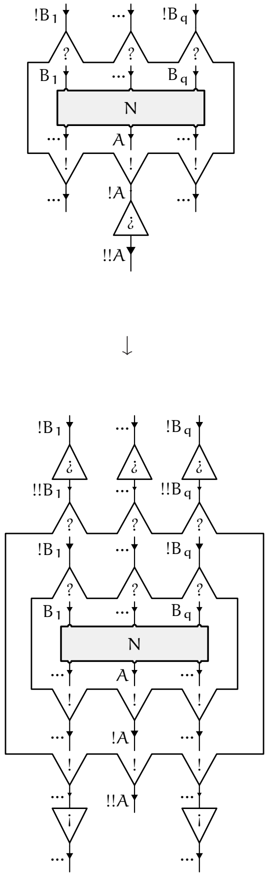




Déréliction — Sortie-Réplication Une boîte de réplication fonctorielle se fait simplement ouvrir par une *déréliction*.

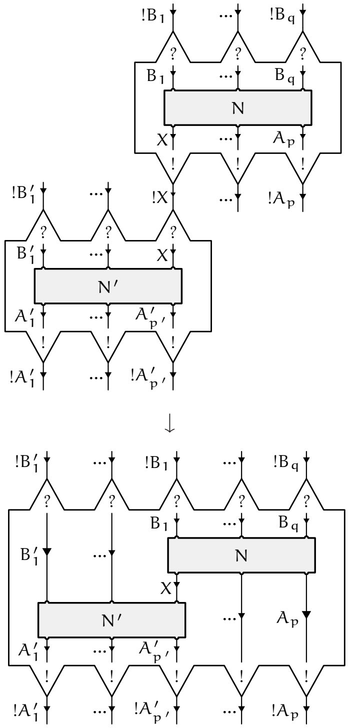


Enfouissement — Sortie-Réplication Une boîte de réplication fonctorielle se fait dédoubler par un *enfouissement*.



Entrée-Réplication — Co-affaiblissement, Entrée-Réplication—Co-contraction, Entrée-Réplication—Co-déréliction, Entrée-Réplication—Co-enfouissement Ces règles de réductions sont obtenues par symétrie des modalités exponentielles. On les déduit des quatre règles que nous venons de présenter.

Entrée-Réplication — Sortie-Réplication Deux boîtes de réplication fonctorielle qui se rencontrent fusionnent :

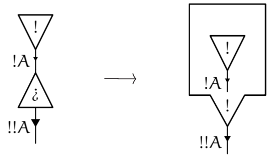


La terminaison de la réduction boîte contre boîte ne pose plus de problème de terminaison. En réalité, dans le système fonctoriel, on s'aperçoit que ce problème vient plus spécifiquement de l'interaction entre *enfouissement* et *co-enfouissement* qui sera étudiée un peu plus loin.

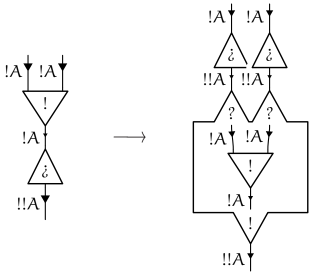
9.3.3 Réduction de l'opérateur *enfouissement*

Maintenant que nous avons explicité toutes les réductions faisant intervenir des boîtes de *réplication*, il reste à expliciter les interactions de l'opérateur *enfouissement* face aux quatre autres opérateurs : le *co-affaiblissement*, la *co-contraction*, la *co-déréliction* et le *co-enfouissement*. On s'apercevra que la complexité de l'interaction entre les opérateurs différentiels et les boîtes de promotion provient essentiellement de leur interaction avec la composante « enfouissement » qui est incluse dans ce type de boîte.

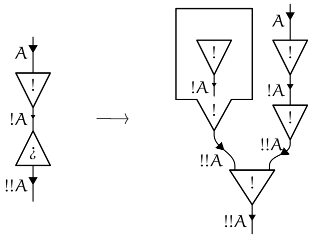
Co-affaiblissement* — *Enfouissement Une façon de deviner cette interaction consiste à remarquer qu'un *affaiblissement* est équivalent à un réseau nul dans une boîte.



Co-contraction* — *Enfouissement Une façon de deviner cette interaction consiste à remarquer qu'une *contraction* est équivalente aux deux réseaux ayant la même interface et construits avec un fil et un *affaiblissement*, sommés et mis dans une boîte.

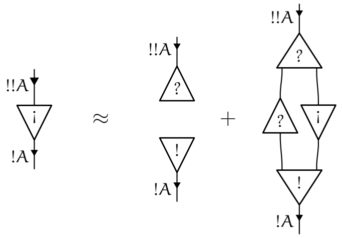


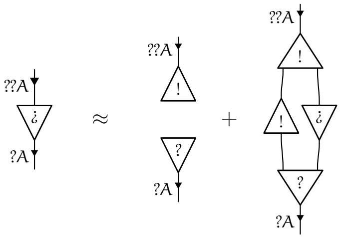
Co-déréliction* — *Enfouissement Cette règle décrit une réduction locale qui correspond à la règle de la chaîne.



***Co-enfouissement* — *Affaiblissement*, *Co-enfouissement*—*Contraction*, *Co-enfouissement*—*Déréliction*,** Une fois de plus, ces règles sont obtenues par symétrie des modalités exponentielles.

Co-enfouissement* — *Enfouissement L'interaction problématique est à présent celle entre opérateurs *co-enfouissement* et *enfouissement*. À nouveau, on trouve des expansions :





Et la règle que l'on peut écrire avec celles-ci ne termine pas et ne respecte pas les sémantiques quantitatives.

Voir Fig. 9.1 (page 110).

En particulier, contrairement à la *déréliction*, l'*enfouissement* a une interaction déterministe avec les opérateurs structurels (*co-affaiblissement* et *co-contraction*) mais pas avec lui-même.

9.3.4 Équivalences

Tout se passe comme dans le cas des boîtes de *super-promotion*. On fera seulement attention à rajouter les *déréliction*s et *co-déréliction*s nécessaires dans le développement de *Taylor–Ehrhard* :

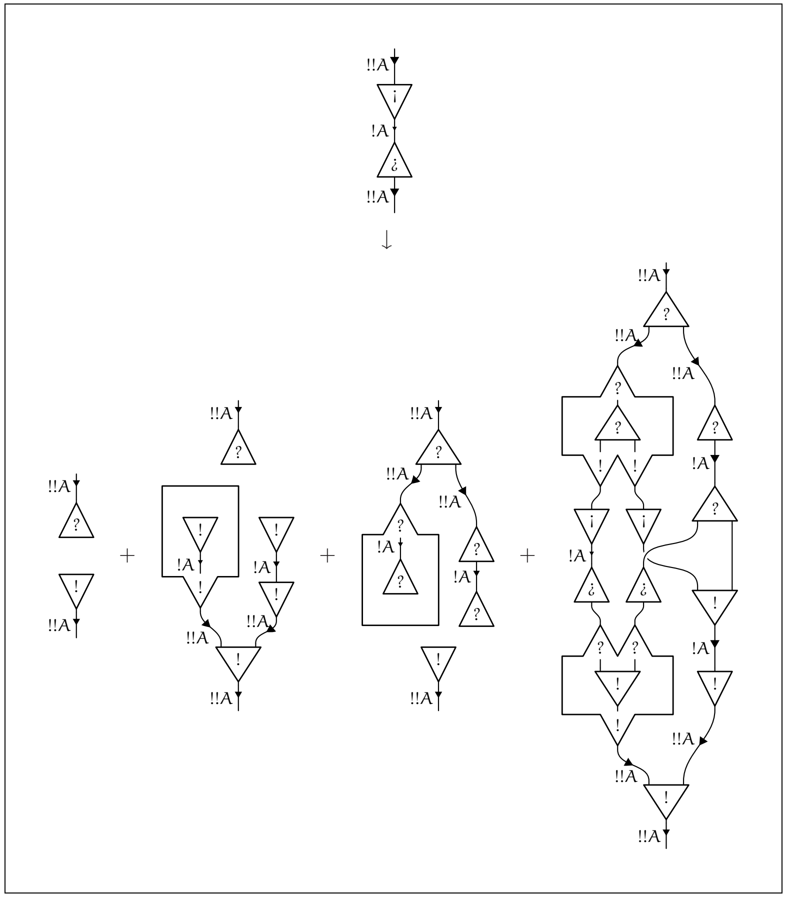
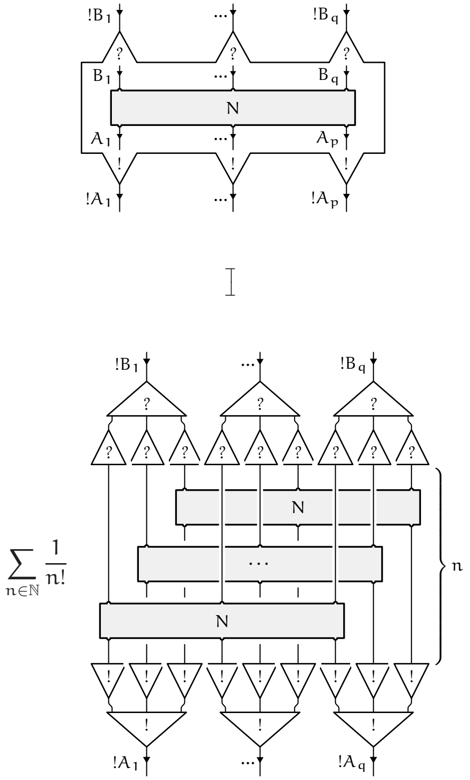
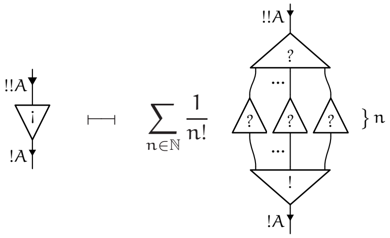
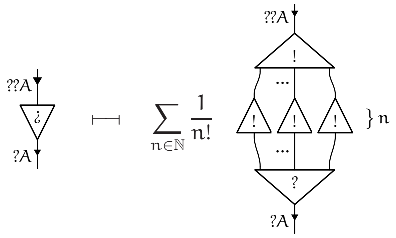


Figure 9.1



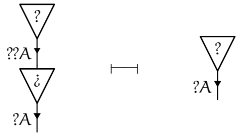
On remarquera que les opérateurs *enfouissement* et *co-enfouissement* peuvent également être développés :



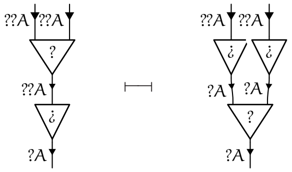


De plus, on constate les équivalences suivantes :

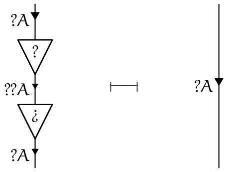
Affaiblissement → Enfouissement



Contraction → Enfouissement



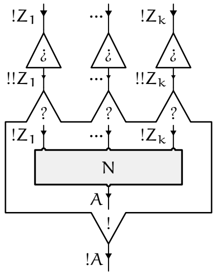
Déréliction → Enfouissement



Les mêmes équivalences peuvent bien sûr être écrites pour les opérateurs duaux. On peut également en trouver certaines qui font intervenir des boîtes.

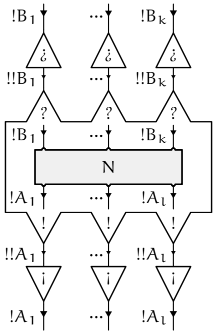
9.3.5 Codage de la promotion

Une promotion habituelle se code grâce à une réplication et des *enfouissements* placés en amont des entrées.



À présent, la règle habituelle de réduction entre deux promotions est naturellement codée par le système. Les autres interactions que peuvent avoir les boîtes sont simulées modulo les équivalences que nous venons d'évoquer impliquant les opérateurs *enfouissement*.

La *super-promotion* se retrouve elle de cette façon :



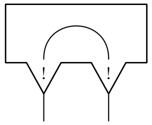
9.4 Discussion et perspectives

9.4.1 Justification sémantique

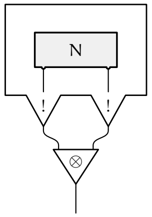
La principale justification aux systèmes que nous avons présentés est de nature sémantique. Nous présenterons une sémantique relationnelle pour les nouvelles constructions différentielles en 12.2. Les codages que nous venons d'écrire se comprennent d'ailleurs plus facilement lorsqu'on connaît la sémantique relationnelle qui est associée aux différentes constructions qu'ils contiennent. Les réductions et codages que nous avons décrits préservent tous cette sémantique.

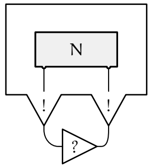
9.4.2 Critère de validité

Il est difficile d'exprimer un critère de validité dans ces systèmes tels qu'ils ont été introduits. On peut chercher une notion de cycle au sens du critère de *Danos–Regnier* en considérant les nœuds *co-contraction* comme des nœuds *tenseur* et les nœuds *contraction* comme des nœuds *par*, mais c'est relativement compliqué. Le réseau suivant contient un cycle caché :



Ce cycle apparaît lorsqu'on décompose la boîte selon un développement de *Taylor–Ehrhard*. On a bien envisagé de travailler avec le critère hérité de ce développement, mais celui-ci s'avère difficile à manipuler. Donnons d'autres exemples particulièrement atypiques : établir un lien entre deux sorties d'une même boîte donne généralement un réseau correct, pourvu que son contenu soit lui-même correct (obtenu par réduction d'un réseau correct).





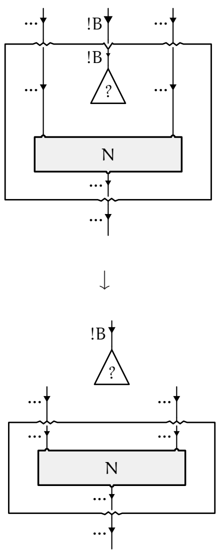
Le problème est que les différents liens sortants d'une boîte (de *super-promotion* ou de *réplication*) doivent être considérés comme liés par un connecteur ⊗ et non pas par un connecteur ⅋. Les séquents utilisés jusque là pour définir l'interface de nos réseaux ne savent parler que de relations ⅋. Nous donnerons dans le prochain chapitre un formalisme de structures qui savent parler de relations ⊗ entre les fils d'une interface aussi bien que de relations ⅋. Ce formalisme donnera un critère de validité solide qui passe par l'existence d'une séquentialisation (appelée ainsi bien qu'elle ne soit pas faite avec des séquents).

9.4.3 Non terminaison

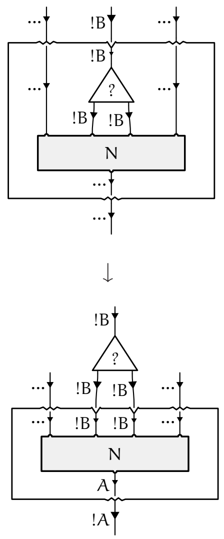
Les systèmes présentés possèdent chacun une règle problématique qui ne termine pas et qu'il serait préférable d'éviter si on le pouvait. Pour donner une idée des travaux en cours, certaines raisons poussent à croire que la non terminaison des systèmes présentés est artificiellement introduite parce qu'on considère des boîtes qui sont des macro-entités (des constructions globales).

Un système hybride où les entrées et sorties de boîtes exponentielles peuvent être indépendamment de toutes les autres portes de type soit fonctoriel soit non fonctoriel aurait un sens. D'un point de vue sémantique cela ne pose aucun problème. Côté opérationnel, la dynamique des portes fonctorielles peut être conservée, et pour éviter les réductions problématiques on se rend compte que la dynamique des portes non fonctorielles peut être inversée : elle peut se faire avec l'intérieur de la boîte plutôt qu'avec l'extérieur.

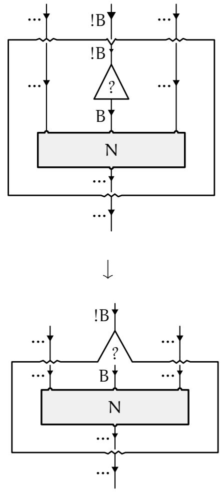
Affaiblissement — Entrée-non-fonctorielle



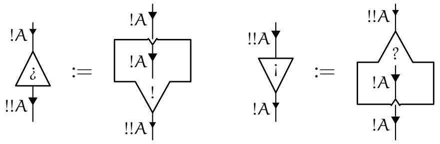
Contraction — Entrée-non-fonctorielle



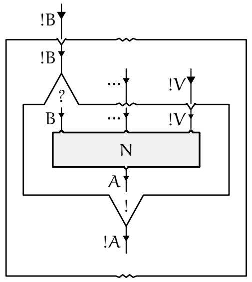
Déréliction — Entrée-non-fonctorielle



Les opérateurs *enfouissement* et *co-enfouissement* n'ont plus besoin d'être introduits, il s'expriment avec ces boîtes hybrides. Dans le codage qui leur correspond leur port principal a changé de direction, et cela évite précisément leur interaction problématique.



Une seule difficulté apparaît, les rédex du type suivant :



Mais il semble qu'une réduction adéquate peut être écrite dans système où ces boîtes sont localisées (comme dans les chapitres de la première partie) et peuvent partiellement se superposer...

9.4.4 Sur le λ-calcul avec ressources et super-promotion

Revenons également sur la *super-promotion* dans le λ-calcul avec ressources. En fait, la réduction que nous avons proposée pour la *super-promotion* (rappelée ici) n'est pas la seule réduction envisageable.

$$\langle \lambda x\, s\rangle \mathsf{U}^{!}\mathsf{R} \xrightarrow{a} \langle \lambda x\, s\rangle \mathsf{R} + \langle \lambda x\, s\rangle \mathsf{U}\mathsf{U}^{!}\mathsf{R}$$

Nous avons introduit l'opération d'extraction d'une ressource en 8.1.7, or avec cette opération on pourrait également envisager une réduction du type :

$$\langle \lambda x\, s\rangle \mathsf{R} \longrightarrow s[0/x] + \int_{\mathsf{R}} \left\langle \frac{\partial}{\partial x} s \cdot d\mathsf{U} \right\rangle \mathsf{U}$$

Cette réduction rappelle une formule mathématique bien connue lorsque le λ-terme $\lambda x\, s$ est assimilé à une fonction $s : \mathbb{R} \longrightarrow \mathbb{R}$:

$$s(r) \;=\; s(0) + \int_0^r s'(u)du$$

C'est déjà une nouvelle variation pour le λ-calcul avec ressource de base, qui est plus proche de celle des réseaux (car symétrique en offre et demande), et elle s'étend à la *promotion* ou même à la *super-promotion* :

$$\frac{\partial}{\partial x}\mathsf{R}^{!}\cdot u \;:=\; \mathsf{R}^{!}\left(\frac{\partial}{\partial x}\mathsf{R}\cdot u\right)$$

$$\int_{\mathsf{R}^{!}} t[d\mathsf{U},\mathsf{U}] \;:=\; \int_{\mathsf{R}} t[d\mathsf{U},\mathsf{R}^{!}\mathsf{U}]$$

Elle ne provoque pas automatiquement l'« explosion » dont il a été plusieurs fois question. En revanche, bien que sa sémantique qualitative soit correcte, sa sémantique quantitative ne l'est pas (même dans le calcul sans promotion).

Chapitre 10

Logique différentielle

Nous proposons dans ce chapitre un nouveau formalisme logique qui permet d'exprimer sainement le paradigme différentiel. Il est fortement inspiré du calcul des structures, étudié par *Alessio Guglielmi* et *Lutz Straßburger* dans le cadre de la logique linéaire [Str03]. Il permettra une formulation plus claire des règles de réduction que nous avons évoquées dans les réseaux, car il contient intrinsèquement le critère de correction qui nous manquait.

10.1 La rigidité des séquents

Peut-on exprimer le calcul différentiel tel qu'il a été introduit dans les réseaux à l'aide d'un calcul des séquents ? On rappelle que les connecteurs multiplicatifs sont introduits en logique linéaire de la façon suivante :

$$\dfrac{\vdash \Gamma_1, A \quad \vdash \Gamma_2, B}{\vdash \Gamma_1, \Gamma_2, A \otimes B}\;\textit{tenseur} \qquad \dfrac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}\;\textit{par}$$

$$\dfrac{}{\vdash 1}\;\textit{unité} \qquad \dfrac{\vdash \Gamma}{\vdash \Gamma, \bot}\;\textit{co-unité}$$

On peut s'en inspirer et introduire les constructions *co-contraction*, *contraction*, *co-affaiblissement* et *affaiblissement* de façon similaire.

$$\dfrac{\vdash \Gamma_1, !A \quad \vdash \Gamma_2, !A}{\vdash \Gamma_1, \Gamma_2, !A}\;\textit{co-contraction}$$

$$\dfrac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A}\;\textit{contraction}$$

$$\dfrac{}{\vdash !A}\;\textit{co-affaiblissement} \qquad \dfrac{\vdash \Gamma}{\vdash \Gamma, ?A}\;\textit{affaiblissement}$$

Mais on se heurte à une particularité lorsqu'on veut écrire la règle de réduction lors de l'interaction entre une *co-contraction* et une *contraction*. La syntaxe des séquents est très rigide, et c'est au prix de quelques contorsions que l'on parvient à écrire la règle d'élimination pour une telle coupure.

Voir Fig. 10.1 (page 116).

L'astuce consiste à introduire un peu artificiellement (on pourra comprendre plus tard ce qui justifie l'emploi de ce qualificatif) de nouveaux *axiomes* pour effectuer la jonction entre les nouvelles *coupures*. Mais ce genre de contournement à des limites. Si on souhaite introduire une *super-promotion* dans ce calcul des séquents, on se trouve cette fois face à une réelle impossibilité syntaxique lorsqu'on souhaite exprimer leur dynamique.

10.2 Un calcul de structures

Les structures qu'on considère contiennent des formules de la logique linéaire, elles possèdent la syntaxe suivante :

$$\sigma \quad ::= \quad \cdot \;\mid\; \sigma\,;\sigma \;\mid\; \circ \;\mid\; \sigma\,,\sigma \;\mid\; A$$

On utilisera les noms σ, τ, ω… pour désigner des structures. Les structures forment de bon candidats pour définir l'interface d'un réseau, meilleures que de simples listes de formules (i.e. des séquents). Abstraction faite des formules, les structures définissent une interface pour des câblages qui, contrairement aux simples listes de liens, évite les problèmes de cyclicité lors des compositions.

