HAL
open science

De la construction de preuves à la programmation parallèle en logique linéaire

Guy Perrier

To cite this version:

Guy Perrier. De la construction de preuves à la programmation parallèle en logique linéaire. Autre [cs.OH]. Université Henri Poincaré - Nancy 1, 1995. Français. NNT : 1995NAN10022 . tel-01748580v1

HAL Id: tel-01748580

https://hal.univ-lorraine.fr/tel-01748580v1

Submitted on 29 Mar 2018 (v1), last revised 4 Apr 2016 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ DE LORRAINE

AVERTISSEMENT

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur. Ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document.

D'autre part, toute contrefaçon, plagiat, reproduction illicite encourt une poursuite pénale.

Contact : ddoc-theses-contact@univ-lorraine.fr

LIENS

Code de la Propriété Intellectuelle. articles L 122. 4
Code de la Propriété Intellectuelle. articles L 335.2- L 335.10
http://www.cfcopies.com/V2/leg/leg_droi.php
http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm

Université Henri Poincaré – Nancy I

UFR S.T.M.I.A.

Département de Formation Doctorale en Informatique
École Doctorale IAE + M

De la construction de preuves à la programmation parallèle en logique linéaire

THÈSE

présentée et soutenue publiquement le mercredi 25 janvier 1995

pour l'obtention du

Doctorat de l'Université Henri Poincaré – Nancy I
(Spécialité Informatique)

par

Guy Perrier

BIBLIOTHÈQUE UNIVERSITÉ NANCY 1 SCIENCES

Composition du jury

Président :	Jean-Pierre FINANCE
Rapporteurs :	Jean GALLIER Claude KIRCHNER Jacqueline VAUZEILLES
Examinateurs :	Jean-Marc ANDREOLI Didier GALMICHE

Centre de Recherche en Informatique de Nancy
INRIA–Lorraine

Résumé

Cette thèse se propose d'étudier la construction de preuves en logique linéaire et sur la base de cette étude, de l'utiliser ensuite comme paradigme de calcul pour la programmation parallèle.
Dans le calcul des séquents linéaire, nous montrons que la propriété de permutabilité d'inférences joue un rôle essentiel dans la normalisation des preuves en vue de leur construction automatique. Cette normalisation est déterminée par deux facteurs : le fragment logique dans lequel nous nous plaçons et le sens de construction des preuves envisagé. Nous mettons ainsi à jour une dualité entre principes de construction des preuves en chaînage avant et principes de construction des preuves en chaînage arrière à partir de laquelle nous proposons une méthode pour normaliser les preuves et élaborer des stratégies de construction dans un fragment donné de la logique linéaire. Appliquée de façon systématique dans la logique linéaire toute entière, elle nous permet d'y proposer une stratégie originale de construction des preuves en chaînage avant.
Cette méthode va ensuite nous aider à choisir un fragment approprié de la logique linéaire intuitionniste pour y élaborer un calcul de processus asynchrone, CPL (Concurrent Programming Logic), basé sur la construction de preuves. Les formules y sont interprétées comme des processus et les preuves, avec certaines restrictions, comme des réductions de processus. Nous proposons une sémantique opérationnelle sous forme d'une relation de transition qui découle de cette interprétation.
Pour exprimer la capacité d'interaction des processus avec le monde extérieur, nous proposons une sémantique dénotationnelle, issue de la sémantique des phases de la logique linéaire, qui repose sur les notions de co-processus et d'interface. Un co-processus est un testeur qui caractérise un aspect de l'interaction possible du processus avec l'extérieur. Il est représenté par une formule qui vérifie une certaine forme de complémentarité avec la formule représentant le processus. Une interface est un ensemble de co-processus caractérisant de manière globale la capacité d'interaction d'un processus avec l'extérieur. Utilisant le cadre déductif, nous construisons un calcul sur les interfaces qui s'articule autour des concepts de réduction et de relativisation d'une interface.
La notion de communication synchrone est captée par une simple restriction sur la forme des preuves dans le système déductif de CPL. Cette restriction permet de définir un calcul synchrone, SCPL qui englobe notamment le π-calcul de Milner.
En tant que modèle logique du parallélisme, CPL peut être un outil d'analyse dans le domaine et peut servir de base à de nouvelles propositions de langages de programmation parallèle. On peut même d'emblée concevoir CPL comme un langage de programmation logique. Enfin, il peut constituer un cadre commun pour l'étude sémantique de programmes écrits dans des langages les plus divers.