On va introduire des *dérivations*. Elles sont obtenues par l'application successive d'un certain nombre de *règles* élémentaires :

$$\dfrac{\sigma}{\tau}$$

et possèdent la forme suivante :

$$\begin{array}{c} \sigma \\ \vdots \\ \hline \omega_i \\ \hline \omega_{i+1} \\ \hline \vdots \\ \hline \tau \end{array} \qquad \text{(notée } \begin{array}{c} \sigma \\ \hline\hline \tau \end{array} \text{ de façon abrégée)}$$

$$\dfrac{\dfrac{\overline{\overline{\vdash \Gamma_1, !A}}\,\pi_1 \quad \overline{\overline{\vdash \Gamma_2, !A}}\,\pi_2}{\vdash \Gamma_1, \Gamma_2, !A}\;\textit{co-contraction} \qquad \dfrac{\overline{\overline{\vdash ?A^\perp, ?A^\perp, \Gamma_3}}\,\pi_3}{\vdash ?A^\perp, \Gamma_3}\;\textit{contraction}}{\vdash \Gamma_1, \Gamma_2, \Gamma_3}\;\textit{cut}$$

$$\downarrow$$

$$\dfrac{\overline{\overline{\vdash \Gamma_1, !A}}\,\pi_1 \quad \overline{\overline{\vdash \Gamma_2, !A}}\,\pi_2 \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash ?A^\perp, !A}\,\textit{ax} \quad \dfrac{}{\vdash ?A^\perp, !A}\,\textit{ax}}{\vdash ?A^\perp, ?A^\perp, !A}\;\textit{co-contraction} \quad \dfrac{\dfrac{}{\vdash ?A^\perp, !A}\,\textit{ax} \quad \dfrac{}{\vdash ?A^\perp, !A}\,\textit{ax}}{\vdash ?A^\perp, ?A^\perp, !A}\;\textit{co-contraction} \quad \overline{\overline{\vdash ?A^\perp, ?A^\perp, \Gamma_3}}\,\pi_3}{\vdash ?A^\perp, ?A^\perp, ?A^\perp, ?A^\perp, \Gamma_3}\;\textit{cut}^2}{\vdash ?A^\perp, ?A^\perp, ?A^\perp, ?A^\perp, \Gamma_3}\;\textit{commutation}}{\vdash ?A^\perp, ?A^\perp, \Gamma_3}\;\textit{contraction}^2}{\vdash \Gamma_1, \Gamma_2, \Gamma_3}\;\textit{cut}^2$$

Figure 10.1

Ces dérivations correspondent à un raisonnement et donnent une façon de déduire une structure τ, dite *conclusion*, d'une structure σ, dite *hypothèse*. Une dérivation est une *preuve* de sa conclusion τ lorsque cette dérivation part d'une hypothèse vide $\langle \cdot \rangle$. C'est une *contre-preuve* de son hypothèse σ lorsqu'elle aboutit à la conclusion inerte $\langle \circ \rangle$. Nous évoquerons plus tard un résultat de cohérence stipulant qu'il ne peut exister à la fois une preuve et une contre-preuve d'une même formule.

10.2.1 Règles structurelles

Fondamentalement les symboles binaires $\langle\ ;\ \rangle$ et $\langle\ ,\ \rangle$ utilisés pour les structures sont commutatifs, un échange est matérialisé par l'utilisation d'une règle.

$$\dfrac{\tau\,;\sigma}{\sigma\,;\tau}\;\textit{échange} \qquad \dfrac{\tau\,,\sigma}{\sigma\,,\tau}\;\textit{échange}$$

Ces symboles admettent respectivement pour éléments neutres $\langle \cdot \rangle$ et $\langle \circ \rangle$, à gauche et à droite.

$$\dfrac{\sigma}{\cdot\,;\sigma}\;\textit{neutralité} \qquad \dfrac{\cdot\,;\sigma}{\sigma}\;\textit{neutralité}$$

$$\dfrac{\sigma}{\circ\,,\sigma}\;\textit{neutralité} \qquad \dfrac{\circ\,,\sigma}{\sigma}\;\textit{neutralité}$$

Viennent ensuite des propriétés d'associativité.

$$\dfrac{(\sigma\,;\tau)\,;\omega}{\sigma\,;(\tau\,;\omega)}\;\textit{associativité} \qquad \dfrac{(\sigma\,,\tau)\,,\omega}{\sigma\,,(\tau\,,\omega)}\;\textit{associativité}$$

Dans cet ensemble toutes les règles sont réversibles. Ces règles pourraient donc nous servir à quotienter les structures, néanmoins on ne souhaite pas perdre l'identité des emplacements de formules qu'elles contiennent. Dans la dérivation suivante :

$$\dfrac{\dfrac{\dfrac{((A\,;A)\,;A)\,;A}{(A\,;A)\,;(A\,;A)}\;\textit{associativité}}{(A\,;A)\,;(A\,;A)}\;\textit{échange}}{((A\,;A)\,;A)\,;A}\;\textit{associativité}$$

la structure donnée en hypothèse et la structure obtenue en conclusion sont en apparence similaires, mais il ne faut pas oublier de prendre en compte le fait que la position de deux formules A ont été inversées. Même si ce sont les mêmes formules, leur emplacement les distinguent. Pour être plus précis, il faudrait étiqueter les emplacements de cette façon :

$$\dfrac{\dfrac{\dfrac{((A_3\,;A_4)\,;A_1)\,;A_2}{(A_3\,;A_4)\,;(A_1\,;A_2)}\;\textit{associativité}}{(A_1\,;A_2)\,;(A_3\,;A_4)}\;\textit{échange}}{((A_1\,;A_2)\,;A_3)\,;A_4}\;\textit{associativité}$$

En pratique, on écrira toutes les dérivations composées uniquement de règles structurelles de façon abrégée :

$$\frac{A\,;A\,;A\,;A}{A\,;A\,;A\,;A}\;\diamond$$

en expliquant à coté quelle est la permutation correspondante lorsqu'il y a ambiguïté (ce qui n'arrive que lorsqu'une même formule apparaît plusieurs fois).

Règles d'appariement

Une dernière règle structurelle mélange les deux symboles binaires, elle est déjà connue sous le nom de *distributivité faible* et existe sous deux formes :

$$\frac{(\sigma\,,\tau)\,;\omega}{\sigma\,,(\tau\,;\omega)}\;\textit{appariement} \qquad \frac{(\sigma\,,\tau)\,;\omega}{(\sigma\,;\omega)\,,\tau}\;\textit{appariement}$$

Par commutativité, et grâce à l'utilisation de règles sous des contextes structurels (ce sera expliqué bientôt), il n'est en fait nécessaire d'introduire que l'une des deux, l'autre pouvant être déduite. Cependant, utiliser l'une ou l'autre de ces formulations fait un choix irréversible sur la façon d'utiliser l'hypothèse (ou de prouver la conclusion).

Toutes les règles que nous avons vu jusqu'à présent laissent les formules inchangées, et sont appelées règles structurelles. Nous verrons qu'un réseau peut être associé à une dérivation de σ vers τ. Ce réseau contient des ports libres typés par les formules qui apparaissent dans l'hypothèse σ et des ports libres typés par les formules qui apparaissent dans la conclusion τ. Toutes les règles structurelles évoquées vont être traduites par de simples câblages qui relient les premiers aux seconds.

10.2.2 Règles logiques

Les règles qui inspectent le contenu des formules sont appelées règles *logiques*. Une fois de plus, on va énumérer ces règles en les regroupant dans différents fragments.

Fragment identité Ces règles sont nécessaires pour atteindre l'expressivité classique, et permettent par exemple de coder la logique linéaire.

$$\frac{\cdot}{A\,,A^{\perp}}\;\textit{ax}{\downarrow} \qquad \frac{A^{\perp}\,;A}{\circ}\;\textit{cut}{\uparrow}$$

Fragment multiplicatif Ce fragment est celui qui empaquette la structure dans les formules de la logique linéaire.

$$\frac{\cdot}{1}\;\textit{unité}{\downarrow} \qquad \frac{1}{\cdot}\;\textit{co-unité}{\uparrow} \qquad \frac{\circ}{\perp}\;\textit{co-unité}{\downarrow} \qquad \frac{\perp}{\circ}\;\textit{unité}{\uparrow}$$

$$\frac{A\,;B}{A\otimes B}\;\textit{tenseur}{\downarrow} \qquad \frac{A\otimes B}{A\,;B}\;\textit{par}{\uparrow}$$

$$\frac{A\,,B}{A\parr B}\;\textit{par}{\downarrow} \qquad \frac{A\parr B}{A\,,B}\;\textit{tenseur}{\uparrow}$$

Dans les réseaux, les règles structurelles vont être traduites par de simples manipulations de fils. Ces nouvelles règles seront traduites par des réseaux qui regroupent plusieurs fils hypothèse en un seul fil conclusion (ou inversement) : les réseaux élémentaires formés d'un nœud multiplicatif. Ceux-ci pourront être orientés de deux façons, leur port principal peut être considéré comme un port hypothèse ou comme un port conclusion, et c'est pour cela qu'il y a deux règles associées à chacun.

Fragment exponentiel Les règles qui correspondent aux opérateurs différentiels sont les suivantes :

$$\frac{\cdot}{!A}\;\textit{co-affaiblissement}{\downarrow} \qquad \frac{!A}{\cdot}\;\textit{affaiblissement}{\uparrow}$$

$$\frac{\circ}{?A}\;\textit{affaiblissement}{\downarrow} \qquad \frac{?A}{\circ}\;\textit{co-affaiblissement}{\uparrow}$$

$$\frac{A}{!A}\;\textit{co-déréliction}{\downarrow} \qquad \frac{!A}{A}\;\textit{déréliction}{\uparrow}$$

$$\frac{A}{?A}\;\textit{déréliction}{\downarrow} \qquad \frac{?A}{A}\;\textit{co-déréliction}{\uparrow}$$

$$\frac{!A\,;!A}{!A}\;\textit{co-contraction}{\downarrow} \qquad \frac{!A}{!A\,;!A}\;\textit{contraction}{\uparrow}$$

$$\frac{?A\,,?A}{?A}\;\textit{contraction}{\downarrow} \qquad \frac{?A}{?A\,,?A}\;\textit{co-contraction}{\uparrow}$$

10.2.3 Contexte structurel

Toutes ces règles, qu'elles soient structurelles ou logiques, peuvent être placées dans un contexte structurel. Par exemple, les règles suivantes sont valides, elles sont obtenues à partir de la règle *contraction*↓ en plaçant son hypothèse et sa conclusion dans un même contexte, $\sigma\,,\square$ pour la première et $\sigma\,;\square$ pour la seconde.

$$\frac{\sigma\,,?A\,,?A}{\sigma\,,?A}\;-\,,\textit{contraction}{\downarrow}$$

$$\frac{\sigma\,;(?A\,,?A)}{\sigma\,;?A}\;-\,;\textit{contraction}{\downarrow}$$

On va de plus utiliser des abréviations lorsque plusieurs règles peuvent être utilisées simultanément et que l'ordre dans lequel elles sont placées n'importe pas :

$$\frac{?A\,,?A\,,?A\,,?A}{?A\,,?A}\;\textit{contraction}{\downarrow}\,,\textit{contraction}{\downarrow}$$

Ce n'est qu'une notation, mais on peut ainsi placer plusieurs règles au sein d'un même contexte structurel.

On peut dès à présent comprendre l'avantage de ce calcul de structures par rapport à une présentation classique sous forme de séquents. Considérons la règle :

$$\frac{\sigma\,,(!A\,;!A)}{\sigma\,,!A}\;-\,,\textit{co-contraction}{\downarrow}$$

Dans la tentative faite pour d'introduire la *co-contraction* dans le calcul des séquents les deux liens de type !A se trouvent dans des séquents séparés. Les autres formules présentes dans ces séquents forment un contexte spécifique à chaque lien, mais contrairement à ce qu'il se passe pour σ dans notre règle, la syntaxe des séquents ne permet pas d'exprimer un contexte partagé entre les deux liens. Ce type de flexibilité manque pour pouvoir écrire correctement une réduction avec des séquents.

10.2.4 Remarques

Nous avons donné un ensemble de règles élémentaires qui correspond à la logique différentielle (sans promotion). Ce choix a été fait car il ne nécessite pas l'introduction des règles qui correspondent aux constructions des réseaux qui nécessitent des boîtes. Nous ne considérerons ces règles que plus tard, en 10.5.

Deux niveaux

La principale nouveauté de notre calcul est la présence de deux niveaux : un niveau structurel et un niveau logique. Les calculs de structures connus cherchent généralement à embarquer le contenu logique dans un monde purement structurel.

Notons d'ailleurs que le fragment identité constitué des règles *axiome* et *coupure* est à la limite entre le monde structurel et le monde logique, il est contenu dans une généralisation :

$$\frac{\cdot}{\sigma\,,\sigma^{\perp}}\;\textit{multi-ax}{\downarrow} \qquad \frac{\sigma^{\perp}\,;\sigma}{\circ}\;\textit{multi-cut}{\uparrow}$$

à laquelle on peut donner un sens, et qui s'écrit presque au niveau structurel. Ce ne sont pas exactement des règles structurelles car l'orthogonalité utilisée ici pour les structures s'appuie tout de même sur l'orthogonalité de la logique.

Deux dualités

Notre calcul possède deux formes de dualité. Les règles ont été introduites par groupes de deux, de manière telle qu'à toute dérivation de σ vers τ il correspond une dérivation de $\tau^{\perp}$ vers $\sigma^{\perp}$. C'est la symétrie apportée par le formalisme des dérivations de structures.

Il est également envisageable d'associer à toute dérivation de σ vers τ une dérivation de $\sigma^{\perp}$ vers $\tau^{\perp}$ puisque les groupes de deux règles déjà évoqués ont eux-mêmes été introduits par deux selon cette nouvelle dualité : à chaque cellule correspond une co-cellule. C'est la dualité du calcul différentiel. Mais les règles d'appariement font exception, ajouter un *co-appariement* permettrait de créer des cycles. Les règles *ax* et *cut* du fragment identité échappent également à cette forme de dualité. Elles auraient pour correspondantes :

$$\frac{\circ}{A\,;A^{\perp}}\;\textit{co-ax}{\downarrow} \qquad \frac{A^{\perp}\,,A}{\cdot}\;\textit{co-cut}{\uparrow}$$

Nous n'avons pas introduit ces règles. Leur adjoindre une dynamique semble difficile. Il se pourrait cependant que se soit possible pour les règles généralisées correspondantes (*co-multi-ax*↓ et *co-multi-cut*↑), mais cela sort du propos dans lequel on s'est engagés ici. Ces règles rappellent (et pourraient à tout hasard correspondre à) la création et la mesure de paires quantiques.

10.3 Sémantique opérationnelle

Une dynamique peut être mise en place à l'intérieur de notre calcul de structures. Nous verrons plus tard que celle-ci est en étroite correspondance avec les réductions de réseaux.

10.3.1 Principe de la dynamique

Les règles données se répartissent en trois catégories. Il est fondamental de les distinguer pour comprendre le mécanisme de réduction proposé. On distingue :

- les règles *structurelles*, qui incluent la règle particulière d'*appariement*
- les règles logiques d'*introduction*, qui ont été suffixées par un symbole ‹ ↓ ›
- les règles logiques d'*élimination*, qui ont été suffixées par un symbole ‹ ↑ ›

Les notations suivantes tiennent compte de ce partitionnement et seront utilisées pour désigner les dérivations constituées uniquement d'introductions et de règles structurelles (respectivement, d'éliminations et de règles structurelles) :

$$\begin{array}{cl} \sigma & \\ \hline\hline \tau & \downarrow\!\diamond \end{array} \qquad\qquad \begin{array}{cl} \sigma & \\ \hline\hline \tau & \uparrow\!\diamond \end{array}$$

Formes normales

Nous souhaitons donner une forme particulière à nos dérivations, de sorte que les éliminations soient placées au dessus des introductions. En ce qui concerne les règles structurelles il faut accepter leur présence à des positions arbitraires au milieu des autres constructions. Les dérivations de cette forme seront dites en *forme normale* :

$$\begin{array}{cl} \sigma & \\ \hline\hline \omega & \uparrow\!\diamond \\ \hline\hline \tau & \downarrow\!\diamond \end{array} \qquad \text{(aussi notées } \begin{array}{cl} \sigma & \\ \hline\hline \tau & \updownarrow\!\diamond \end{array} \text{ de façon abrégée)}$$

Ce genre de forme normale a également été proposé par *Kai Brünnler* pour un calcul des structures qui traite la logique classique [Brü06].

Réduction

La dynamique que nous allons introduire cherche à réduire les dérivations en exhibant un équivalent logique (plus proche de la forme normale) à toute combinaison de deux règles dans laquelle une introduction est suivie d'une élimination. Les réductions logiques auront la forme suivante :

$$\begin{array}{cl} \sigma & \\ \hline \omega & \downarrow \\ \hline \tau & \uparrow \end{array} \quad\longrightarrow\quad \begin{array}{cl} \sigma & \\ \hline\hline \tau & \updownarrow\!\diamond \end{array}$$

Pour que ces règles se rencontrent, il faudra également savoir faire commuter les règles structurelles avec les introductions :

$$\begin{array}{cl} \sigma & \\ \hline \omega & \downarrow \\ \hline \tau & \diamond \end{array} \quad\longrightarrow\quad \begin{array}{cl} \sigma & \\ \hline \omega' & \diamond \\ \hline \tau & \downarrow \end{array}$$

et/ou avec les éliminations :

$$\begin{array}{cl} \sigma & \\ \hline \omega & \diamond \\ \hline \tau & \uparrow \end{array} \quad\longrightarrow\quad \begin{array}{cl} \sigma & \\ \hline \omega' & \uparrow \\ \hline \tau & \diamond \end{array}$$

Le seul cas où on ne parviendra pas simplement à les faire commuter avec une introduction qui souhaite descendre ou une élimination qui souhaite remonter est le cas des règles d'appariement. En direction du haut on ne peut les faire réagir atomiquement avec des *axiomes* qui relient deux de leurs formules, et en direction du bas c'est la même chose avec les *coupures*. Mais cela ne pose pas de problème car elles disparaîtront lorsque l'*axiome* et la *coupure* sont simultanément présents de part et d'autre. Pour cela on s'autorise la réduction suivante, déclenchée de façon atypique par une combinaison de trois règles :

$$\begin{array}{cl} \cdot\,;A & \\ \hline (A\,,A^{\perp})\,;A & \textit{axiome}{\downarrow}\,;- \\ \hline A\,,(A^{\perp}\,;A) & \textit{appariement} \\ \hline A\,,\circ & -\,,\textit{coupure}{\uparrow} \end{array} \quad\longrightarrow\quad \begin{array}{cl} \cdot\,;A & \\ \hline A\,,\circ & \diamond \end{array}$$

10.3.2 Commutations

Commutations de règles logiques

Lorsqu'une introduction est suivie d'une élimination mais que ces deux constructions n'agissent pas sur la même formule, une réduction simple de commutation à lieu. Par exemple :

$$\begin{array}{cl} A\,;B\,;!C & \\ \hline A\otimes B\,;!C & \textit{tenseur}{\downarrow}\,;- \\ \hline A\otimes B\,;!C\,;!C & -\,;\textit{contraction}{\uparrow} \end{array}$$

$$\downarrow$$

$$\begin{array}{cl} A\,;B\,;!C & \\ \hline A\,;B\,;!C\,;!C & -\,;-\,;\textit{contraction}{\uparrow} \\ \hline A\otimes B\,;!C\,;!C & \textit{tenseur}{\downarrow}\,;-\,;- \end{array}$$

Une fois de plus, l'ordre important peu, ces séquences de deux règles pourraient être résumées par une règle simultanée (implicitement considérée en forme normale lorsqu'on s'intéresse à la dynamique) :

$$\begin{array}{cl} A\,;B\,;!C & \\ \hline A\otimes B\,;!C\,;!C & \textit{tenseur}{\downarrow}\,;\textit{contraction}{\uparrow} \end{array}$$

Toutes ces dérivations se traduiront de façon identique dans les réseaux.

Commutations structurelles

Donnons également quelques exemples de commutations pour les règles structurelles. Il faut savoir que ces règles se prêtent facilement aux commutations sur chaque emplacement qu'elles mettent en jeu.

Pour les règles d'appariement, excepté le problème déjà soulevé concernant la commutation avec *axiomes* ou *coupures*, les autres commutations se passent très bien. On les utilise pour faire remonter les éliminations et descendre les introductions. Par exemple :

$$\begin{array}{cl} (A\,;B)\,,(\tau\,;\omega) & \\ \hline (A\otimes B)\,,(\tau\,;\omega) & \textit{tenseur}{\downarrow}\,,- \\ \hline (A\otimes B\,,\tau)\,;\omega & \textit{appariement} \end{array}$$

$$\downarrow$$

$$\begin{array}{cl} (A\,;B)\,,(\tau\,;\omega) & \\ \hline ((A\,;B)\,,\tau)\,;\omega & \textit{appariement} \\ \hline (A\otimes B\,,\tau)\,;\omega & (\textit{tenseur}{\downarrow}\,,-)\,;- \end{array}$$

ou bien sur un autre emplacement de la même règle structurelle :

$$\begin{array}{cl} \sigma\,,((A\,;B)\,;\omega) & \\ \hline \sigma\,,(A\otimes B\,;\omega) & -\,,(\textit{tenseur}{\downarrow}\,;-) \\ \hline (\sigma\,,A\otimes B)\,;\omega & \textit{appariement} \end{array}$$

$$\downarrow$$

$$\begin{array}{cl} \sigma\,,((A\,;B)\,;\omega) & \\ \hline (\sigma\,,(A\,;B))\,;\omega & \textit{appariement} \\ \hline (\sigma\,,A\otimes B)\,;\omega & (-\,,\textit{tenseur}{\downarrow})\,;- \end{array}$$

10.3.3 Réductions logiques

Une réduction est effectuée lorsqu'une introduction d'un connecteur logique est immédiatement suivie de l'élimination de ce connecteur. Ces cas de figure sont appelés rédex logiques. Si une introduction n'est pas immédiatement suivie d'une élimination correspondante, on pourra faire descendre cette introduction par les règles de commutation déjà présentées et on lui trouvera une élimination correspondante. À moins qu'elle n'atteigne, et c'est d'autant plus satisfaisant, la conclusion de notre dérivation. De la même manière, on peut faire remonter les éliminations.

Prenons pour exemple la réduction logique suivante (qui nous intéresse puisque c'est celle que nous n'arrivons pas à formuler naturellement avec des séquents classiques) :

$$\begin{array}{cl} !A\,;!A & \\ \hline !A & \textit{co-contraction}{\downarrow} \\ \hline !A\,;!A & \textit{contraction}{\uparrow} \end{array}$$

$$\downarrow$$

$$\begin{array}{cl}
!A\,;\,!A & \\ \cline{1-1}
!A\,;\,!A\,;\,!A\,;\,!A & \textit{contraction}\uparrow\,;\,\textit{contraction}\uparrow \\ \cline{1-1}
!A\,;\,!A\,;\,!A\,;\,!A & \diamond \\ \cline{1-1}
!A\,;\,!A & \textit{co-contraction}\downarrow\,;\,\textit{co-contraction}\downarrow
\end{array}$$

La règle structurelle qui a été utilisée dans la partie droite permute les deux formules !A situées au milieu. De façon duale :

$$\begin{array}{cl}
?A\,,\,?A & \\ \cline{1-1}
?A & \textit{contraction}\downarrow \\ \cline{1-1}
?A\,,\,?A & \textit{co-contraction}\uparrow
\end{array}$$

$$\downarrow$$

$$\begin{array}{cl}
?A\,,\,?A & \\ \cline{1-1}
?A\,,\,?A\,,\,?A\,,\,?A & \textit{co-contraction}\uparrow\,,\,\textit{co-contraction}\uparrow \\ \cline{1-1}
?A\,,\,?A\,,\,?A\,,\,?A & \diamond \\ \cline{1-1}
?A\,,\,?A & \textit{contraction}\downarrow\,,\,\textit{contraction}\downarrow
\end{array}$$

Les autres réductions peuvent être obtenues à partir de celles présentées en 7.2.2 pour le système des réseaux d'interaction différentiels. Nous donnerons plus de précisions sur cette correspondance dans la section qui suit (cf. 10.4). On remarque en effet que toutes les réductions des réseaux sont compatibles avec l'information supplémentaire donnée par les structures. Et comme dans notre exemple, chacune peut être orientée de deux façons : l'un des opérateurs est utilisé comme une introduction et l'autre comme élimination. On définit ainsi l'ensemble des réductions logiques dont nous avons besoin.

Notons que les réductions différentielles indéterministes nécessitent de travailler dans un monoïde de dérivations, leur formulation est faite de la même façon que pour les réseaux (voir 7.2.2).

10.3.4 Réduction des identités

Lorsqu'une élimination est face à un axiome, la réduction fait passer cette construction de l'autre côté de l'axiome et elle devient une introduction. Dans le cas d'une coupure se sont les introductions qui se présentent face à elle qui la traversent et qui deviennent des éliminations :

$$\begin{array}{cl}
A\,;\,B\,;\,A^{\perp}\parr B^{\perp} & \\ \cline{1-1}
A\otimes B\,;\,A^{\perp}\parr B^{\perp} & \textit{tenseur}\downarrow\,;\,- \\ \cline{1-1}
\circ & \textit{cut}\uparrow
\end{array}$$

$$\downarrow$$

$$\begin{array}{cl}
A\,;\,B\,;\,A^{\perp}\parr B^{\perp} & \\ \cline{1-1}
A\,;\,B\,;\,(A^{\perp}\,,\,B^{\perp}) & -\,;\,-\,;\,\textit{tenseur}\uparrow \\ \cline{1-1}
(A\,;\,A^{\perp})\,,\,(B\,;\,B^{\perp}) & \diamond \\ \cline{1-1}
\circ\,,\,\circ & \textit{cut}\uparrow\,,\,\textit{cut}\uparrow \\ \cline{1-1}
\circ & \diamond
\end{array}$$

Grâce à ces réductions, on peut simuler l'élimination traditionnelle des coupures de la logique linéaire. Au choix, on peut faire traverser la *co-contraction* ou la *contraction* lorsque celles-ci se présentent de part et d'autre d'une coupure. On obtient dans les deux cas cette réduction :

Voir Fig. 10.2 (page 122).

De même, lorsque ces constructions se présentent liées par un axiome, on parvient à effectuer la réduction suivante : Voir Fig. 10.3 (page 122).

10.3.5 Normalisation

Partant d'une dérivation donnée, les diverses réductions évoquées permettent d'obtenir en un nombre fini d'étapes (cela reste à prouver) une dérivation en forme normale qui a la même hypothèse et la même conclusion. De fait, cette procédure élimine :

- toutes les coupures dans une preuve $\overset{\cdot}{\underset{\tau}{=}}$

En forme normale, une coupure ne peut être précédée que d'éliminations, or celles-ci ne permettent pas d'introduire les formules d'une telle coupure.

- tous les axiomes dans une contre-preuve $\overset{\sigma}{\underset{\circ}{=}}$

Pour la même raison à changement d'orientation près.

- axiomes et coupures dans une dérivation close $\overset{\cdot}{\underset{\circ}{=}}$

Cela montre que de telles dérivations n'existent pas (sauf la dérivation nulle) et assure par conséquent une cohérence de la logique.

$$\begin{array}{cl}
!A\,;\,!A\,;\,(?A^{\perp}\,,\,?A^{\perp}) & \\ \cline{1-1}
!A\,;\,?A^{\perp} & \textit{co-contraction}\downarrow\,;\,\textit{contraction}\downarrow \\ \cline{1-1}
\circ & \textit{cut}\uparrow
\end{array}$$

$$\twoheaddownarrow$$

$$\begin{array}{cl}
!A\,;\,!A\,;\,(?A^{\perp}\,,\,?A^{\perp}) & \\ \cline{1-1}
!A\,;\,!A\,;\,!A\,;\,!A\,;\,(?A^{\perp}\,,\,?A^{\perp}\,,\,?A^{\perp}\,,\,?A^{\perp}) & \textit{contraction}\uparrow\,;\,\textit{contraction}\uparrow\,;\,(\textit{co-contraction}\uparrow\,,\,\textit{co-contraction}\uparrow) \\ \cline{1-1}
(!A\,;\,?A^{\perp})\,,\,(!A\,;\,?A^{\perp})\,,\,(!A\,;\,?A^{\perp})\,,\,(!A\,;\,?A^{\perp}) & \diamond \\ \cline{1-1}
\circ\,,\,\circ\,,\,\circ\,,\,\circ & \textit{cut}\uparrow\,,\,\textit{cut}\uparrow\,,\,\textit{cut}\uparrow\,,\,\textit{cut}\uparrow \\ \cline{1-1}
\circ & \diamond
\end{array}$$

Figure 10.2

$$\begin{array}{cl}
\cdot & \\ \cline{1-1}
?A\,,\,!A^{\perp} & \textit{ax}\downarrow \\ \cline{1-1}
?A\,,\,?A\,,\,(!A^{\perp}\,;\,!A^{\perp}) & \textit{contraction}\uparrow\,;\,\textit{co-contraction}\uparrow
\end{array}$$

$$\twoheaddownarrow$$

$$\begin{array}{cl}
\cdot & \\ \cline{1-1}
\cdot\,;\,\cdot\,;\,\cdot\,;\,\cdot & \diamond \\ \cline{1-1}
(?A\,,\,!A^{\perp})\,;\,(?A\,,\,!A^{\perp})\,;\,(?A\,,\,!A^{\perp})\,;\,(?A\,,\,!A^{\perp}) & \textit{ax}\downarrow\,;\,\textit{ax}\downarrow\,;\,\textit{ax}\downarrow\,;\,\textit{ax}\downarrow \\ \cline{1-1}
?A\,,\,?A\,,\,?A\,,\,?A\,,\,(!A^{\perp}\,;\,!A^{\perp}\,;\,!A^{\perp}\,;\,!A^{\perp}) & \diamond \\ \cline{1-1}
?A\,,\,?A\,,\,(!A^{\perp}\,;\,!A^{\perp}) & \textit{co-contraction}\downarrow\,,\,\textit{co-contraction}\downarrow\,,\,(\textit{contraction}\downarrow\,;\,\textit{contraction}\downarrow)
\end{array}$$

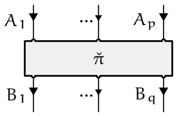
Figure 10.3

10.4 Des réseaux structurés

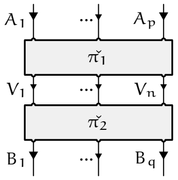
Les dérivations issues de l'« inférence profonde » dont font partie les calculs de structures se traduisent dans différents formalismes à base de graphes [Str09]. Tel qu'il a été présenté, le calcul de structure qu'on considère admet une traduction directe dans les réseaux d'interaction.

10.4.1 Réseau associé à une dérivation

Comme annoncé, une dérivation π de σ vers τ est traduite en un réseau :



où $A_1, \dots, A_p$ désignent les formules contenues dans σ, et $B_1, \dots, B_p$ celles contenues dans τ. Une dérivation qui est composition de deux dérivations, disons π_1 de σ vers ω et π_2 de ω vers τ, se traduit en connectant les réseaux correspondants :

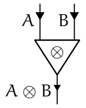


Il ne nous reste plus qu'à donner les traductions des règles élémentaires. Lorsque celles-ci sont utilisées sous un contexte, on ajoutera dans leur traductions un certain nombre de fils qui lient les ports hypothèse associées à ce contexte aux ports conclusion correspondants.

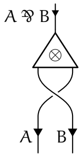
10.4.2 Réseaux associés aux règles élémentaires

Chaque règle logique est traduite par un réseau constitué d'un seul nœud du même nom. Nous prenons l'exemple des règles *tenseur* et *par* (l'orientation haut–bas est significative pour les réseaux donnés).

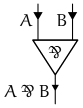
$$\frac{A ; B}{A \otimes B}\ \textit{tenseur}\!\downarrow \qquad \mapsto$$



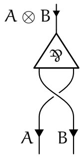
$$\frac{A \parr B}{A , B}\ \textit{tenseur}\!\uparrow \qquad \mapsto$$



$$\frac{A , B}{A \parr B}\ \textit{par}\!\downarrow \qquad \mapsto$$



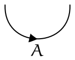
$$\frac{A \otimes B}{A ; B}\ \textit{par}\!\uparrow \qquad \mapsto$$



Les règles du fragment identité sont les seules règles qui changent l'orientation d'un fil :

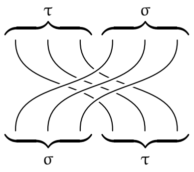
$$\frac{\cdot}{A^{\perp} , A}\ \textit{ax}\!\downarrow \qquad \mapsto$$

$$\frac{A ; A^{\perp}}{\circ}\ \textit{cut}\!\uparrow \qquad \mapsto$$

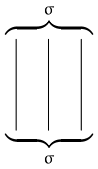


Les règles structurelles sont quant à elles traduites par de simples câblages qui tiennent comptent des permutations (éventuelles) de formules. Par exemple :

$$\frac{\tau ; \sigma}{\sigma ; \tau}\ \textit{échange} \qquad \mapsto$$

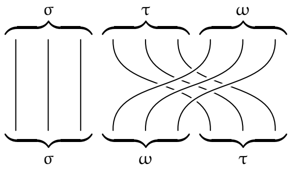


$$\frac{\sigma}{\cdot ; \sigma}\ \textit{neutralité} \qquad \mapsto$$



$$\frac{(\sigma , \tau) ; \omega}{(\sigma ; \omega) , \tau}\ \textit{appariement}$$

$$\mapsto$$



10.4.3 Deux réductions similaires

Une réduction d'un réseau d'interaction qui est issue de cette traduction correspond à une réduction logique dans le calcul originel, plus d'éventuelles commutations qui servent à faire apparaître l'interaction choisie. Une dérivation du calcul des structures que nous avons présenté est donc un objet très proche d'un réseau, celui-ci à simplement été enrichi par une information qui garantit sa validité, et qui contient subsidiairement une information de séquentialité.

10.5 Boîtes

Nous pourrions ajouter dans nos structures une composante qui correspond à la modalité exponentielle comme le fait *Lutz Straßburger* [Str03], mais pour nos besoins nous nous contenterons de constructions globales qui rappellent les boîtes.

10.5.1 Les boîtes traditionnelles de la logique linéaire

La *promotion* peut être introduite dans ce calcul, c'est une règle qui est elle-même paramétrée par une dérivation.

$$\frac{\begin{array}{c} !V_1 ; \cdots ; !V_k \\ \boxed{\begin{array}{c} !V_1 ; \cdots ; !V_k \\ \hline \vdots \\ \hline A \end{array}} \end{array}}{!A}\ \textit{promotion}\!\downarrow$$

$$\frac{?A}{\begin{array}{c} \boxed{\begin{array}{c} A \\ \hline \vdots \\ \hline ?V_1 , \cdots , ?V_k \end{array}} \\ ?V_1 , \cdots , ?V_k \end{array}}\ \textit{promotion}\!\uparrow$$

Pour qu'une dérivation soit considérée en forme normale il faudra que les dérivations qui paramètrent ces « boîtes » soient également en forme normale. Notons aussi que la dynamique de ce calcul fait qu'il n'est pas naturel de considérer des réductions au niveau des entrées de boîtes, bien qu'on puisse les écrire.

Et au passage, il n'y a pas plus de difficultés pour introduire les constructions additives.

$$\frac{A}{A \oplus B}\ \textit{plus gauche}\!\downarrow \qquad \frac{A \mathbin{\&} B}{A}\ \textit{plus gauche}\!\uparrow$$

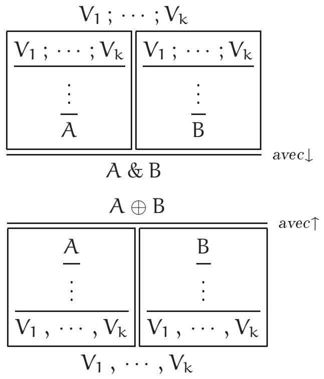
$$\frac{B}{A \oplus B}\ \textit{plus droit}\!\downarrow \qquad \frac{A \mathbin{\&} B}{B}\ \textit{plus droit}\!\uparrow$$

$$\frac{\begin{array}{c} \omega \\ \boxed{\begin{array}{c} \omega \\ \hline \vdots \\ \hline A \end{array}}\ \boxed{\begin{array}{c} \omega \\ \hline \vdots \\ \hline B \end{array}} \end{array}}{A \mathbin{\&} B}\ \textit{avec}\!\downarrow \qquad \frac{A \oplus B}{\begin{array}{c} \boxed{\begin{array}{c} A \\ \hline \vdots \\ \hline \omega \end{array}}\ \boxed{\begin{array}{c} B \\ \hline \vdots \\ \hline \omega \end{array}} \\ \omega \end{array}}\ \textit{avec}\!\uparrow$$

et :

$$\frac{\omega}{\top}\ \textit{top}\!\downarrow \qquad \frac{0}{\omega}\ \textit{top}\!\uparrow$$

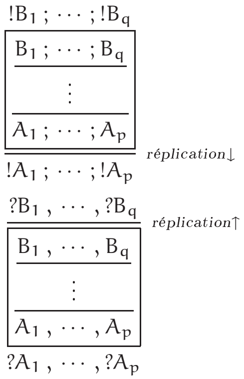
Celles-ci sont d'ailleurs un petit peu plus flexibles que celles utilisées lorsqu'on considère des séquents. Ces dernières correspondent aux restrictions suivantes :



$$\frac{V_1\,;\cdots;V_k}{\top}\ \textit{top}\!\downarrow \qquad\qquad \frac{0}{V_1\,,\cdots,V_k}\ \textit{top}\!\uparrow$$

10.5.2 Réplication

La *réplication* est introduite comme ci-dessous, c'est une règle qu'on ne pouvait pas écrire avec des séquents :



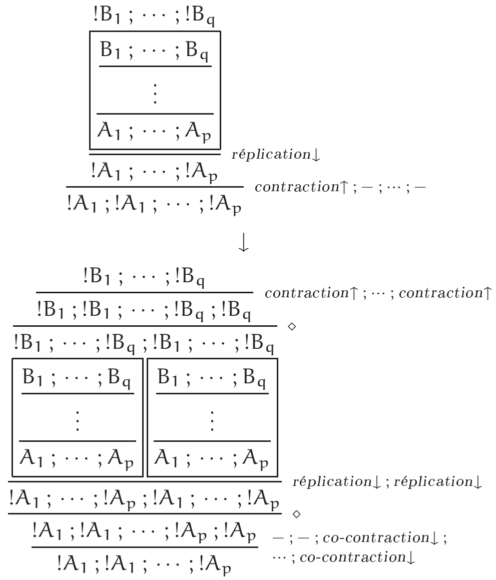
On n'oublie pas de lui adjoindre les *enfouissements* et *co-enfouissements*.

$$\frac{!!A}{!A}\ \textit{co-enfouissement}\!\downarrow \qquad \frac{??A}{?A}\ \textit{enfouissement}\!\downarrow$$

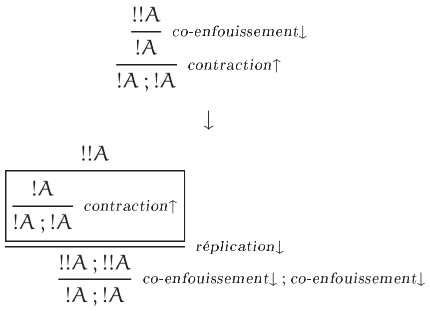
$$\frac{!A}{!!A}\ \textit{enfouissement}\!\uparrow \qquad \frac{?A}{??A}\ \textit{co-enfouissement}\!\uparrow$$

Les règles de réduction des réseaux s'écrivent alors dans ce formalisme, et grâce à celui-ci il est naturel de savoir si un réseau est bien formé.

Nous donnerons deux exemples de réductions pour ces constructions, écrites dans le calcul des structures. Tout d'abord, la réduction qui duplique une boîte.

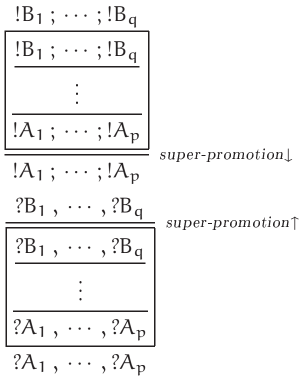


Ensuite, la réduction qui fait apparaître une boîte qu'on ne peut écrire avec des séquents classiques.

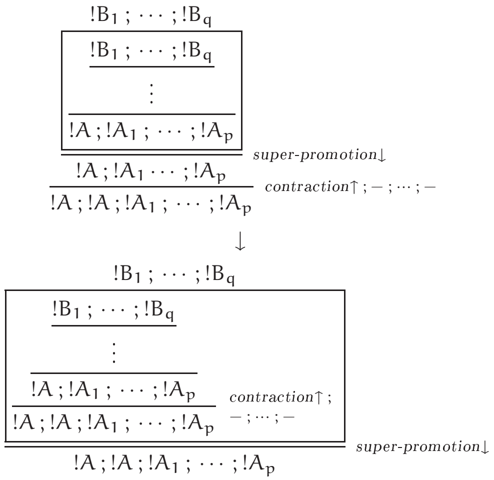


10.5.3 Super-promotion

On peut préférer n'introduire que la règle de *super-promotion*. Étant donné qu'on n'effectue pas de réduction sur les entrées de boîtes dans ce calcul, cela évite l'intervention de la réduction « explosive ». Elle est introduite comme ceci :



Et pour donner un exemple de réduction, voici comment celle-ci interagit avec une *contraction* :



10.6 Bilan

Ce chapitre présente un travail en cours, et les idées présentées nécessiteraient de plus amples justifications. Il fournit néanmoins un éclaircissement qui nous a longtemps fait défaut sur les systèmes différentiels avec boîtes de *réplication* ou de *super-promotion*. Ces systèmes ont été élaborés principalement à l'aide de leurs interprétations sémantiques, celles-ci ayant un sens même pour des réseaux mal formés, mais nous sommes contents de retrouver le sens logique qui se cachait derrière ces nouvelles constructions.

Le système logique que nous avons présenté utilise des structures et offre plus de flexibilité que les systèmes logiques fondés sur des séquents. En effet, on peut exprimer aisément une règle du calcul des séquents telle que :

$$\frac{\vdash B_1,\ldots,B_p \quad \vdash C_1,\ldots,C_q}{\vdash A_1,\ldots,A_n}$$

à l'aide d'une règle de ce calcul de structures :

$$\frac{(B_1\,,\cdots,B_p)\,;(C_1\,,\cdots,C_q)}{A_1\,,\cdots,A_n}$$

Mais on est incapable d'écrire dans le calcul des séquents des règles qui utilisent des connecteurs ‹ ; › dans leur conclusion, ou même dans les hypothèses s'ils sont utilisés en profondeur. Les règles de *réplication* ou de *super-promotion* ne peuvent être écrites sans cette flexibilité.

Troisième partie

Compléments

Chapitre 11

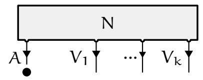
Réalisabilité

Des méthodes de réalisabilité permettent de montrer certains résultats sur les systèmes de réseaux qui ont été présentés dans les chapitres précédents. Nous décrirons dans un premier temps la preuve de normalisation des réseaux de preuve de la logique linéaire que donne *Jean-Yves Girard* [Gir87] en adaptant les méthodes de preuve par réalisabilité au formalisme des réseaux. Nous donnerons ensuite une extension de cette preuve pour le formalisme différentiel.

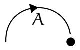
11.1 Outils de réalisabilité

11.1.1 Observable et orthogonalité

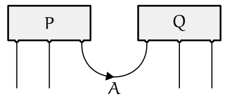
Définition 11-1. *Un* réseau pointé *est un réseau dont l'un des ports est distingué.* $\mathcal{N}_{\mathsf{A}}$ *désignera l'ensemble des réseaux pointés dont le port en question est de type* A.



On notera $Axiome_{\mathsf{A}}$ l'axiome pointé de type A :



Et, lorsque $\mathsf{P} \in \mathcal{N}_{\mathsf{A}}$ et $\mathsf{Q} \in \mathcal{N}_{\mathsf{A}^\perp}$, on notera $Coupure_{\mathsf{A}}(\mathsf{P},\mathsf{Q})$ la coupure de P et Q selon leur port distingué :



On se donne un ensemble $\mathcal{O}$ de réseaux dit *observable*, et on définit une relation binaire $\star$ telle que $\mathsf{P}\star\mathsf{Q}$ si et seulement si pour tous réseaux pointés $\mathsf{P} \in \mathcal{N}_{\mathsf{A}}$ et $\mathsf{Q} \in \mathcal{N}_{\mathsf{A}^\perp}$ de types duaux on a $Coupure_{\mathsf{A}}(\mathsf{P},\mathsf{Q}) \in \mathcal{O}$.

Définition 11-2. *Si* $\mathbf{P}$ *désigne un ensemble de réseaux pointés de type* A, *on définit son orthogonal ainsi :*

$$\mathbf{P}^\star \;:=\; \{\, \mathsf{Q} \mid \forall \mathsf{P} \in \mathbf{P},\; \mathsf{P} \star \mathsf{Q} \,\}$$

Propriété 11-3. *Pour tout ensembles de réseaux pointés* $\mathbf{P} \subseteq \mathcal{N}_{\mathsf{A}}$ *et* $\mathbf{Q} \subseteq \mathcal{N}_{\mathsf{A}^\perp}$ *:*

$$\mathbf{P} \subseteq \mathbf{Q} \;\Rightarrow\; \mathbf{Q}^\star \subseteq \mathbf{P}^\star \qquad \mathbf{P} \subseteq \mathbf{P}^{\star\star} \qquad \mathbf{P}^{\star\star\star} = \mathbf{P}^\star$$

Preuve. Les deux premières assertions découlent sans difficultés de la définition. Pour la troisième, l'inclusion $\mathbf{P}^{\star\star\star} \subseteq \mathbf{P}^\star$ s'obtient en remplaçant $\mathbf{P}$ par $\mathbf{P}^\star$ dans la seconde, et l'inclusion $\mathbf{P}^{\star\star\star} \supseteq \mathbf{P}^\star$ s'obtient en remplaçant $\mathbf{Q}$ par $\mathbf{P}^{\star\star}$ dans la première. □

On notera $\mathcal{O}_{\mathsf{A}}$ l'observable au type A : l'ensemble constitué des réseaux de $\mathcal{O}$ pointés sur l'un de leurs liens de type A. Son orthogonal ${\mathcal{O}_{\mathsf{A}}}^\star$ sera appelé *inobservable* au type A.

Propriété 11-4. *De façon remarquable on a :*

$$\mathcal{O}_{\mathsf{A}} \;=\; \{\, Axiome_{\mathsf{A}^\perp} \,\}^\star \;=\; {\mathcal{O}_{\mathsf{A}}}^{\star\star}$$

Preuve. On parvient à montrer cela simplement, parce qu'on est dans un formalisme de réseaux où $Coupure_{\mathsf{A}}(\mathsf{N}, Axiome_{\mathsf{A}^\perp}) = \mathsf{N}$ pour tout N (cf. remarque 3-2). Cela nous donne immédiatement la première égalité, et on en déduit la seconde car $\{\, Axiome_{\mathsf{A}^\perp} \,\}^{\star\star\star} = \{\, Axiome_{\mathsf{A}^\perp} \,\}^\star$. □

11.1.2 Comportements

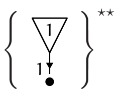
Définition 11-5. *On appellera* comportement *de type* A *un ensemble de réseaux pointés de type* A *clos par double orthogonalité.*

$$\mathbf{P} \;=\; \mathbf{P}^{\star\star}$$

Il est possible de définir certains comportements remarquables ainsi que des opérations sur les comportements. On notera que les comportements ainsi définis sont bien sûr dépendants de l'observation $\mathcal{O}$ qui a été choisie.

Multiplicatifs

Le comportement $\mathbf{1} \subseteq \mathcal{N}_{\mathsf{1}}$ est défini par extension observationelle de l'unique construction associée au connecteur logique correspondant :

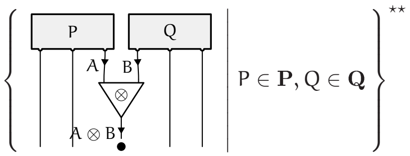
$\mathbf{1} :=$


Le comportement $\bot \subseteq \mathcal{N}_\bot$ sera lui défini comme l'orthogonal de ce premier :

$$\bot := \mathbf{1}^\star$$

Pour des comportements $\mathbf{P} \subseteq \mathcal{N}_A$ et $\mathbf{Q} \subseteq \mathcal{N}_B$, la définition du comportement $\mathbf{P} \otimes \mathbf{Q} \subseteq \mathcal{N}_{A\otimes B}$ est faite sur le même principe :

$$\mathbf{P} \otimes \mathbf{Q} :=$$

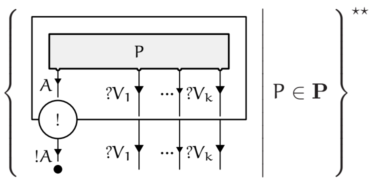


On définit le comportement $\mathbf{P} \parr \mathbf{Q} \subseteq \mathcal{N}_{A\parr B}$ par dualité :

$$\mathbf{P} \parr \mathbf{Q} := (\mathbf{P}^\star \otimes \mathbf{Q}^\star)^\star$$

Exponentiels

Pour un comportement $\mathbf{P} \subseteq \mathcal{N}_A$, on définit le comportement $!\mathbf{P} \subseteq \mathcal{N}_{!A}$ par extension observationnelle de l'unique construction syntaxique associée à l'exponentielle :

$!\mathbf{P} :=$


On définit le comportement $?\mathbf{P} \subseteq \mathcal{N}_{?A}$ par dualité :

$$?\mathbf{P} := (!\mathbf{P}^\star)^\star$$

Additifs

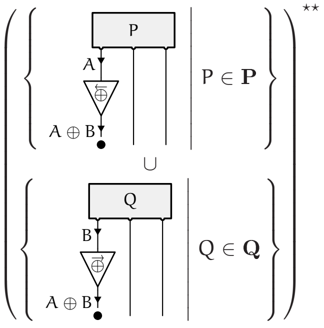
Le comportement $\mathbb{0} \subseteq \mathcal{N}_0$ est $\mathcal{O}_0{}^\star$, l'inobservable au type 0 :

$$\mathbb{0} := \{\mathit{Axiome}_0\}^{\star\star}$$

Le comportement $\top \subseteq \mathcal{N}_\top$ est $\mathcal{O}_0$, l'observable au type $\top$, qui est orthogonal au précédent :

$$\top := \mathbb{0}^\star$$

Pour $\mathbf{P} \subseteq \mathcal{N}_A$ et $\mathbf{Q} \subseteq \mathcal{N}_B$, on définit le comportement $\mathbf{P} \oplus \mathbf{Q} \subseteq \mathcal{N}_{A\oplus B}$ par extension observationelle des constructions syntaxiques associées aux additifs :

$\mathbf{P} \oplus \mathbf{Q} :=$


On définit le comportement $\mathbf{P} \,\&\, \mathbf{Q} \subseteq \mathcal{N}_{A\& B}$ par dualité :

$$\mathbf{P} \,\&\, \mathbf{Q} := (\mathbf{P}^\star \oplus \mathbf{Q}^\star)^\star$$

11.1.3 Candidats de réductibilité

Définition 11-6. *On appellera* candidat de réductibilité *de type* A *un comportement de type* A *compris entre l'inobservable et l'observable au type* A *:*

$$\mathcal{O}_A{}^\star \subseteq \mathbf{P} = \mathbf{P}^{\star\star} \subseteq \mathcal{O}_A$$

L'ensemble des candidats de réductibilité de type A *sera noté* $\mathfrak{C}_A$.

En particulier, le dual $\mathbf{P}^\star$ d'un candidat de réductibilité $\mathbf{P}$ est aussi un candidat de réductibilité.

On dira que qu'une observable est *compatible* avec la composition des comportements lorsque les opérations définies précédemment (11.1.2) sur les comportements transforment des candidats de réductibilité en candidats de réductibilité.

Interprétation des formules

Lorsque l'observable est compatible, on va pouvoir associer inductivement à chaque formule des candidats de réductibilité en partant de candidats de réductibilité donnés pour chaque formule atomique.

En fait, le candidat Ξ_X choisi pour chacune des variables X n'importe pas. L'induction est faite de la façon suivante :

$$\begin{array}{rclcrcl}
[\![X]\!] & := & \Xi_X & \qquad & [\![\bar{X}]\!] & := & \Xi_X^\star \\
[\![1]\!] & := & \mathbf{1} & & [\![\bot]\!] & := & \bot \\
[\![A \otimes B]\!] & := & [\![A]\!] \otimes [\![B]\!] & & [\![A \parr B]\!] & := & [\![A]\!] \parr [\![B]\!] \\
[\![!A]\!] & := & ![\![A]\!] & & [\![?A]\!] & := & ?[\![A]\!] \\
[\![0]\!] & := & \mathbb{0} & & [\![\top]\!] & := & \top \\
[\![A \oplus B]\!] & := & [\![A]\!] \oplus [\![B]\!] & & [\![A \,\&\, B]\!] & := & [\![A]\!] \,\&\, [\![B]\!]
\end{array}$$

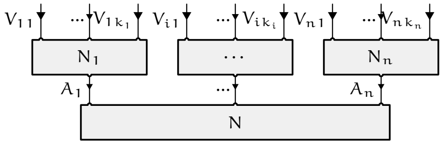
Et on a bien sûr $[\![A^\perp]\!] = [\![A]\!]^\star$.

11.2 Normalisation faible

On choisit comme observable $\mathcal{O}$ l'ensemble des réseaux faiblement normalisants : lorsque $P \in \mathcal{N}_A$ et $Q \in \mathcal{N}_{A^\perp}$ alors $P \star Q$ si et seulement si $\mathit{Coupure}_A(P, Q)$ admet une forme normale. Cette observation est compatible avec la composition des comportements, on associe donc à chaque formule A un candidat de réductibilité $[\![A]\!]$.

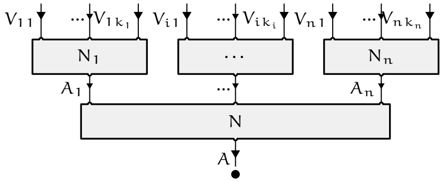
Définition 11-7. *Un réseau* N *est dit réductible lorsque de façon équivalente :*

(i) N *est d'interface* $\vdash A_1{}^\perp, \ldots, A_n{}^\perp$ *et pour toute famille de réseaux* $(N_i)_{i\in[\![1,n]\!]}$ *tels que* $N_i \in [\![A_i]\!]$, *on a :*



$$\in \mathcal{O}$$

(ii) N *est d'interface* $A_1, \ldots, A_n \vdash A$ *et pour toute famille de réseaux* $(N_i)_{i\in[\![1,n]\!]}$ *tels que* $N_i \in [\![A_i]\!]$, *on a :*



$$\in [\![A]\!]$$

Les deux réseaux présentés seront notés $\mathit{Coupures}(N; N_1, \ldots, N_n)$. *Lorsque* N *désigne un réseau pointé on sera dans le second cas : on ne branche pas de réseau sur le port pointé.*

Le lemme suivant, appelé lemme d'interprétation ou lemme d'adéquation, permet de prouver un théorème de normalisation qui stipule que tout réseau admet une forme normale.

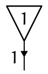
Lemme 11-8. *Tout réseau est réductible.*

Preuve. On raisonne par induction sur le réseau N considéré :

(Axiome) On doit montrer que pour tout $N_1 \in [\![A]\!]$ et $N_2 \in [\![A^\perp]\!]$, $N' = \mathit{Coupures}(\mathit{Axiome}_A; N_1, N_2)$ normalise. Or ce réseau n'est autre que $\mathit{Coupure}_A(N_1, N_2)$, qui normalise car, puisque $[\![A^\perp]\!] = [\![A]\!]^\star$, on a $N_1 \star N_2$.

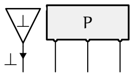
(Coupure) On doit montrer que pour tout $R_1 \in [\![A]\!]$ et $R_2 \in [\![A^\perp]\!]$ réductibles, le réseau $N' = \mathit{Coupures}(\mathit{Coupure}(R_1, R_2); N_i)$ normalise. La quantification sur les réseaux pointés N_i est universelle, on les suppose dans leurs interprétations correspondantes : $N_i \in [\![A_i]\!]$ (ce type de quantification sera réutilisé et restera implicite dans le reste de la preuve). Or, le fait que $\mathit{Coupure}(R_1, N_i) \in [\![A]\!]$ et $\mathit{Coupure}(R_2, N_i) \in [\![A^\perp]\!]$ justifie cette normalisation.

(Unité) On suppose que N est de la forme suivante :



On note $\mathit{Unité}_\bullet$ le réseau pointé correspondant. La réductibilité de N s'écrit $\mathit{Unité}_\bullet \in [\![1]\!]$, or cela est vérifié par définition.

(Co-unité) On suppose que N est de la forme suivante, avec comme hypothèse d'induction la réductibilité de P :

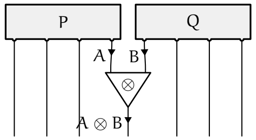


Lorsqu'on nomme $N^\bullet$ le réseau pointé correspondant, la réductibilité du réseau N s'écrit $\mathit{Coupures}(N^\bullet; N_i) \in \llbracket\bot\rrbracket$. Étant donné que :

$$\llbracket\bot\rrbracket \;=\; \{\,\mathit{Unité}\,\}^\star$$

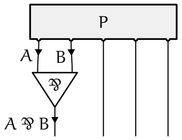
cette assertion est vérifiée lorsque le réseau $\mathit{Coupures}(N; \mathit{Unité}, N_i)$ normalise. Or ce réseau se réduit en $\mathit{Coupures}(P; N_i)$ qui normalise par réductibilité de P.

(Tenseur) On suppose que N est de la forme suivante avec pour hypothèse d'induction la réductibilité de P et Q :



On note $\mathit{Tenseur}_\bullet(P, Q)$ le réseau pointé correspondant. La réductibilité du réseau N s'écrit $\mathit{Coupures}(\mathit{Tenseur}_\bullet(P, Q), N_i) \in \llbracket A \otimes B\rrbracket$, qui est vérifié d'après la définition de $\llbracket A \otimes B\rrbracket$ car $\mathit{Coupures}(P; N_i) \in \llbracket A\rrbracket$ et $\mathit{Coupures}(Q; N_i) \in \llbracket B\rrbracket$ par réductibilité de P et Q.

(Par) On suppose que N est de la forme suivante, avec pour hypothèse d'induction la réductibilité de P :

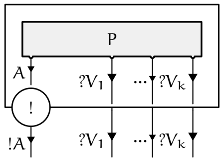


La réductibilité du réseau N s'écrit $\mathit{Coupures}(N^\bullet; N_i) \in \llbracket A \parr B\rrbracket$. Or, étant donné que :

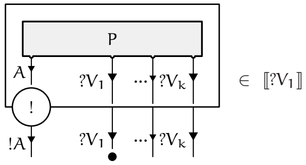
$$\llbracket A \parr B\rrbracket \;=\; \left\{\,\mathit{Tenseur}_\bullet(R_1, R_2) \;\middle|\; R_1 \in \llbracket A^\perp\rrbracket, R_2 \in \llbracket B^\perp\rrbracket\,\right\}^\star$$

cette assertion est vérifiée lorsque pour tout $R_1 \in \llbracket A^\perp\rrbracket$ et $R_2 \in \llbracket B^\perp\rrbracket$, le réseau $\mathit{Coupures}(N; \mathit{Tenseur}_\bullet(R_1, R_2), N_i)$ normalise. Mais ce dernier se réduit en $\mathit{Coupures}(P; R_1, R_2, N_i)$ qui normalise par réductibilité de P.

(Promotion) On suppose que N est de la forme suivante, avec pour hypothèse d'induction la réductibilité de P :



On note ce réseau $\mathit{Promotion}(P)$. Sa réductibilité s'écrit : pour tout $N_0 \in \llbracket !A\rrbracket$, pour tout $N_i \in \llbracket ?V_i\rrbracket^\star$, $\mathit{Coupures}(\mathit{Promotion}(P); N_0, N_i) \in \mathcal{O}$. Mais cela est vérifié si pour tout $N_0 \in \llbracket !A\rrbracket$ et pour tout $N_i \in \llbracket ?V_i\rrbracket^\star$ $(i > 1)$ on a :



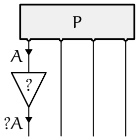
Or, comme :

$$\llbracket ?V_k\rrbracket \;=\; \left\{\,\mathit{Promotion}_\bullet(R) \;\middle|\; R \in \llbracket {V_k}^\perp\rrbracket\,\right\}^\star$$

on est ramenés à montrer comme avant que $\mathit{Coupures}(\mathit{Promotion}(P); N_0, N_i) \in \mathcal{O}$, mais cette fois pour $N_1 \in \left\{\,\mathit{Promotion}_\bullet(R) \;\middle|\; R \in \llbracket {V_1}^\perp\rrbracket\,\right\}$ uniquement. Soit $N_1 = \mathit{Promotion}_\bullet(R_1)$ un réseau de cet ensemble (on a $R_1 \in \llbracket {V_1}^\perp\rrbracket$). On considère alors les réseaux pointés sur le second lien, et en itérant le procédé on se ramène au cas où les N_i sont de la forme $N_i = \mathit{Promotion}_\bullet(R_i)$ pour $R_i \in \llbracket {V_1}^\perp\rrbracket$. Il suffit au final de montrer que $\mathit{Coupures}(\mathit{Promotion}_\bullet(P); \mathit{Promotion}(R_i;)) \in \llbracket !A\rrbracket$.

Mais ce réseau se réduit par commutation exponentielle en $\mathit{Promotion}_\bullet(\mathit{Coupures}(P; R_i))$. Et P étant réductible par hypothèse d'induction, comme $R_i \in \llbracket {V_i}^\perp\rrbracket$, on a $\mathit{Coupures}(P; R_i) \in \llbracket A\rrbracket$. On obtient par conséquent $\mathit{Promotion}_\bullet(\mathit{Coupures}(P; R_i)) \in \llbracket !A\rrbracket$.

(Déréliction) On suppose que N est de la forme suivante, avec comme hypothèse d'induction la réductibilité de P :



La réductibilité du réseau N s'écrit $\mathit{Coupures}(N^\bullet; N_i) \in \llbracket ?A\rrbracket$. Étant donné que :

$$\llbracket ?A\rrbracket \;=\; \left\{\,\mathit{Promotion}_\bullet(R) \;\middle|\; R \in \llbracket A^\perp\rrbracket\,\right\}^\star$$

cette assertion est vérifiée si pour tout $R \in \llbracket A^\perp\rrbracket$, le réseau $\mathit{Coupures}(N; \mathit{Promotion}_\bullet(R), N_i)$ normalise. Or ce réseau se réduit en $\mathit{Coupures}(P; R, N_i)$, qui normalise par réductibilité de P.

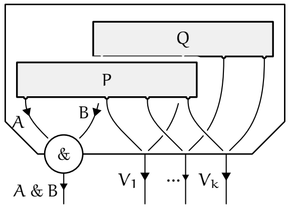
(Affaiblissement) Idem, le réseau se réduit en $\mathit{Coupures}(P; N_i)$ plus des *affaiblissements*.

(Contraction) Idem, le réseau se réduit en $\mathit{Coupures}(P; \mathit{Promotion}_\bullet(R), \mathit{Promotion}_\bullet(R), N_i)$ plus des *contractions*.

(Plus gauche) On suppose $N = \mathit{Plus}^{\leftarrow}(P)$ avec pour hypothèse d'induction la réductibilité de P. La réductibilité du réseau N s'écrit $\mathit{Coupures}(\mathit{Plus}_\bullet^{\leftarrow}(P); N_i) \in \llbracket A \oplus B\rrbracket$, qui est vérifié d'après la définition de $\llbracket A \oplus B\rrbracket$ car $\mathit{Coupures}(P; N_i) \in \llbracket A\rrbracket$ par réductibilité de P.

(Plus droit) Idem.

(Avec) On suppose que N est de la forme suivante, avec pour hypothèse d'induction la réductibilité de P et Q :



La réductibilité du réseau N s'écrit $\mathit{Coupures}(N; N_i) \in \llbracket A \,\&\, B\rrbracket$. Étant donné que :

$$\llbracket A \,\&\, B\rrbracket \;=\; \begin{pmatrix} \left\{\,\mathit{Plus}_\bullet^{\rightarrow}(R_1) \;\middle|\; R_1 \in \llbracket A^\perp\rrbracket\,\right\} \\ \cup \\ \left\{\,\mathit{Plus}_\bullet^{\leftarrow}(R_2) \;\middle|\; R_2 \in \llbracket B^\perp\rrbracket\,\right\} \end{pmatrix}^\star$$

cette assertion est vérifiée si pour tout $R_1 \in \llbracket A^\perp\rrbracket$, le réseau $\mathit{Coupures}(N; \mathit{Plus}_\bullet^{\leftarrow}(R_1), N_i)$ normalise, et pour tout $R_2 \in \llbracket B^\perp\rrbracket$, le réseau $\mathit{Coupures}(N; \mathit{Plus}_\bullet^{\rightarrow}(R_1), N_i)$ normalise. Or ces réseaux se réduisent respectivement en $\mathit{Coupures}(P; R_1, N_i)$ et $\mathit{Coupures}(Q)R_2, N_i$ qui normalisent par réductibilité de P et Q.

(Top) On suppose que N est une boîte *top*. La réductibilité du réseau N s'écrit $\mathit{Coupures}(\mathit{Top}_\bullet, N_i) \in \llbracket\top\rrbracket$. Étant donné que :

$$\llbracket\top\rrbracket \;=\; \{\,\mathit{Axiome}_0\,\}^\star$$

cette assertion est vérifiée si le réseau $\mathit{Coupures}(N, N_i)$ normalise. Or ce réseau se réduit par commutation additive en *Top* qui est normalisé.

Théorème 11-9. *La réduction des réseaux est faiblement normalisante.*

Preuve. C'est un corollaire du lemme précédent : tout réseau N est réductible, et on instancie les réseaux N_i par des axiomes. □

11.3 Brève présentation du second ordre

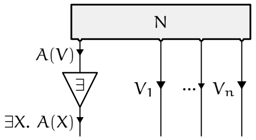
L'intérêt majeur d'une preuve par réalisabilité est de traiter facilement les quantifications du second ordre. Nous n'avons pas encore introduit les constructions correspondantes. Brièvement, le second ordre est introduit dans la logique linéaire par deux nouvelles façons de construire des formules à partir d'une formule A à une indéterminée :

$$\exists X.\, A(X) \qquad\qquad \forall X.\, A(X)$$

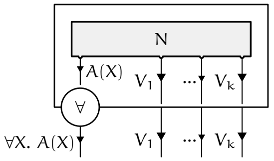
La dualité entre ces deux formules s'écrit $(\exists X.\, A(X))^\perp = \forall X.\, A^\perp(X)$, et on peut former des preuves qui les manipulent grâce à deux constructions.

11.3.1 Constructions pour les quantifications du second ordre

Existentielle Cette construction cache simplement le type V qui a permis de construire un élément de type $A(V)$.

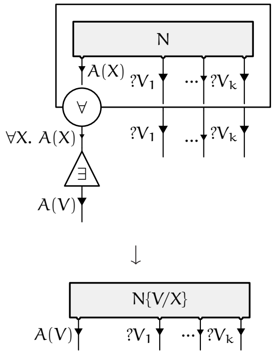


Universelle Une boîte universelle généralise l'interface d'un réseau N : celui-ci pourra être utilisé de façon polymorphe en la variable de type X. Pour que cela ait un sens il faut supposer que cette variable type apparaît seulement dans le type $A(X)$ d'un lien donné de l'interface de N. Ici, on suppose $X \notin V_i$.



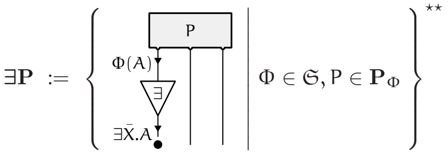
Réduction

Ces deux constructions n'ont pas de vraie influence sur le calcul, elle permettent seulement d'assouplir le typage. La règle de réduction correspondante fait juste en sorte d'oublier la généralité permise par l'utilisation d'une variable X lorsque l'interaction veut qu'elle désigne un type V :



11.3.2 Comportements associés au second ordre

Étant donnée une formule A, un ensemble de substitutions $\mathfrak{S} \subseteq \Lambda^{\bar{X}}$, et une famille $\{\, \mathbf{P}_\Phi \,\}_{\Phi \in \mathfrak{S}}$ de comportements pour les types respectifs $\{\, \Phi(A) \,\}_{\Phi \in \mathfrak{S}}$, on définit le comportement $\exists\, \mathbf{P} \subseteq \mathcal{N}_{\exists \bar{X}.A}$ par extension observationelle de la construction existentielle :



Étant donnée une famille $\mathbf{P}_\Phi \subseteq \mathcal{N}_{\Phi(A)}$, on définit $\forall\mathbf{P} \subseteq \mathcal{N}_{\forall \bar{X}.A}$ par dualité :

$$\forall\mathbf{P} \;:=\; (\exists\mathbf{P}^\star)^\star$$

11.3.3 Interprétation des formules avec substitution

L'interprétation d'une formule doit être faite de façon un peu plus subtile.

Si Σ désigne un ensemble de formules atomiques, et Φ une famille de formules indexée par Σ, on notera $\mathfrak{C}^\Sigma_\Phi$ l'ensemble des familles de candidats de réductibilité $\{\, \Xi_X \,\}_{X\in\Sigma}$ ayant pour types respectifs $\{\, \Phi_X \,\}_{X\in\Sigma}$. Pour chaque formule A construite sur un ensemble de formules atomiques Σ on définit alors une interprétation :

$$\begin{array}{rcccc} \llbracket A \rrbracket & : & \mathfrak{C}^{\Sigma_A}_\Phi & \longrightarrow & \mathfrak{C}_{\Phi(A)} \\ & & \Xi & \mapsto & \llbracket A \rrbracket_\Xi \end{array}$$

L'induction est faite de la façon suivante :

$$\begin{array}{rclcrcl} \llbracket X \rrbracket_\Xi & := & \Xi_X & & \llbracket \bar{X} \rrbracket_\Xi & := & \Xi^\star_X \\ \llbracket 1 \rrbracket_\Xi & := & 1 & & \llbracket \bot \rrbracket_\Xi & := & \bot \end{array}$$

$$\begin{array}{rcl} \llbracket A \otimes B \rrbracket_\Xi & := & \llbracket A \rrbracket_\Xi \otimes \llbracket B \rrbracket_\Xi \\ \llbracket A \parr B \rrbracket_\Xi & := & \llbracket A \rrbracket_\Xi \parr \llbracket B \rrbracket_\Xi \end{array}$$

$$\begin{array}{rclcrcl} \llbracket !A \rrbracket_\Xi & := & !\llbracket A \rrbracket_\Xi & & \llbracket ?A \rrbracket_\Xi & := & ?\llbracket A \rrbracket_\Xi \\ \llbracket 0 \rrbracket_\Xi & := & 0 & & \llbracket \top \rrbracket_\Xi & := & \top \end{array}$$

$$\begin{array}{rcl} \llbracket A \oplus B \rrbracket_\Xi & := & \llbracket A \rrbracket_\Xi \oplus \llbracket B \rrbracket_\Xi \\ \llbracket A \,\&\, B \rrbracket_\Xi & := & \llbracket A \rrbracket_\Xi \,\&\, \llbracket B \rrbracket_\Xi \end{array}$$

Enfin, lorsque la formule est quantifiée :

$$\llbracket \exists \bar{X}.A \rrbracket_\Xi \;:=\; \exists \left\{ \bigcup_{\Theta \in \mathfrak{C}^{\bar{X}}_\Phi} \llbracket A \rrbracket \right\}_{\Phi \in \Lambda^{\bar{X}}}$$

$$\llbracket \forall \bar{X}.A \rrbracket_\Xi \;:=\; \forall \left\{ \bigcap_{\Theta \in \mathfrak{C}^{\bar{X}}_\Phi} \llbracket A \rrbracket \right\}_{\Phi \in \Lambda^{\bar{X}}}$$

Substitution

Étant donnée une famille de candidats de réductibilité Ξ, on définit par extension l'interprétation $\llbracket \Phi \rrbracket_\Xi$ d'une famille de formules $\Phi = \{\, \Phi_X \,\}_{X\in\Sigma}$ comme la famille des interprétations $\{\, \llbracket \Phi_X \rrbracket_\Xi \,\}_{X\in\Sigma}$.

Lemme 11-10. *Soit* A *une formule, et* Φ *une substitution des formules atomiques de* A.

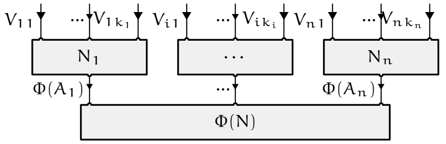
$$\llbracket \Phi(A) \rrbracket_\Xi \;=\; \llbracket A \rrbracket_{\Xi \circ \llbracket \Phi \rrbracket_\Xi}$$

Preuve. Cela provient simplement de la définition des comportements. □

11.3.4 Réductibilité avec instanciation des variables de type

La définition de la réductibilité est également un peu modifiée.

Définition 11-11. *Un réseau* N *d'interface* $\vdash A_1{}^\perp, \ldots, A_n{}^\perp$ *est dit réductible lorsque pour toute famille de réseaux* $(N_i)_{i\in\llbracket 1,n\rrbracket}$ *tels que* $N_i \in \llbracket \Phi(A_i) \rrbracket_\Xi$*, on a :*



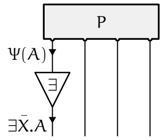
$$\in \;\; \mathcal{O}$$

On peut trouver une formulation équivalente de cette définition, similaire à celle donnée dans le second point de la définition 11-7. Et le théorème principal fonctionne toujours.

Théorème 11-12. *Tout réseau est réductible.*

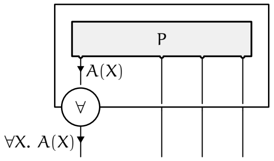
Preuve. On traite les deux nouveaux cas.

(Existentielle) On suppose N est de la forme suivante, avec pour hypothèse d'induction la réductibilité de P :



On notera $\mathit{Exist}^\Psi_\bullet(P)$ le réseau pointé correspondant. La réductibilité du réseau N s'écrit $\mathit{Coupures}(\mathit{Exist}^\Psi_\bullet(P); N_i) \in \llbracket \exists\bar{X}.A \rrbracket_\Xi$, qui est vérifié d'après la définition de $\exists \left\{ \bigcup_{\Theta\in\mathfrak{C}^{\bar{X}}_\Phi} \llbracket A \rrbracket_{\Xi\circ\Theta} \right\}_{\Phi\in\Lambda^{\bar{X}}}$ en prenant $\Phi = \Psi$ et $\Theta = \llbracket \Psi \rrbracket_\Xi$ car $\mathit{Coupures}(P^\bullet; N_i) \in \llbracket \Psi(A) \rrbracket_\Xi$ par réductibilité de P, mais $\llbracket \Psi(A) \rrbracket_\Xi = \llbracket A \rrbracket_{\Xi\circ\llbracket\Psi\rrbracket_\Xi}$ par lemme 11-10 de substitution.

(Universelle) On suppose que N est de la forme suivante, avec pour hypothèse d'induction la réductibilité de P :



La réductibilité du réseau N s'écrit $\mathit{Coupures}(N^\bullet; N_i) \in \llbracket \forall\bar{X}.A \rrbracket_\Xi$. Étant donné que :

$$\forall \left\{ \bigcup_{\Theta\in\mathfrak{C}^{\bar{X}}_\Phi} \llbracket A \rrbracket \right\}_{\Phi\in\Lambda^{\bar{X}}} \;=\; \left\{ \mathit{Exist}^\Phi_\bullet(P) \;\middle|\; \Phi\in\Lambda^{\bar{X}}, \Theta\in\mathfrak{C}^{\bar{X}}_\Phi, P\in\llbracket A \rrbracket \right\}^\star$$

cette assertion est vérifiée si pour tout $\Phi \in \Lambda^{\bar{X}}$, $\Theta \in \mathfrak{C}^{\bar{X}}_\Phi$ et $R \in \llbracket A \rrbracket$, le réseau $\mathit{Coupures}(N; \mathit{Exist}^\Phi_\bullet(R), N_i)$ normalise. Or ce réseau se réduit en $\mathit{Coupures}(\Phi(P); R, N_i)$, qui normalise par réductibilité de P. □

11.4 Extension différentielle

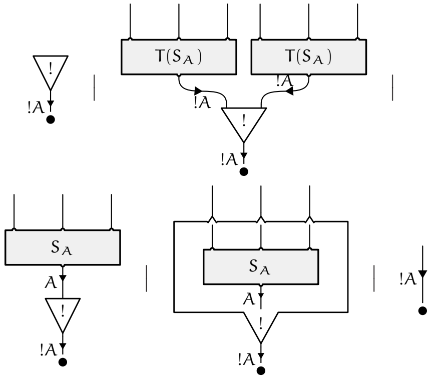
On veut étendre la preuve précédente pour englober les constructions différentielles vues dans la deuxième partie. On se contentera dans un premier temps du système avec promotions simples.

Pour alléger le formalisme on se replace dans le cas simple sans second ordre. L'intérêt majeur d'une preuve par réalisabilité est la gestion des quantifications, mais l'oublions ici pour pouvoir nous focaliser sur les véritables difficultés qui apparaissent. Conserver assez de généralité pour accepter les constructions quantifiées ne poserait pourtant pas de difficultés.

11.4.1 Candidats de réductibilité

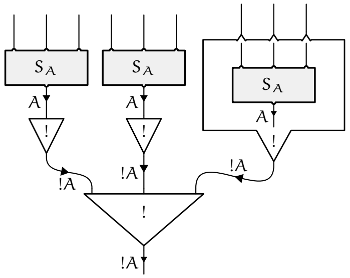
On étend la définition du candidat de réductibilité associé à un type $!A$ à l'aide de générateurs syntaxiques pour le comportement exponentiel, définis récursivement comme suit (pour un ensemble S_A donné de réseaux pointés de type A) :

$$T(S_A) \quad ::=$$



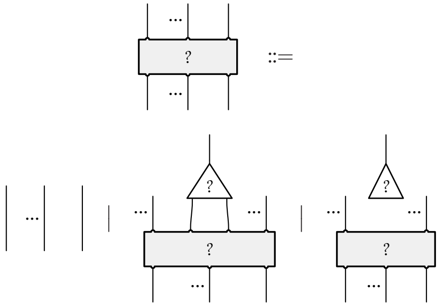
Cette syntaxe correspond à un ensemble de macro-constructions qui introduisent plusieurs opérateurs exponentiels positifs en même temps. On prend $!\mathbf{P} := \mathsf{T}(\mathbf{P})^{\star\star}$, et donc en particulier $\llbracket !A \rrbracket = \mathsf{T}(\llbracket A \rrbracket)^{\star\star}$.

Un exemple d'élément de $\mathsf{T}(S_A)$ est donné ci-dessous, où un certain nombre de *co-dérélictions* et de *promotions* se trouvent derrière un arbre de *co-contractions* d'arité quelconque :

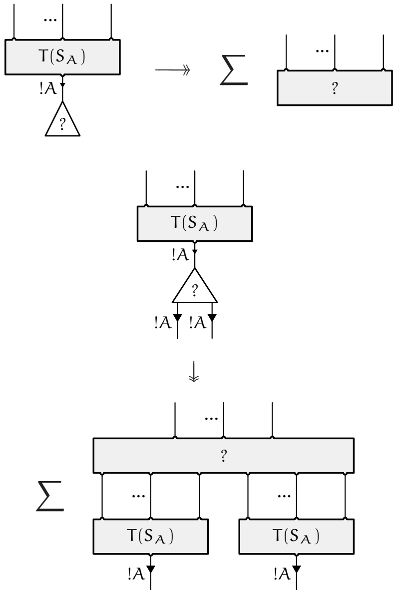


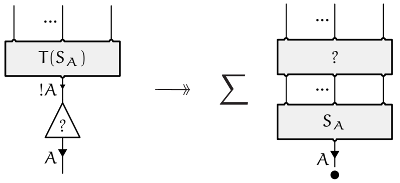
Comme ici, il nous arrivera dans la suite d'utiliser S_A ou $\mathsf{T}(S_A)$ pour désigner un réseau, on se référera à la convention expliquée en 1.3. Étant donné que nous manipulons ici des sommes de réseaux, on utilisera également la notation $\sum \mathcal{E}$ pour désigner une somme quelconque d'éléments pris dans l'ensemble $\mathcal{E}$.

On notera par une cellule rectangulaire ‹ ? › l'ensemble des réseaux constitués d'arbres de nœuds *affaiblissement* et *contraction* dirigés vers le haut :



Propriété 11-13. *On obtient les réductions remarquables suivantes :*



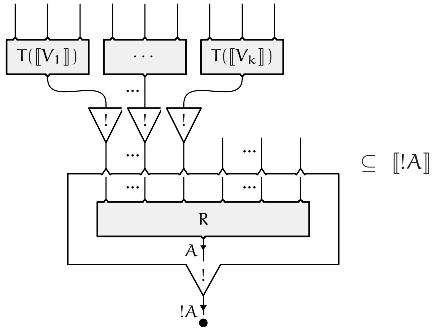


Preuve. La bonne méthode consiste à toutes les prouver en même temps par induction sur le réseau $N \in \mathsf{T}(S_A)$ considéré. □

11.4.2 Réductibilité

La réductibilité d'un réseau s'exprime toujours par les deux propriétés équivalentes déjà données dans la définition 11-7. Sans surprise, l'une des difficultés pour prouver la normalisation de notre système vient de la règle de la chaîne (voir 8.2.1). Nous utiliserons le lemme suivant.

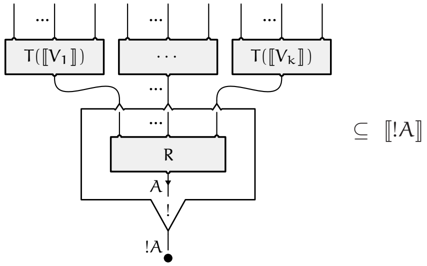
Lemme 11-14. *Pour tout réseau R réductible d'interface $!V_1, \dots, !V_k, !V'_1, \dots, !V'_{k'} \vdash A$, on a :*



Preuve. Par récurrence sur le nombre k de *co-dérélictions*. Aucune difficulté lorsque $k = 0$. Lorsque $k > 0$, on effectue une réduction entre l'une des *co-dérélictions* et la boîte, cela nous ramène à une somme de différents réseaux pour lesquels l'hypothèse de récurrence s'applique pour un nombre de *co-dérélictions* strictement inférieur à k (on utilise une hypothèse de récurrence généralisée). On parvient à montrer grâce à celle-ci que les différents réseaux de la somme sont dans $\llbracket !A \rrbracket$. □

Ce lemme peut être généralisé au cas plus général où des constructions exponentielles quelconques se présentent face aux entrées d'une boîte de *promotion*.

Lemme 11-15. *Tout réseau formé d'une boîte de promotion qui contient un réseau R réductible d'interface $!V_1, \dots, !V_k \vdash A$ coupée avec des réseaux de $\mathsf{T}(\llbracket V_i \rrbracket)$ appartient à l'interprétation $\llbracket !A \rrbracket$. Avec nos conventions on écrit cela :*



Preuve. La méthode de preuve consiste à réduire ce réseau, mais pour éviter les complications il faut utiliser une stratégie de réduction particulière. Appelons N_i les réseaux pris dans chacun des $\mathsf{T}(\llbracket V_i \rrbracket)$. On peut distinguer deux cas : ***(i)*** l'une des constructions associées aux N_i n'est pas une construction *co-dérélicion* ou un axiome, ***(ii)*** toutes les constructions associées aux N_i sont des constructions *co-dérélicion* ou des axiomes.

On remarque que le premier cas se ramène au second par commutations exponentielles. En effet, les réductions de commutation sont sont simples pour les constructions *co-affaiblissement*, *promotion* : elles entrent dans la boîte. Pour les constructions *co-contractions* c'est pareil, en une étape on fait rentrer le nœud *co-contraction* et on répète l'opération pour les deux réseaux qu'elle contient. Mais comme par hypothèse toutes ces constructions appartiennent à leurs interprétations, le réseau obtenu à l'intérieur de la boîte reste réductible.

Le second cas est traité par le lemme précédent. □

La définition un peu compliquée de $\mathsf{T}(S_A)$ a été choisie dans le but d'obtenir ce dernier lemme. Il constitue un pivot essentiel entre la définition de $\llbracket !A \rrbracket$ et le théorème clef qu'il est temps d'énoncer.

Théorème 11-16. *Tout réseau est réductible.*

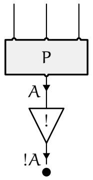
Preuve. On raisonne par induction sur le réseau N :

(Axiome, Coupure et Multiplicatifs) Comme précédemment.

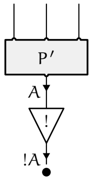
(Co-affaiblissement) Le réseau réduit à un simple *co-affaiblissement* :

se trouve immédiatement dans la syntaxe $\mathsf{T}(\llbracket A \rrbracket)$ et donc dans $\llbracket !A \rrbracket$ d'après sa définition.

(Co-dérélicion) On veut montrer que le réseau suivant est réductible :



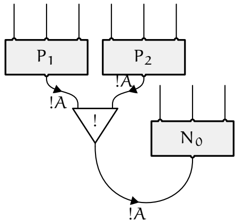
On a supposé que le réseau P est réductible par hypothèse d'induction. Lorsqu'on le coupe avec des réseaux N_i (supposés dans leurs interprétations respectives), on obtient donc un réseau $\mathsf{P}' \in [\![\mathsf{A}]\!]$. Il reste à montrer que :



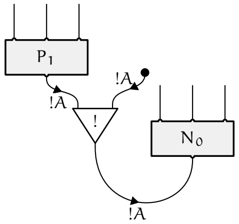
$\in [\![!\mathsf{A}]\!]$

Or ce réseau est dans $\mathsf{T}([\![\mathsf{A}]\!])$ et par conséquent il est aussi dans $[\![!\mathsf{A}]\!]$.

(Co-contraction) La *co-contraction* soulève un problème assez similaire à celui de la promotion dans la preuve précédente (théorème 11-8). Pour régler ce cas, il suffit de montrer que lorsque $\mathsf{P}_1 \in [\![!\mathsf{A}]\!]$, $\mathsf{P}_2 \in [\![!\mathsf{A}]\!]$ et $\mathsf{N}_0 \in [\![!\mathsf{A}]\!]^\star$ le réseau suivant normalise :

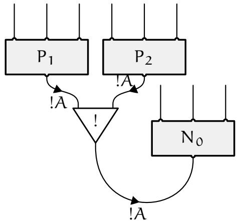


Soit, écrit d'une autre façon :

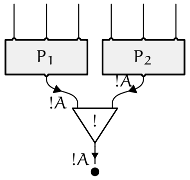


$\in [\![!\mathsf{A}]\!]^\star$

Ce qui revient à dire, par définition de $[\![!\mathsf{A}]\!]$, que le réseau suivant normalise pour tout $\mathsf{P}_2 \in \mathsf{T}([\![\mathsf{A}]\!])$:



On itère à nouveau l'astuce pour n'avoir à vérifier cela que pour $\mathsf{P}_1 \in \mathsf{T}([\![\mathsf{A}]\!])$. Il nous suffit en définitive que pour $\mathsf{P}_1 \in \mathsf{T}([\![\mathsf{A}]\!])$ et $\mathsf{P}_2 \in \mathsf{T}([\![\mathsf{A}]\!])$:

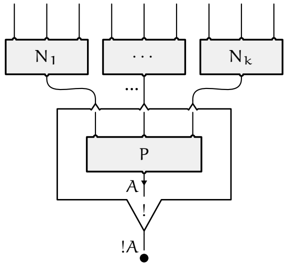


$\in [\![!\mathsf{A}]\!]$

Mais cela est une fois de plus vérifié par définition de $[\![!\mathsf{A}]\!]$.

(Promotion) Le cas de la promotion est lui aussi un peu plus difficile à régler qu'avant. La méthode employée est grosso modo la même, il faut s'assurer que la présence de réseaux face aux entrées de boîte ne pose pas de problèmes. Côté technique c'est plus difficile mais l'outil nécessaire a déjà été présenté. Il s'agit du lemme 11-15, qui a été réalisé dans ce but précis.

Lorsque P est réductible et $\mathsf{N}_1 \in [\![!\mathsf{V}_1]\!]$, ..., $\mathsf{N}_k \in [\![!\mathsf{V}_k]\!]$, on veut :



$\in [\![!\mathsf{A}]\!]$

Toujours avec la même astuce on se ramène au cas $\mathsf{N}_1 \in \mathsf{T}([\![\mathsf{V}_1]\!])$, ..., $\mathsf{N}_k \in \mathsf{T}([\![\mathsf{V}_k]\!])$, et on peut appliquer le lemme.

(Affaiblissement, Déréliction, Contraction) L'interaction de réseaux créés par ces constructions avec les générateurs du candidat de réductibilité $[\![!\mathsf{A}]\!]$ normalise dans tous les cas. Il suffit d'utiliser la propriété 11-13.

(Sommes de réseaux) Soit J un ensemble fini. On se donne pour tout $j \in \mathsf{J}$, un réseau P_j réductible, on veut montrer que $\sum_{j\in \mathsf{J}} \mathsf{P}_j$ l'est aussi. Or pour toute famille $\mathsf{N}_i \in [\![\mathsf{A}_i]\!]$, le réseau $\mathit{Coupures}(\sum_{j\in \mathsf{J}} \mathsf{P}_j ; \mathsf{N}_i) = \sum_{j\in \mathsf{J}} \mathit{Coupures}(\mathsf{P}_j ; \mathsf{N}_i)$ normalise car pour tout $j \in \mathsf{J}$, par réductibilité de P_j on sait que $\mathit{Coupures}(\mathsf{P}_j ; \mathsf{N}_i)$ normalise.

Chapitre 12

Sémantique relationnelle

Nous allons expliciter une façon d'identifier le comportement d'un réseau par une *relation*, au sens mathématique du terme. Il s'agit ici de donner une sémantique dénotationnelle aux réseaux.

12.1 Sémantique standard

12.1.1 Interprétation des types

On notera $E_1 \times E_2$ le produit cartésien de E_1 et E_2. L'élément neutre de ce produit sera noté $\mathcal{U}$, c'est un ensemble ayant pour seul élément $()$, le tuple d'arité nulle.

On notera $\mathcal{M}\, E$ l'ensemble des multi-ensembles finis d'éléments de E. Le multi-ensemble vide est noté $\varnothing$, l'union de deux multi-ensembles m_1 et m_2 est notée $m_1 \uplus m_2$, le multi-ensemble singleton contenant l'élément x est noté $\hat{x}$. Nous utiliserons aussi l'opération $\check{p}$ qui aplatit un multi-ensemble de multi-ensembles p en l'union de ses éléments.

On notera $E_1 + E_2$ la somme disjointe de E_1 et E_2, de sorte que si $x \in E_1$, on a $x^{\leftarrow} \in E_1 + E_2$ et si $y \in E_2$, on a $y^{\rightarrow} \in E_1 + E_2$.

Soit $|\cdot|$ une fonction d'interprétation qui associe un ensemble à tout type de base. On peut l'étendre à tous les types de la façon suivante :

$$\begin{aligned}
|A \parr B| = |A \otimes B| &= |A| \times |B| \\
|1| = |\bot| &= \mathcal{U} \\
|!A| = |?A| &= \mathcal{M}\,|A| \\
|A \,\&\, B| = |A \oplus B| &= |A| + |B| \\
|\top| = |0| &= \varnothing
\end{aligned}$$

12.1.2 Interprétation des preuves

Une preuve Π de $\vdash A_1, \dots, A_n$ dans la logique linéaire est traditionnellement interprétée par une relation $[\Pi] \subseteq |A_1| \times \cdots \times |A_n|$. Cette relation est définie inductivement selon la structure de la preuve, c'est l'ensemble $[\Pi]$ minimal vérifiant les conditions données ci-dessous.

Axiome

$$\overline{\vdash A, A^{\perp}}$$

$$\forall a \in |A|,\ (a, a) \in [\Pi]$$

Coupure

$$\dfrac{\overline{\overline{\vdash A, \Gamma_1}}\ \Pi_1 \qquad \overline{\overline{\vdash A^{\perp}, \Gamma_2}}\ \Pi_2}{\vdash \Gamma_1, \Gamma_2}$$

$$(a, z_1, \dots) \in [\Pi_1] \ \wedge\ (a, z_2, \dots) \in [\Pi_2] \ \Rightarrow\ (z_1, \dots, z_2, \dots) \in [\Pi]$$

Tenseur

$$\dfrac{\overline{\overline{\vdash A, \Gamma_1}}\ \Pi_1 \qquad \overline{\overline{\vdash A, \Gamma_2}}\ \Pi_2}{\vdash A \otimes B, \Gamma_1, \Gamma_2}$$

$$(a, z_1, \dots) \in [\Pi_1] \ \wedge\ (b, z_2, \dots) \in [\Pi_2] \ \Rightarrow\ ((a, b), z_1, \dots, z_2, \dots) \in [\Pi]$$

Par

$$\dfrac{\overline{\overline{\vdash A, B, \Gamma}}\ \Pi'}{\vdash A \parr B, \Gamma}$$

$$(a, b, z, \dots) \in [\Pi'] \ \Rightarrow\ ((a, b), z, \dots) \in [\Pi]$$

Affaiblissement

$$\dfrac{\overline{\overline{\vdash \Gamma}}\ \Pi'}{\vdash ?A, \Gamma}$$

$$(z, \dots) \in [\Pi'] \ \Rightarrow\ (\varnothing, m, \dots) \in [\Pi]$$

Déréliction

$$\dfrac{\overline{\overline{\vdash ?A, \Gamma}}\ \Pi'}{\vdash A, \Gamma}$$

$$(a, z, \dots) \in [\Pi'] \ \Rightarrow\ (\hat{a}, z, \dots) \in [\Pi]$$

Contraction

$$\dfrac{\overline{\overline{\vdash ?A, ?A, \Gamma}}\ \Pi'}{\vdash ?A, \Gamma}$$

$$(m_1, m_2, z, \dots) \in [\Pi'] \ \Rightarrow\ (m_1 \uplus m_2, z, \dots) \in [\Pi]$$

Promotion

$$\dfrac{\overline{\overline{\vdash A, ?\Gamma}}\ \Pi'}{\vdash !A, ?\Gamma}$$

$$\forall i \in I,\ (a_i, p_i, \dots) \in [\Pi'] \ \Rightarrow\ \left(\biguplus_{i \in I} \hat{a}_i, \biguplus_{i \in I} p_i, \dots\right) \in [\Pi]$$

$\overleftarrow{Pl}$us

$$\dfrac{\overline{\overline{\vdash A, \Gamma}}\ \Pi'}{\vdash A \oplus X, \Gamma}$$

$$(a, z, \dots) \in [\Pi'] \ \Rightarrow\ (a^{\leftarrow}, z, \dots) \in [\Pi]$$

$\overrightarrow{Pl}$us

$$\dfrac{\overline{\overline{\vdash A, \Gamma}}\ \Pi'}{\vdash X \oplus B, \Gamma}$$

$$(b, z, \dots) \in [\Pi'] \ \Rightarrow\ (b^{\rightarrow}, z, \dots) \in [\Pi]$$

Avec

$$\dfrac{\overline{\overline{\vdash A, \Gamma}}\ \Pi_1 \qquad \overline{\overline{\vdash B, \Gamma}}\ \Pi_2}{\vdash A \,\&\, B, \Gamma}$$

$$\begin{aligned}
(a, z, \dots) \in [\Pi_1] &\ \Rightarrow\ (a^{\leftarrow}, z, \dots) \in [\Pi] \\
(b, z, \dots) \in [\Pi_2] &\ \Rightarrow\ (b^{\rightarrow}, z, \dots) \in [\Pi]
\end{aligned}$$

Aucune condition n'est imposée pour une construction *top*, sa sémantique se réduira donc nécessairement à une sémantique vide, et celle-ci se propage à l'intérieur de la boîte la qui contient.

Notons qu'on peut étendre la sémantique qui a été définie à la règle de *récursion* introduite dans le chapitre 5.

Récursion

$$\dfrac{\overline{\overline{\vdash A, ?A^{\perp}, ?\Gamma}}\ \Pi'}{\vdash !A, ?\Gamma}$$

$$\forall i \in I,\ (a_i, r_i, z_i, \dots) \ \wedge\ (r_i, z'_i, \dots) \in [\Pi] \ \Rightarrow\ \left(\biguplus_{i \in I} \hat{a}_i, \biguplus_{i \in I} z_i \uplus z'_i, \dots\right) \in [\Pi]$$

12.1.3 Interprétation des réseaux

On peut définir une interprétation des réseaux sans recourir à une séquentialisation préalable. Si N est un réseau d'interface $\vdash A_1, \dots, A_n$, celui-ci va être interprété par une relation $[N] \subseteq |A_1| \times \cdots \times |A_n|$.

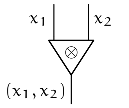
Définition 12-1. *Une* expérience *d'un réseau* N *est un étiquetage de chaque fil de ce réseau par un élément de l'interprétation du type qui lui est associé. Une expérience est* réussie *si elle satisfait au voisinage de chaque nœud une relation spécifique à l'opérateur qu'il incarne.*

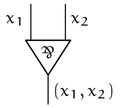
Les relations spécifiques à chaque opérateur dont il est question seront explicités juste après, à l'aide de motifs.

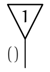
Définition 12-2. *Un tuple* $(x_1, \dots, x_n)$ *est un point de la sémantique relationnelle* $[N]$ *d'un réseau* N *s'il étiquette l'interface d'une expérience réussie de ce réseau.*

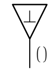
Multiplicatifs

Les relations spécifiques associées aux constructions multiplicatives sont des relations simples autour de la structure de produit cartésien considérée. Les combinaisons suivantes définissent les points de ces relations :





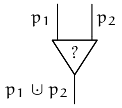


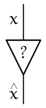


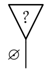
Ces diagrammes sont appelés motifs, ils contiennent des variables qui peuvent être instanciées par n'importe quel élément de l'interprétation correspondante.

Exponentiels

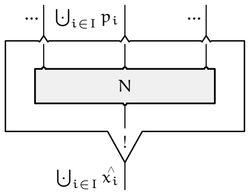
Les relations associées aux opérateurs exponentiels sont des relations basiques construites autour de la notion de multi-ensemble.



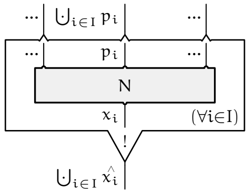




La relation qui doit être satisfaite au voisinage d'une boîte de *promotion* contenant un réseau N est constituée des points décrits ci-dessous, qu'on peut obtenir à partir d'une famille quelconque de points $(x_i, p_i, \dots)_{i\in I}$ de la sémantique de N :



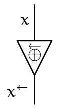
Une expérience est donc réussie pour une boîte si elle correspond à une famille d'expériences réussies du réseau qu'elle contient. Aussi, pour exprimer que les points $(x_i, p_i, \dots)$ sont supposés dans la sémantique de N pour tout $i \in I$, il nous arrivera de placer la quantification sur le dessin, à l'intérieur de la boîte. On obtient un motif un peu particulier, dans lequel plusieurs familles de variables peuvent être instanciées, qui résume notre définition :

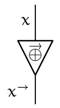


Au niveau de la sortie de boîte, on trouve un multi-ensemble contenant les éléments x_i, et au niveau des entrées on combine par union tous les multi-ensembles de ressources p_i liés aux x_i par le réseau N.

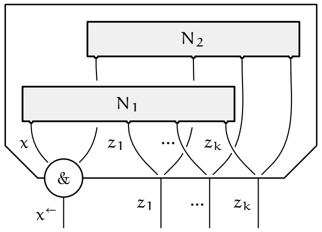
Additifs

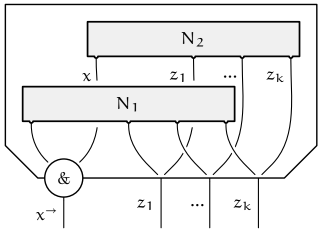
Les relations associées aux opérateurs additifs sont des relations basiques construites autour de la notion de somme disjointe.





Les points de la sémantique d'une boîte *avec* combinant deux tranches N_1 et N_2 sont obtenus à partir des points des sémantiques de N_1 et N_2 : à chaque expérience réussie de N_1 correspond une expérience réussie de la boîte *avec*, et il en est de même pour chaque expérience réussie de N_2. Voici les deux motifs qui décrivent ces correspondances :

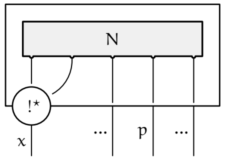




Étant donné que $|\top| = \varnothing$, il n'y a bien sûr aucun point dans la sémantique d'une boîte *top*.

Récursion

Les points $(x, p, \dots)$ de la sémantique d'une boîte de *récursion* :



sont ceux de l'ensemble E, défini comme le plus petit ensemble tel que pour toutes familles (x_i), (r_i), (p_i)… et (p'_i)…, indexées par $i \in I$, on ait :

$$\forall i \in I,\ (x_i, r_i, p_i, \dots) \in [N] \ \wedge\ (r_i, p'_i, \dots) \in E$$
$$\Rightarrow \left(\biguplus_{i\in I} \hat{x_i}, \biguplus_{i\in I} p_i \uplus p'_i, \dots \right) \in E$$

12.1.4 Propriétés

La relation qui définit la sémantique d'une coupure entre deux réseaux est, selon la définition donnée, obtenue par composition des relations associées à ces deux réseaux. La propriété donnée ici justifie le caractère dénotationnel que possède cette sémantique.

Propriété 12-3. *La sémantique d'un réseau est conservée par réduction :*

$$N \longrightarrow N' \ \Rightarrow\ [N'] = [N]$$

Preuve. On vérifie que chaque règle de réduction préserve la sémantique. □

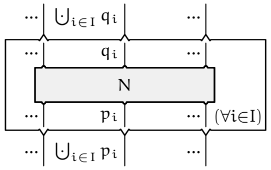
Par ailleurs, il existe une sémantique relationnelle pour le langage *PCF* que nous avons présenté dans la première partie. Nous pensons que la traduction vers les réseaux qui en a été donnée, et qui utilise les boîtes récursives, respecte cette sémantique.

12.2 Sémantique des systèmes différentiels

À titre indicatif nous allons décrire la sémantique relationnelle qu'il faut donner aux nouvelles constructions différentielles, introduites dans la deuxième partie, pour retrouver la propriété recherchée : la conservation de la sémantique par la réduction.

12.2.1 Super-promotion

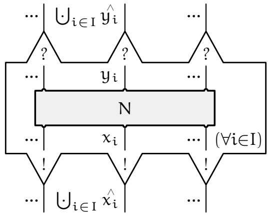
Un point de la sémantique d'une boîte de *super-promotion* contenant un réseau N est obtenu pour toute famille de points de la sémantique de N de la façon suivante :



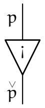
Les deux différences avec la *promotion* sont qu'il y a ici plusieurs sorties et que la sémantique des sorties est maintenant similaire à celle qu'avaient les entrées.

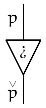
12.2.2 Réplication

Lorsqu'on regarde ce qui se passe pour la *réplication* fonctorielle, on s'aperçoit qu'il s'agit de l'autre choix, celui où on utilise pour toutes les portes une sémantique similaire à celle qu'avaient les sorties de promotion. Un point de la sémantique d'une boîte de *réplication* contenant un réseau N est obtenu pour toute famille de points de la sémantique de N de la façon suivante :



Et les opérateurs d'*enfouissement* et de *co-enfouissement* ayant pour sémantique :





on retrouve bien sûr la sémantique de la *super-promotion* lorsqu'on place ces opérateurs autour de la construction *réplication* :

$$\overset{\vee}{\biguplus_{i\in I}} \hat{p_i} = \biguplus_{i\in I} p_i$$

12.3 Sémantique exponentielle localisée

On se propose ici d'obtenir une sémantique relationnelle uniquement à partir de relations locales. Les définitions de la sémantique standard utilisent des relations qui ne sont pas locales pour définir la sémantique des boîtes.

Nous utiliserons le formalisme des systèmes localisés étudiés dans les chapitres 3 et 4 pour exprimer cette sémantique. Nous nous appuierons d'abord sur les systèmes de réseaux localisés actifs, mais nous verrons que la sémantique localisée obtenue n'est pas tout à fait dénotationnelle. C'est en utilisant les autres systèmes localisés (passifs et contrôlés), et plus spécifiquement grâce aux canaux de contrôle, qu'on parviendra à exprimer une vraie sémantique dénotationnelle localisée.

12.3.1 Arborescences exponentielles

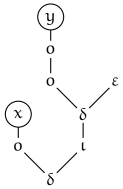
Une arborescence exponentielle est une structure pratique pour contenir plusieurs éléments d'un ensemble Δ de façon structurée.

Définition 12-4. *Soit Δ un ensemble, on appelle arborescences exponentielles les éléments de l'ensemble Δ^e défini par la syntaxe suivante :*

$$\Delta^e \quad ::= \quad \varepsilon \;\big|\; \delta\, \Delta^e\, \Delta^e \;\big|\; o\, \Delta \;\big|\; \iota\, \Delta^{ee}$$

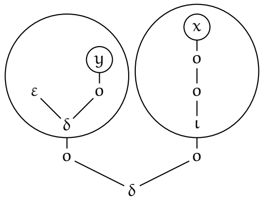
Par exemple, si $x, y \in \Delta$ alors l'expression suivante désigne une arborescence exponentielle de Δ^e :

$$\delta\,(o\,x)\,(\iota\,(\delta\,(o\,(o\,y))\,\varepsilon))$$



On peut aussi construire des arborescences d'arborescences (c'est-à-dire des éléments de Δ^{ee}), par exemple :

$$\delta\,(o\,(\delta\,\varepsilon\,(o\,y)))\,(o\,(\iota\,(o\,(o\,x))))$$



Notamment, le constructeur ι transforme ces arborescences d'arborescences en de simples arborescences.

On fera attention au fait que les éléments $\varepsilon \in \Delta^e$ et $\varepsilon \in \Delta^{ee}$ sont considérés comme des éléments différents. Cette ambiguïté en génère d'autres comme $\delta\,\varepsilon\,\varepsilon \in \Delta^e$ et $\delta\,\varepsilon\,\varepsilon \in \Delta^{ee}$. On précisera donc l'ensemble auquel appartient toute expression qu'on emploiera.

Définition 12-5. *Les arborescences de profondeur i d'un ensemble Δ sont les éléments de $\Delta^{(i)}$, qui est défini par :*

$$\Delta^{(0)} \;=\; \Delta \qquad\qquad \Delta^{(i+1)} \;=\; \Delta^{(i)e}$$

On note Δ^* l'ensemble des arborescences de Δ de profondeurs arbitraires :

$$\Delta^* \;=\; \bigcup_{i \in \mathbb{N}} \Delta^{(i)}$$

Clôture contextuelle

Les arborescences seront utilisées pour réaliser plusieurs expériences simultanément. En particulier, si deux expériences particulières sont réussies, on doit faire en sorte qu'une expérience sur des arborescences qui les combinent soit également réussie. On introduit pour cela une notion de clôture.

Définition 12-6. *Étant donné une relation $R \subseteq \Delta_1^* \times \cdots \times \Delta_n^*$, on définit sa* clôture contextuelle *$\overline{R} \subseteq \Delta_1^* \times \cdots \times \Delta_n^*$ comme la plus petite relation contenant R telle que pour tout entiers $i_1, \ldots, i_n$:*

$$(\varepsilon \in \Delta_1^{(i_1)e}, \ldots, \varepsilon \in \Delta_n^{(i_n)e}) \in \overline{R}$$

$$\begin{array}{c}(t_1 \in \Delta_1^{(i_1)e}, \ldots, t_n \in \Delta_n^{(i_n)e}) \in \overline{R}\\ (t'_1 \in \Delta_1^{(i_1)e}, \ldots, t'_n \in \Delta_n^{(i_n)e}) \in \overline{R}\end{array} \;\Rightarrow$$

$$(\delta\, t_1\, t'_1 \in \Delta_1^{(i_1)e}, \ldots, \delta\, t_n\, t'_n \in \Delta_n^{(i_n)e}) \in \overline{R}$$

$$(t_1 \in \Delta_1^{(i_1)}, \ldots, t_n \in \Delta_n^{(i_n)}) \in \overline{R} \;\;\Rightarrow$$

$$(o\, t_1 \in \Delta_1^{(i_1)e}, \ldots, o\, t_n \in \Delta_1^{(i_n)e}) \in \overline{R}$$

$$(t_1 \in \Delta_1^{(i_1)ee}, \ldots, t_n \in \Delta_n^{(i_n)ee}) \in \overline{R}$$

$$\Rightarrow$$

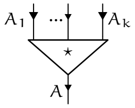
$$(\iota\, t_1 \in \Delta_1^{(i_1)e}, \ldots, \iota\, t_n \in \Delta_n^{(i_n)e}) \in \overline{R}$$

12.3.2 Interprétation des réseaux localisés

Nous nous plaçons dans le cadre des systèmes localisés actifs, celui correspondant aux constructions exponentielles a été décrit en 3.2. Nous évoquerons également la construction récursive abordée en 5.2.

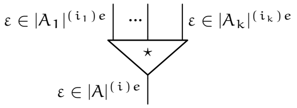
Soit N un réseau d'interface $\vdash A_1, \ldots, A_n$. Pour définir la sémantique localisée de N, un fil de type A ne sera plus étiqueté par un élément de $|A|$, mais par une arborescence de $|A|^*$. Le réseau N sera donc interprété par une relation $[N]_\iota \subseteq |A_1|^* \times \cdots \times |A_n|^*$. Nous nous arrangerons pour que celle-ci soit contextuellement close. Malgré ce léger changement, on reprend les définitions 12-1 et 12-2 (données en 12.1.3) pour définir *expériences*, expériences *réussies* et *sémantique relationnelle* localisée.

Il ne reste plus qu'à préciser quelles sont les relations qu'on considère pour les opérateurs des systèmes localisés. Celles-ci seront contextuellement closes et seront une fois de plus décrites à l'aide de motifs. On peut remarquer que cette clôture s'exprime également à l'aide de motifs. Par exemple, pour un nœud $\star$ typé comme suit :

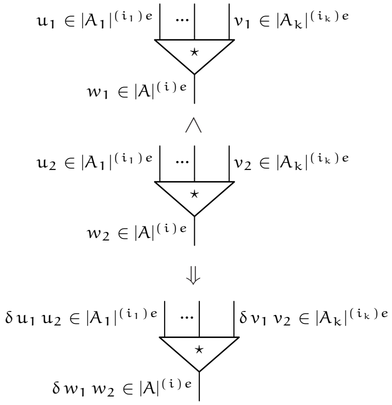


on peut utiliser les motifs et déductions de motifs suivants pour compléter une relation déjà connue à propos de l'opérateur $\star$:

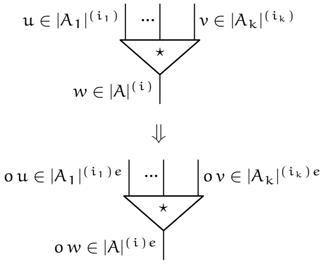
- ε-clôture :



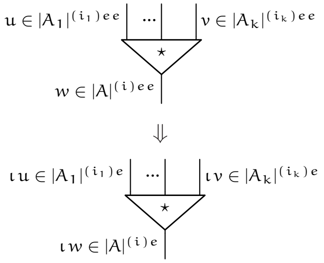
- δ-clôture :



- o-clôture :



- ι-clôture :



Propriété 12-7. *Une sémantique relationnelle engendrée par des relations locales contextuellement closes est elle même contextuellement close.*

Preuve. On prouve sans difficultés que la composition de deux relations closes est close. □

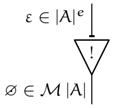
Multiplicatifs

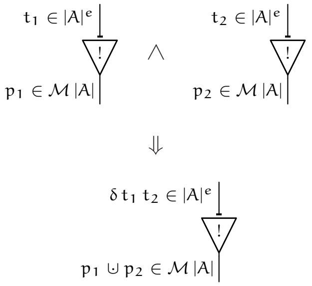
Les relations associées aux opérateurs mutiplicatifs sont simplement les clôtures contextuelles des relations qui ont été définies à l'aide de motifs dans le cas de la sémantique standard, en 12.1.3.

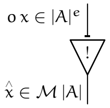
Exponentiels

Les relations associées aux opérateurs exponentiels (actifs) et aux opérateurs de boîtes sont les relations définies par les motifs et déductions de motifs donnés ci-dessous. On y rajoutera les règles de clôture que l'on vient de voir pour obtenir une relation contextuellement close.

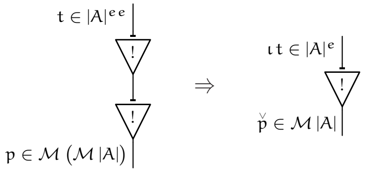
Les sorties de *promotion* structurent les multi-ensembles qui étiquettent leur port principal en des arborescences (en toutes les arborescences imaginables qui contiennent les éléments du multi-ensemble).



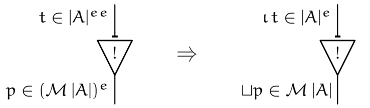




Le cas des contextes d'ingestion pourrait lui aussi être traité directement par une déduction de motif :



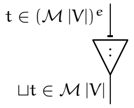
Cependant, il est judicieux de définir une opération $\sqcup : (\mathcal{M}\,\Delta)^e \longrightarrow \mathcal{M}\,\Delta$ sur les arborescences de multi-ensembles qui écrase par unions cette arborescence en un simple multi-ensemble. La déduction précédente peut alors être remplacée par celle-ci :



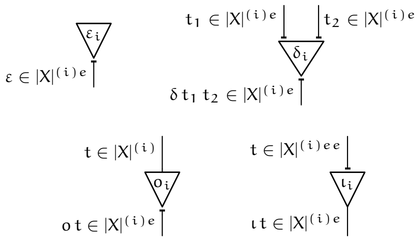
et formellement, cette opération est définie récursivement par :

$$\sqcup\varepsilon \;:=\; \varnothing \qquad \sqcup(\delta\, t_1\, t_2) \;:=\; \sqcup t_1 \uplus \sqcup t_2$$
$$\sqcup(o\, t) \;:=\; t \qquad \sqcup(\iota\, t) \;:=\; \sqcup\sqcup t$$

D'ailleurs, la sémantique des entrées de *promotion* correspond à cette pure opération :

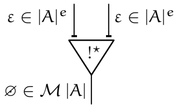


Pour les opérateurs de boîtes, on utilise les motifs suivants (ils dépendent de l'altitude de l'opérateur) :

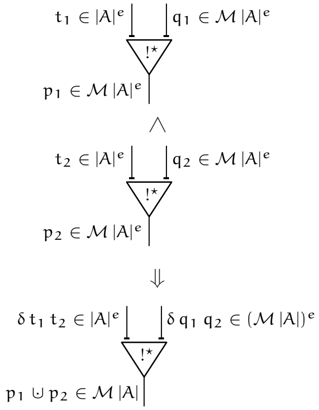


Récursion

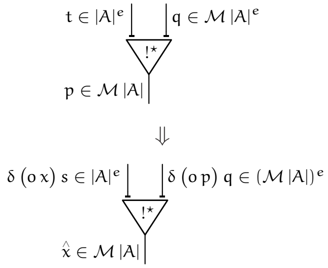
Une parenthèse pour signaler que la sémantique localisée de l'opérateur de récursion peut aussi être définie, par le motif :



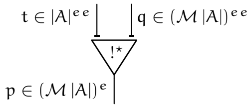
et par les déductions de motifs suivantes :

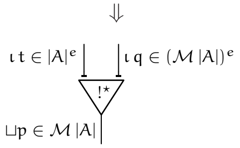


la récursion proprement dite étant faite grâce à :



et enfin, pour gérer les contextes ι :





Comprendre la sémantique de l'opérateur de récursion demande un plus grand effort que celle des autres opérateurs. Elle est relativement complexe, cependant elle reste locale.

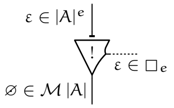
Nous venons d'achever la définition de la sémantique des réseaux du système exponentiel actif. Elle possède des propriétés communes avec la sémantique des autres systèmes localisés dont il va être question maintenant. Nous n'énoncerons ces propriétés qu'après, en 12.3.4.

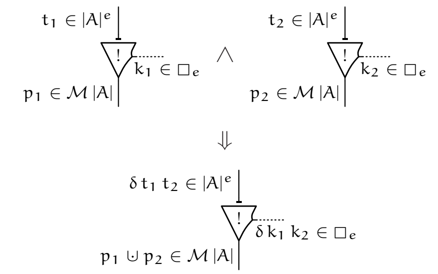
12.3.3 Interprétation des réseaux localisés avec contrôle

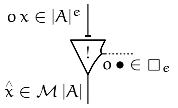
Les codages localisés passifs et contrôlés utilisent eux des canaux de contrôle. En tenir compte va permettre de raffiner la sémantique relationnelle, en lui apportant des propriétés intéressantes. Une sémantique pour le système passif se déduit d'une sémantique pour le système contrôlé : le choix de considérer certains ports principaux ou auxiliaires n'a aucune influence sur la définition d'une sémantique relationnelle. Nous écrirons donc cette sémantique commune en utilisant la syntaxe du système contrôlé.

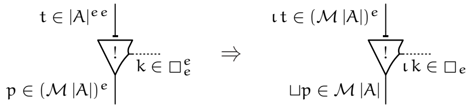
Exponentiels

On étend l'interprétation des types au type associé aux canaux de contrôle exponentiels : ce type sera interprété dans l'ensemble $\square_e = \{\bullet\}^e$, où $\bullet$ désigne un élément fixé donné. Les motifs peuvent être complétés assez simplement, et on rappelle que c'est leur clôture contextuelle qui sert de relation locale pour l'opérateur considéré :

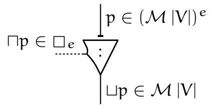








Pour les entrées ça marche de la même manière qu'avant :



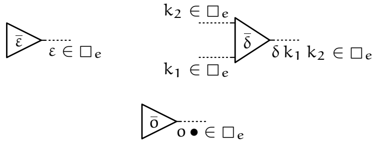
Pour utiliser ce raccourci, on a cette fois besoin de l'opération $\sqcap\colon \Delta^e \longrightarrow \{\bullet\}^e$ qui ne garde que la structure arborescente de surface de son argument et remplace les éléments de Δ qu'elle contient par l'élément $\bullet$. C'est l'opération *map* $(x \mapsto \bullet)$, où *map* : $(\Delta_1 \longrightarrow \Delta_2) \longrightarrow (\Delta_1^e \longrightarrow \Delta_2^e)$ est définie par :

$$\begin{aligned} \mathit{map}\ \varphi\ \varepsilon &:= \varepsilon \\ \mathit{map}\ \varphi\ (\delta\, t_1\, t_2) &:= \delta\, (\mathit{map}\ \varphi\ t_1)\, (\mathit{map}\ \varphi\ t_2) \\ \mathit{map}\ \varphi\ (o\, t) &:= o\, (\varphi\, t) \\ \mathit{map}\ \varphi\ (\iota\, t) &:= \iota\, (\mathit{map}\ (\mathit{map}\ \varphi)\ t) \end{aligned}$$

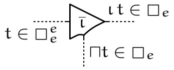
Si on considère une portion P de réseau qui n'a pas de canal de contrôle dans son interface, et P' sa correspondante dans le formalisme actif. La sémantique $[P]_c$ obtenue en considérant les canaux de contrôle est plus fine que (incluse dans) $[P']_a$, la sémantique que l'on obtiendrait sans eux. Une contrainte d'uniformité des arborescences est imposée au moyen des canaux de contrôle à tous les nœuds (sortie et entrées) matérialisant l'interface d'une même boîte. Ces contraintes se propagent dans la boîte par connexité. En l'absence de canal de contrôle, une expérience peut être réussie même si les arborescences utilisées sur différents fils d'une même boîte ne sont pas identiques.

Messages

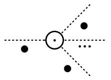
Les interprétations des messages obéissent aux motifs suivants :



Pour les messages d'ingestion on utilisera :

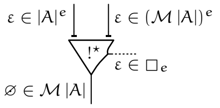


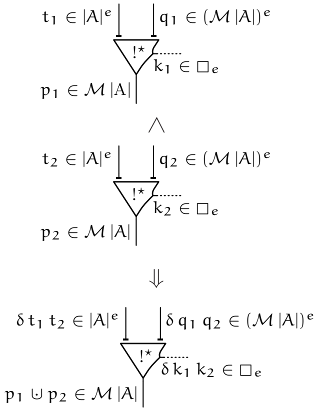
Et pour les opérateurs de diffusion c'est simple, l'élément $\bullet$ étiquette tous les fils :



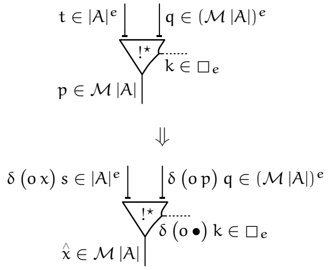
Récursion

Pour l'opérateur de récursion, on reprend les motifs du cas actif et on les complète par l'information adéquate qu'il faut placer sur le canal de contrôle.

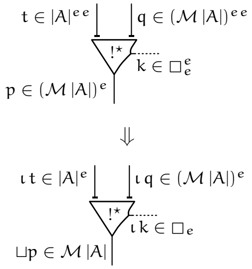




puis :



et :



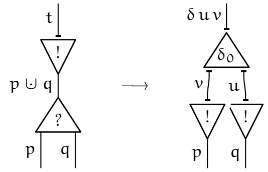
12.3.4 Propriétés

On notera $[N]_a$ la sémantique localisée d'un réseau N du formalisme actif, et $[N]_c$ la sémantique localisée d'un réseau N des formalismes passif ou contrôlé. Tous les motifs nécessaires à leurs définitions ont été donnés. Lorsqu'un résultat est vrai dans les trois systèmes localisés, on utilisera la notion $[N]_l$ pour désigner la sémantique correspondante.

Propriété 12-8. *La sémantique localisée d'un morceau de réseau (pas nécessairement valide) est affinée par réduction :*

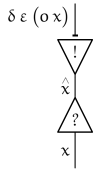
$$P \longrightarrow P' \quad \Rightarrow \quad [P']_l \subseteq [P]_l$$

Preuve. On vérifie que chaque règle de réduction affine la sémantique localisée. Prenons un exemple où la sémantique décroît vraiment, cela arrive lorsque les liens du réseau n'ont pas la même profondeur :

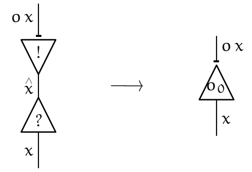


Les seules contraintes qu'il manque dans les annotations fournies ici sont celles données par la sémantique des sorties de promotion. Or à partir des deux motifs de droite on peut déduire le motif de gauche pour $t = \delta\, u\, v$ par utilisation de la déduction de motif correspondante. Les autres règles sont traitées de façon similaire. □

En particulier, on notera que l'inclusion peut être stricte pour certains morceaux de réseaux. Après réduction, le point de la sémantique correspondant à l'expérience donnée ici n'est plus représenté :



Sur le lien interne à la boîte (ici celui du haut), après interaction avec une *déréliction*, seule la représentation arborescente $o\, x$ sera conservée dans l'interprétation du réseau réduit :



En fait, les seules règles qui ne conservent pas exactement la sémantique sont celles qui interviennent aux limites des boîtes. Mais dans le cas d'un réseau valide, les structures arborescentes utilisées pour l'interprétation de l'intérieur des boîtes (comme $\delta\,\varepsilon\,(o\,x)$ ou $o\,x$ dans notre exemple) ne sont pas accessibles depuis son interface. Seul leur contenu commun est accessible, et celui-ci est préservé par réduction. Dans notre exemple :

$$\sqcup(\delta\,\varepsilon\,(o\,x)) \;=\; \sqcup(o\,x) \;=\; \hat{x}$$

Les résultat donnés ci-dessous n'ont pas été vérifiés pour la construction récursive, nous pensons néanmoins qu'ils restent valides en sa présence.

Théorème 12-9. *La sémantique localisée d'un réseau valide (du système actif, passif ou contrôlé) contient la clôture contextuelle de la sémantique standard de la preuve qui lui est associée.*

$$\Pi \mapsto N \;\;\Rightarrow\;\; \overline{[\Pi]} \subseteq [N]_l$$

Preuve. On raisonne par induction sur la preuve associée au réseau. Les difficultés se trouvent dans les cas qui feraient intervenir des boîtes si nous nous trouvions dans un formalisme non localisé. Dans la sémantique standard, une boîte exponentielle contenant un réseau N possède une expérience $(x, z, \ldots)$ réussie s'il existe une famille d'expériences réussies $(x_i, z_i, \ldots) \in [N]$. Dans la sémantique localisée cette expérience est validée en utilisant un arbre t quelconque contenant les x_i pour satisfaire le motif en sortie de boite, et des arbres contenant les $z_i, \ldots$ ayant la même structure pour satisfaire les motifs en sortie de boîte. On utilise alors la sémantique du réseau N construite inductivement en exploitant la propriété de clôture contextuelle pour combiner tous les expériences selon la structure de l'arbre choisie. □

L'inclusion dans le sens opposé, n'est vraie que dans le cas de réseaux contrôlés.

Théorème 12-10. *La sémantique localisée d'un réseau valide du système contrôlé est égale à la clôture contextuelle de la sémantique standard de la preuve qui lui est associée.*

$$\Pi \mapsto_c N \;\;\Rightarrow\;\; \overline{[\Pi]} = [N]_c$$

Preuve. (idée) Étant donné que le canal de contrôle impose une structure d'arbre identique au niveau de toutes les sorties d'une construction exponentielle, les expériences qui sont validées dans la sémantique localisée sont de la forme de celles qui sont validées dans la sémantique standard. □

Corollaire 12-11. *La sémantique localisée d'un réseau valide (du système passif ou contrôlé) est préservée par réduction :*

$$N \longrightarrow N' \;\;\Rightarrow\;\; [N']_c = [N]_c$$

Preuve. On sait déjà que $[N']_l \subseteq [N]_l$. Nous nous plaçons à présent dans le système contrôlé. Lorsqu'un réseau N provient d'une preuve Π, il se réduit en un réseau N_0 obtenu par traduction de $\Pi{\downarrow}$. Ce dernier vérifie $N \twoheadrightarrow N_0$ par simulation (voir théorème 3-13) et il possède la même sémantique que N, car $[N]_c = \overline{[\Pi]} = \overline{[\Pi{\downarrow}]} = [N_0]_c$. Étant donné que N' est en forme normale, par confluence modulo, pour tout réseau N' tel que $N \twoheadrightarrow N'$ on a : $N' \,(\longrightarrow \cup \longmapsto)^*\, N_0$. Mais on montre que facilement le modulo conserve la sémantique et par conséquent $[N']_c \subseteq [N_0]_c = [N]_c$ (c'est ce qu'il fallait démontrer). □

Ce dernier résultat est peut-être valide dans le système actif, mais la preuve que nous venons de donner pour les systèmes avec contrôle ne s'adapte pas facilement.

12.4 Sémantique additive localisée

On peut poursuivre la localisation de la sémantique relationnelle en l'étendant aux constructions additives. La sémantique des constructions additives peut être donnée indépendamment de celle des constructions exponentielles. Il faut tout de même préciser comment on fait fonctionner les deux en même temps. On utilise pour cela des arborescences mixtes.

12.4.1 Arborescences mixtes

La notion d'arborescence additive est plutôt dégénérée, mais nous définissons ces arborescences de la

même façon que nous avons défini les arborescences exponentielles. Cela nous permettra d'exprimer leurs combinaisons plus facilement.

Définition 12-12. *Soit Δ un ensemble. L'ensemble Δ^a des arborescences additives de Δ est défini ainsi :*

$$\Delta^a \;\;::=\;\; \lambda\,\Delta \;\mid\; \rho\,\Delta \;\mid\; \kappa$$

Définition 12-13. *Les arborescences mixtes de profondeur i d'éléments de Δ sont les éléments de $\Delta^{(i)}$, qui est défini par :*

$$\Delta^{(0)} \;=\; \Delta \qquad\qquad \Delta^{(i+1)} \;=\; \Delta^{(i)e} \cup \Delta^{(i)a}$$

On notera Δ^* l'ensemble des arborescences mixtes de Δ de profondeurs arbitraires :

$$\Delta^* \;=\; \bigcup_{i \in \mathbb{N}} \Delta^{(i)}$$

Ces arborescences génériques contiennent des strates exponentielles et des strates additives. Elles vont permettre de raisonner dans un contexte où boîtes exponentielles et boîtes additives peuvent être imbriquées.

Clôture contextuelle

On définit directement une opération qui combine clôture pour les strates exponentielles et clôture pour les strates additives.

Définition 12-14. *La clôture contextuelle d'une relation $R \subseteq \Delta_1^* \times \cdots \times \Delta_n^*$, est la plus petite relation $\overline{R}$ contenant R, close par les opérations déjà présentées (définition 12-6) auxquelles on ajoute :*

$$(x_1 \in \Delta_1^{(i_1)}, \ldots, x_n \in \Delta_n^{(i_n)}) \in \overline{R} \;\;\Rightarrow$$
$$(\lambda\,x_1 \in \Delta_1^{(i_1)a}, \ldots, \lambda\,x_n \in \Delta_n^{(i_n)a}) \in \overline{R}$$
$$(x_1 \in \Delta_1^{(i_1)}, \ldots, x_n \in \Delta_n^{(i_n)}) \in \overline{R} \;\;\Rightarrow$$
$$(\rho\,x_1 \in \Delta_1^{(i_1)a}, \ldots, \rho\,x_n \in \Delta_n^{(i_n)a}) \in \overline{R}$$

$$(\kappa \in \Delta_1^{(i_1)a}, \ldots, \kappa \in \Delta_n^{(i_n)a}) \in \overline{R}$$

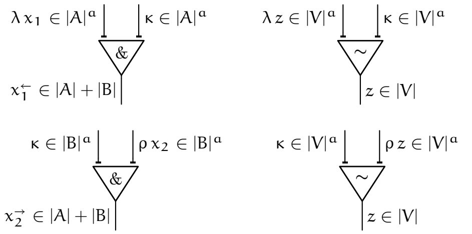
pour tout entiers i_1, …, i_n.

12.4.2 Interprétation des réseaux localisés

On va décrire des relations pour les opérateurs additifs localisés qu'il faudra clore (comme tous les autres opérateurs) selon la nouvelle notion de clôture.

Additifs

Les motifs pour les opérateurs additifs localisés sont :



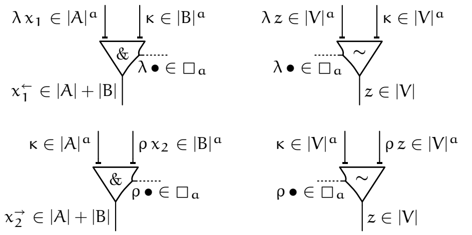
Pour les opérateurs de la construction *top*, aucun motif n'existe. Cela dit, un réseau contenant une telle construction peut avoir une sémantique non vide si cette construction est placée dans la partie « morte » d'une construction additive *avec*, ou même dans une boîte exponentielle non utilisée.

12.4.3 Interprétation des réseaux localisés avec contrôle

L'interprétation du type associé aux canaux de contrôle additifs se fait dans l'ensemble $\square_a = \{\bullet\}^a$.

Additifs

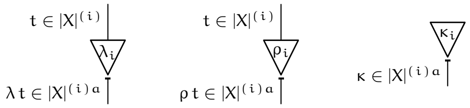
Les motifs sont ainsi complétés :



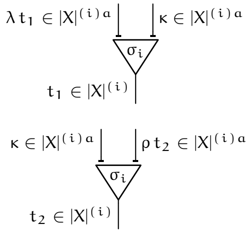
Pour les opérateurs de la construction *top*, aucun motif n'existe.

Opérateurs de boîtes

Sur le même principe que pour leurs homologues exponentiels, on utilise :

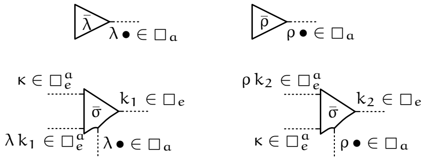


Les opérateurs d'*oubli* n'ont pas de motifs. En ce qui concerne les opérateurs de *copie*, on permet les motifs suivants :



Messages

Enfin, les interprétations des messages obéissent aux motifs suivants :



On a utilisé $\square_e$ pour étiqueter les canaux parcourus car les messages $\bar\sigma$ ne parcourent que des boîtes exponentielles dans le système présenté, mais il n'y a pas vraiment de raison à cela. D'autres types de boîtes pourraient être parcourues, et il est possible de remplacer $\square_e$ par n'importe quel ensemble de contrôle $\square_x$ dans ces motifs.

12.4.4 Propriétés

On espère les mêmes théorèmes que pour la sémantique localisée exponentielle.

Conclusion

Cette thèse explore différentes idées mais le potentiel de chacune n'a pu être exploité pleinement. De nombreuses questions restent pour l'instant sans réponse, et diverses directions de recherche sont apparues. Voici un petit recueil regroupant les questions principales, ainsi que quelques nouveaux puzzles à assembler ou casse-tête à résoudre.

L'exécution localisée en pratique ?

Les réseaux d'interaction forment un modèle de calcul qu'on pourrait qualifier de linéaire : une information atomique est matérialisée par un nœud et ses interconnexions, et chaque atome d'information qui va être produit au cours du calcul sera consommé une seule fois. On a vu dans la première partie que malgré cet aspect linéaire on pouvait effectuer dans ce modèle des opérations de duplication et d'effacement. De telles opérations sont incontournables lorsqu'on souhaite atteindre l'expressivité des langages de programmation habituels. Elles fournissent en particulier une méthode pour exécuter ces langages.

C'est cette méthode qui a été utilisée pour créer un prototype de machine d'exécution et qui est très différente des méthodes d'exécution traditionnelles. Dans une exécution traditionnelle les opérations de copie sont rarement explicites, on utilise à la place une spécificité des matériels informatiques d'aujourd'hui : ils permettent de lire plusieurs fois une information stockée en mémoire sans la détruire. Par comparaison, notre exécution localisée se revêtit de plusieurs atouts :

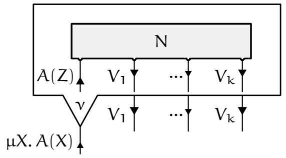
- Elle continuera de fonctionner si par hasard de nouvelles technologies offraient des matériels dépourvus de cette spécificité. On peut penser aux avancées du calcul quantique.
- Elle est adaptée au calcul parallèle, qui est encore très peu répandu malgré la présence de plusieurs unités de calcul dans la majorité des ordinateurs actuels. La difficulté est une fois de plus due à la possibilité de lire (ou écrire) plusieurs fois à un même emplacement, c'est cette possibilité qui soulève les problèmes inhérents à la programmation distribuée appelés « race conditions ».
- Elle évite (ou plutôt intègre naturellement) le processus compliqué de « garbage collection » inévitable avec les approches traditionnelles.

Les résultats donnés dans cette thèse ne donnent aucune information de complexité et n'offrent même pas un élément de réponse sur la question des performances. Il ne serait d'ailleurs pas étonnant que la méthode d'exécution « linéaire » introduise certains surcoûts pour un grand nombre de programmes, dont ceux qui utilisent abondamment et légitimement la spécificité matérielle suscitée. Un axe de recherche serait de modifier suffisamment le modèle des réseaux d'interaction pour pouvoir n'effectuer les opérations de réplication (δ, ε) seulement lorsqu'elles sont vraiment nécessaires, c'est-à-dire de façon paresseuse. Peut-on alors espérer profiter d'une exécution parallèle automatisée sans gaspiller la fonctionnalité fournie par nos machines actuelles ?

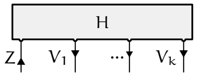
Quels réseaux pour les constructions inductives ?

Nous avons montré comment coder le langage *PCF* dans les réseaux et nous avons utilisé pour cela des boîtes de *récursion*. Nous avons longtemps envisagé de coder le *système T* en utilisant des constructions qui manipulent des types inductifs $\mu X.\, A(X)$ et coinductifs $\nu X.\, A(X)$ et qui permettent une récursion primitive tout en conservant une garantie de terminaison. Mais faute d'avoir obtenu un système suffisamment satisfaisant nous avons préféré attendre un peu plus de maturation avant d'en parler.

Une idée intéressante (mais simplifiée) consiste à autoriser librement le déroulage des types inductifs et coinductifs et de proposer un principe d'induction :



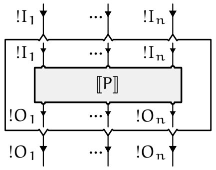
dans lequel la preuve N peut utiliser une variable de preuve particulière H, liée à la boîte qu'on définit. Cette dernière correspond à l'hypothèse d'induction sur un objet de type Z (variable fraîche) qui est un sous objet de l'objet générique de type $A(Z)$ traité. Cette hypothèse offre l'interface suivante :



Lors de la réduction, ces boîtes seront ouvertes et les hypothèses H qu'elle contient seront remplacées par de nouvelles copies de la boîte entière.

Simuler le π-calcul avec réplication ?

Dans la deuxième partie nous avons présenté des boîtes exponentielles généralisées au cadre différentiel. La motivation originelle était d'étendre le codage du π-calcul à la réplication (non gardée). Lors de la traduction d'un terme du π-calcul finitaire en un réseau, on fait en sorte qu'à chaque canal a_k du terme dont on est parti corresponde un couple de ports libres dont les types sont de modalité exponentielle, de la forme $?I_k^{\perp}$ et $!O_k$. Le traduit d'un terme P répliqué possède la même interface que le traduit du terme P lui-même et s'obtient simplement en le plaçant sa traduction dans une boîte de *super-promotion* (ou l'un de ses équivalents sémantiques) :



Comme pour les différents λ-calculs avec ressources, nous ne savons pas encore si on peut établir un lien (simulation, bissimulation, ou autre…) entre la réduction habituelle du π-calcul et la réduction des réseaux.

Les autres bénéfices d'une présentation de la logique avec des structures ?

Avec l'objectif de donner un critère de correction solide pour ces boîtes à plusieurs sorties, une présentation logique qui utilise des structures a été donnée. Son étude n'est pas complètement terminée, mais on peut déjà envisager d'autres utilisations de ce formalisme. En particulier, il semble fournir un éclairage nouveau sur la structure des réseaux d'interaction.

On pense pouvoir montrer directement (et facilement) grâce à lui les propriétés de nos réductions localisées (première partie) dont la preuve reposait jusqu'alors sur la normalisation de la logique linéaire. Il est aussi possible qu'on arrive grâce à lui à rassembler les travaux présentés dans les deux parties et définir un système où boîtes de *super-promotion* et boîtes de *réplication* sont localisées. L'idée serait dans les deux cas de rajouter une composante exponentielle aux structures :

$$\sigma \quad ::= \quad \cdot \;\mid\; \sigma\,;\sigma \;\mid\; \circ \;\mid\; \sigma\,,\sigma \;\mid\; [\sigma] \;\mid\; \{\sigma\} \;\mid\; A$$

les opérateurs de boîtes ε, δ, o, ι correspondraient alors à des règles qui manipulent cette composante :

$$\frac{[\omega]}{\cdot}\ \textit{effacement}\!\uparrow \qquad \frac{\circ}{\{\omega\}}\ \textit{effacement}\!\downarrow$$

$$\frac{[\omega]}{[\omega]\,;[\omega]}\ \textit{duplication}\!\uparrow \qquad \frac{\{\omega\}\,,\{\omega\}}{\{\omega\}}\ \textit{duplication}\!\downarrow$$

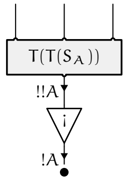
$$\frac{[\omega]}{\omega}\ \textit{ouverture}\!\uparrow \qquad \frac{\omega}{\{\omega\}}\ \textit{ouverture}\!\downarrow$$

$$\frac{[\omega]}{[[\omega]]}\ \textit{ingestion}\!\uparrow \qquad \frac{\{\{\omega\}\}}{\{\omega\}}\ \textit{ingestion}\!\downarrow$$

Une réalisabilité pour les nouvelles constructions différentielles ?

Nous avons généralisé la méthode de preuve par réalisabilité qui existait pour la logique linéaire au cas des constructions différentielles simples. Reste en suspens la possibilité d'étendre également celle-ci à l'opérateur *co-enfouissement* ou aux nouvelles boîtes. On pense en particulier à compléter la syntaxe qui nous a servi à générer les candidats de réductibilité pour la modalité exponentielle avec la construction suivante :

$$T(S_A) \quad ::= \quad \cdots \quad \mid$$



Bibliographie

[Plo77] Gordon D. PLOTKIN. "LCF Considered as a Programming Language". Dans : *Theoretical Computer Science* 5.3 (1977), pages 225–255. DOI : 10.1016/0304-3975(77)90044-5. (Cf. page 58).

[Hue80] Gérard P. HUET. "Confluent Reductions : Abstract Properties and Applications to Term Rewriting Systems". Dans : *J. ACM* 27.4 (1980), pages 797–821. DOI : 10.1145/322217.322230. (Cf. page 45).

[Gir87] Jean-Yves GIRARD. "Linear Logic". Dans : *Theoretical Computer Science* 50 (1987), pages 1–102. DOI : 10.1016/0304-3975(87)90045-4. (Cf. pages 10, 29, 128).

[DR89] Vincent DANOS et Laurent REGNIER. "The Structure of Multiplicatives". Dans : *Archive of Mathematical Logic* 28 (1989), pages 181–203. DOI : 10.1007/BF01622878. (Cf. page 18).

[GLT89] Jean-Yves GIRARD, Yves LAFONT et Paul TAYLOR. *Proofs and Types*. Cambridge Tracts in Theoretical Computer Science 7. Cambridge University Press, 1989. ISBN : 0-521-37181-3. (Cf. page 10).

[Dan90] Vincent DANOS. "La Logique Linéaire appliquée à l'étude de divers processus de normalisation (principalement du λ-calcul)". Thèse de doctorat. 1990. (Cf. page 17).

[Laf90] Yves LAFONT. "Interaction Nets". Dans : *Principles of Programming Languages (POPL '90)* (1990), pages 95–108. DOI : 10.1145/96709.96718. (Cf. page 10).

[Lam90] John LAMPING. "An algorithm for optimal lambda calculus reduction". Dans : *Principles of Programming Languages (POPL '90)* (1990), pages 16–30. DOI : 10.1145/96709.96711. (Cf. page 30).

[GAL92] Georges GONTHIER, Martín ABADI et Jean-Jacques LÉVY. "Linear Logic without Boxes". Dans : *Logic in Computer Science (LICS '92)*. IEEE Computer Society Press, 1992, pages 223–34. (Cf. page 30).

[SM92] Davide SANGIORGI et Robin MILNER. "The Problem of "Weak Bisimulation up to"". Dans : *Concurrency Theory (CONCUR '92)*. Springer-Verlag, 1992, pages 32–46. ISBN : 3-540-55822-5. (Cf. page 85).

[Bou93] Gérard BOUDOL. "The Lambda-Calculus with Multiplicities". Dans : *Concurrency Theory (CONCUR '93)*. Springer-Verlag, 1993, pages 1–6. ISBN : 3-540-57208-2. (Cf. page 79).

[MRA93] Ian MACKIE, Leopoldo ROMÁN et Samson ABRAMSKY. "An internal language for autonomous categories". Dans : *Proceedings of the First Imperial College Department of Computing Workshop on Theory and formal methods 1993*. Springer-Verlag, 1993, pages 235–246. ISBN : 0-387-19842-3. DOI : 10.1007/BF00873993. (Cf. page 21).

[Reg94] Laurent REGNIER. "An equivalence between lambda-terms". Dans : *Theoretical Computer Science* 126.2 (1994), pages 281–292. ISSN : 0304-3975. DOI : 10.1016/0304-3975(94)90012-4. (Cf. pages 25, 93).

[Laf95] Yves LAFONT. "From Proof Nets to Interaction Nets". Dans : *Advances in Linear Logic*. Sous la direction de J.-Y. GIRARD, Y. LAFONT et L. REGNIER. Cambridge University Press, 1995, pages 225–247. ISBN : 0-521-55961-8. (Cf. pages 10, 17).

[AC97] Andrea ASPERTI et Juliusz CHROBOCZEK. "Safe Operators : Brackets Closed Forever". Dans : *Applicable Algebra in Engineering Communication and Computing* 8.6 (1997), pages 437+. DOI : 10.1007/s002000050083. (Cf. page 42).

[Laf97] Yves Lafont. “Interaction Combinators”. Dans : *Information and Computation* 137.1 (1997), pages 69–101. issn : 0890-5401. doi : 10.1006/inco.1997.2643. (Cf. page 14).

[Gir98] Jean-Yves Girard. “Light linear logic”. Dans : *Journal of Information and Computation* 143.2 (1998), pages 175–204. issn : 0890-5401. doi : 10.1006/inco.1998.2700. (Cf. page 75).

[NC00] Uwe Nestmann et Benjamin C. Pierce. “Decoding Choice Encodings”. Dans : *Journal of Information and Computation* 163 (2000), pages 1–59. doi : 10.1006/inco.2000.9999. (Cf. page 85).

[Sou00] Jorge Sousa Pinto. “Sequential and Concurrent Abstract Machines for Interaction Nets”. Dans : *FOSSACS ’00 : Proceedings of the Third International Conference on Foundations of Software Science and Computation Structures*. Springer-Verlag, 2000, pages 267–282. isbn : 3-540-67257-5. doi : 10.1007/3-540-46432-8_18. (Cf. page 73).

[GM01] Stefano Guerrini et Andrea Masini. “Parsing MELL proof nets”. Dans : *Theoretical Computer Science* 254.1-2 (2001), pages 317–335. issn : 0304-3975. doi : 10.1016/S0304-3975(99)00299-6. (Cf. page 17).

[Sou01] Jorge Sousa Pinto. “Parallel Implementation Models for the lambda-Calculus Using the Geometry of Interaction”. Dans : *TLCA*. 2001, pages 385–399. doi : 0.1007/3-540-45413-6_30. (Cf. page 73).

[MS02] Ian Mackie et Jorge Sousa Pinto. “Encoding Linear Logic with Interaction Combinators”. Dans : *Information and Computation* 176.2 (2002), pages 153–186. issn : 0890-5401. doi : 10.1006/inco.2002.3163. (Cf. page 30).

[DJ03] Vincent Danos et Jean-Baptiste Joinet. “Linear logic and elementary time”. Dans : *Journal of Information and Computation* 183.1 (2003), pages 123–137. issn : 0890-5401. doi : 10.1016/S0890-5401(03)00010-5. (Cf. page 75).

[GMM03] Stefano Guerrini, Simone Martini et Andrea Masini. “Coherence for sharing proof-nets”. Dans : *Theoretical Computer Science* 294.3 (2003), pages 379–409. issn : 0304-3975. doi : 10.1016/S0304-3975(01)00162-1. (Cf. page 37).

[Mon03] Raphaël Montelatici. “Polarized Proof Nets with Cycles and Fixpoints Semantics”. Dans : *Typed Lambda Calculi and Applications*. Tome 2701. Lecture Notes in Computer Science. Springer, 2003, pages 256–270. doi : 10.1007/3-540-44904-3_18. (Cf. page 60).

[Str03] Lutz Strassburger. “MELL in the Calculus of Structures”. Dans : *Theoretical Computer Science* 309 (2003), pages 213–285. doi : 10.1016/S0304-3975(03)00240-8. (Cf. pages 115, 124).

[ER04] Thomas Ehrhard et Laurent Regnier. “The differential lambda-calculus”. Dans : *Theoretical Computer Science*. Tome 309. 1. Elsevier Science Publishers Ltd., 2004, pages 1–41. doi : 10.1016/S0304-3975(03)00392-X. (Cf. page 92).

[Laf04] Yves Lafont. “Soft linear logic and polynomial time”. Dans : *Theoretical Computer Science* 318.1-2 (2004), pages 163–180. doi : 10.1016/j.tcs.2003.10.018. (Cf. page 75).

[Mac04] Ian Mackie. “Efficient lambda-Evaluation with Interaction Nets”. Dans : *Rewriting Techniques and Applications*. 2004, pages 155–169. doi : 10.1007/b98160. (Cf. page 38).

[Ehr05] Thomas Ehrhard. “Finiteness spaces”. Dans : *Mathematical Structures in Computer Science* 15.4 (2005), pages 615–646. issn : 0960-1295. doi : 10.1017/S0960129504004645. (Cf. page 105).

[Mac05] Ian Mackie. “Interaction Net Implementation of Additive and Multiplicative Structures”. Dans : *Journal of Logic and Computation* 15.2 (2005), pages 219–237. issn : 0955-792X. doi : 10.1093/logcom/exi011. (Cf. page 53).

[Brü06] Kai Brünnler. “Deep Inference and Its Normal Form of Derivations”. Dans : *CiE*. Sous la direction d’Arnold Beckmann et al. Tome 3988. Lecture Notes in Computer Science. Springer, 2006, pages 65–74. isbn : 3-540-35466-2. doi : 10.1007/11780342_7. (Cf. page 119).

[ER06a] Thomas Ehrhard et Laurent Regnier. “Böhm trees, Krivine machine and the Taylor expansion of ordinary lambda-terms”. Dans : tome 3988. Lecture Notes in Computer Science. Springer-Verlag, 2006, pages 186–197. doi : 10.1007/11780342_20. (Cf. pages 88, 94).

[ER06b] Thomas Ehrhard et Laurent Regnier. “Differential interaction nets”. Dans : *Theoretical Computer Science* 364.2 (2006), pages 166–195. issn : 0304-3975. doi : 10.1016/j.tcs.2006.08.003. (Cf. pages 81, 105).

[EL07] Thomas Ehrhard et Olivier Laurent. “Interpreting a Finitary Pi-Calculus in Differential Interaction Nets”. Dans : *Concurrency Theory (CONCUR ’07)*. Tome 4703. Lecture Notes in Computer Science. Springer, septembre 2007, pages 333–348. doi : 10.1007/978-3-540-74407-8_23. (Cf. pages 82, 99).

[Maz07] Damiano Mazza. “A Denotational Semantics for the Symmetric Interaction Combinators”. Dans : *Mathematical Structures in Computer Science* 17.3 (2007). doi : 10.1017/S0960129507006135. (Cf. pages 13, 14).

[Pou07] Damien Pous. “New up-to techniques for weak bisimulation”. Dans : *Theoretical Computer Science* 380.1-2 (2007), pages 164–180. doi : 10.1016/j.tcs.2007.02.060. (Cf. page 85).

[Vau07a] Lionel Vaux. “The differential λμ-calculus”. Dans : *Theoretical Computer Science*. Tome 379. 1-2. Elsevier Science Publishers Ltd., 2007, pages 166–209. doi : 10.1016/j.tcs.2007.02.028. (Cf. page 92).

[Vau07b] Lionel Vaux. “λ-calcul différentiel et logique classique : interactions combinatoires”. Thèse de doctorat. 2007. url : http://www.lama.univ-savoie.fr/~vaux/pub/these (Cf. page 10).

[BL08] Denis Béchet et Sylvain Lippi. “Hard combinators”. Dans : *Theoretical Computer Science* 203.1 (2008), pages 31–48. issn : 1571-0661. doi : 10.1016/j.entcs.2008.03.032. (Cf. page 77).

[ER08] Thomas Ehrhard et Laurent Regnier. “Uniformity and the Taylor expansion of ordinary lambda-terms”. Dans : *Theoretical Computer Science*. Tome 403. 2-3. Elsevier Science Publishers Ltd., 2008, pages 347–372. doi : 10.1016/j.tcs.2008.06.001. (Cf. pages 79–81).

[Kri08] Jean-Louis Krivine. “A call-by-name lambda-calculus machine.” Dans : *Higher-Order and Symbolic Computation* 20.3 (12 mars 2008), pages 199–207. doi : 10.1007/s10990-007-9018-9. (Cf. page 94).

[Tra08] Paolo Tranquilli. “Intuitionistic Differential Nets and Lambda-Calculus”. Dans : *Theoretical Computer Science*. Elsevier Science Publishers Ltd., 2008. (Cf. page 87).

[Fal09a] Marc de Falco. “An Explicit Framework for Interaction Nets”. Dans : *Rewriting Techniques and Applications (RTA’09)*. Sous la direction de Ralf Treinen. Tome 5595. Lecture Notes in Computer Science. Springer, juin 2009, pages 209–223. doi : 10.1007/978-3-642-02348-4. (Cf. page 10).

[Fal09b] Marc de Falco. “Géométrie de l’ineraction et réseaux différentiels”. Thèse de doctorat. 2009. url : http://iml.univ-mrs.fr/~defalco/these.pdf. (Cf. page 10).

[Str09] Lutz Strassburger. “From Deep Inference to Proof Nets via Cut Elimination”. Dans : *Journal of Logic and Computation* (2009). doi : 10.2168/LMCS-4(1:9)2008. (Cf. page 123).

Liste des définitions et théorèmes

Résumé

Programmer, Calculer et Raisonner avec les Réseaux de la Logique Linéaire

La première partie propose divers systèmes de réseaux d'interaction (calcul par réécriture muni d'une réduction atomique, locale et parallèle) qui simulent l'exécution des preuves de la logique linéaire (considérées comme des programmes). Les différents fragments de cette logique sont abordés, on y ajoute aussi une récursion pour atteindre l'expressivité des langages de programmation usuels. Ce procédé de simulation permet d'exécuter certains langages à l'aide d'une petite machine d'exécution multi-processeurs. Il s'appuie sur des représentations localisées de boîtes issues des réseaux de preuve ; certaines utilisent avantageusement un canal de contrôle pour ne rien perdre de la structure des preuves représentées.

La deuxième partie parle de logique linéaire différentielle et de ses ressources à usage unique. On la munit d'une super-promotion, qui se distingue notamment d'une promotion ordinaire parce qu'elle préserve la symétrie originelle de ce formalisme. C'est la pendante côté logique de la réplication qu'on trouve parfois dans les algèbres de processus. On arrive à isoler l'un de ses composants plus primitifs, le co-enfouissement, responsable de leur dynamique incontrôlée (pour l'instant). Cette construction peut être exprimée dans la syntaxe du λ-calcul avec ressources ou dans un système de réseaux. La séquentialisation de ces derniers requiert une présentation originale de la logique, fondée sur un calcul de structures, et qui a potentiellement d'autres intérêts.

Il est aussi question de réalisabilité pour les systèmes différentiels et de sémantique relationnelle pour les divers réseaux présentés.

Abstract

Programming, Computation and Reasoning using Nets from Linear Logic

The first part describes various systems of interaction nets (calculus using rewriting whose reduction is atomic, local and parallel) which simulate the execution of linear logic proofs (considered as programs). The different fragments of this logic are discussed, and a recursion construct is added to reach the expressiveness of common programming languages. The simulation method that was developed allows the execution of programs written in certain languages using a small multi-processor virtual machine. It relies on localized representations of boxes coming from proof nets; some of them advantageously use a control channel to avoid losing structure from the proofs they represent.

The second part is devoted to differential linear logic and its single-use resources. It presents a super-promotion which, unlike the usual promotion, preserves the original symmetry of this formalism. This construction is the logical counterpart of the replication which can be found in some process algebras. We managed to isolate one of its more primitive components, namely co-digging, which is responsible for their (still) uncontrolled dynamic. Super-promotion can be expressed in the syntax of λ-calculus with resources or in systems of nets. Sequentialization of the latter requires a specific presentation of logic, based on a calculus of structures, which might open some other perspectives.

Realizability for differential systems and relational semantics for the various nets we consider are also discussed.