Mots-clés: logique linéaire, déduction, programmation logique, parallélisme, communication, calcul de processus.

Remerciements

Je voudrais tout d'abord remercier Didier Galmiche, mon directeur de thèse. Ses conseils, ses critiques constructives m'ont aidé et stimulé tout au long de mon activité de recherche. En particulier, j'ai apprécié sa rigueur dans le travail.

Merci à Jacqueline Vauzeilles, Jean Gallier et Claude Kirchner qui ont accepté de consacrer leur temps à rapporter cette thèse.

Je suis reconnaissant à Jean-Pierre Finance d'avoir accepté de présider le jury et à Jean-Marc Andreoli d'avoir accepté d'y participer.

Merci à Denis Roegel que j'ai sollicité un nombre incalculable de fois pour ses compétences LaTeX et qui m'a aidé dans la mise en forme du document de thèse.

Merci enfin à Mariola pour sa patience.

Table des matières

Introduction **9**

Partie I Construction de preuves en logique linéaire **17**

Chapitre 1 Introduction à la logique linéaire **19**

1 La logique linéaire 19

2 Le calcul des séquents linéaire 20

3 Les principaux sous-systèmes 22

Chapitre 2 Concepts pour la normalisation de preuves en logique linéaire **29**

1 Vocabulaire relatif à l'évolution d'une formule dans une preuve 30

2 Permutabilité d'inférences 32

3 Mouvements d'inférences 35

4 Vers des preuves normales 45

Chapitre 3 Normalisation dans CLL et preuves en chaînage arrière **53**

1 Traitement des contractions. 54

2 Traitement des affaiblissements 56

3 CLL$\uparrow$: un système pour la construction ascendante des preuves 57

4 Normalisation des preuves dans CLL$\uparrow$ 59

Chapitre 4 Normalisation dans CLL et preuves en chaînage avant **63**

1 Séquents affaiblissables : le système d'inférence CLL$\downarrow$ 64

2 Formules de base et d'affaiblissement 68

3 Descente des inférences produisant les A-formules 71

4 Base d'affaiblissement associée à un séquent. 73

5 CLL$\downarrow_\Delta$: un système spécialisé pour prouver $\vdash \Delta$ 74

6 Preuves normales dans CLL$\downarrow_\Delta$ 79

Chapitre 5 Stratégies de recherche de preuves **83**

1 Composition et de décomposition immédiates et en chaîne 84

2 Stratégies de construction de preuves en chaînage arrière 86

3 Stratégies de construction de preuves en chaînage avant 92
4 Implantation 100

Partie II Programmation parallèle en logique linéaire 101

Chapitre 6 De la construction de preuves au calcul de processus 103
1 Une question essentielle : le choix d'un fragment logique approprié 103
2 Les grands choix syntaxiques à l'origine de CPL 104
3 Le système formel de CPL 107

Chapitre 7 CPL : un calcul de processus asynchrone 115
1 Le langage de CPL 116
2 Sémantique opérationnelle 125

Chapitre 8 La sémantique dénotationnelle de CPL 141
1 Notion d'interface d'un processus 142
2 Processus et interfaces linéaires 145
3 Comparaison de processus 151

Chapitre 9 SCPL : une restriction synchrone de CPL 159
1 Stratégies de preuves synchrones : le système formel de SCPL 160
2 De la synchronisation intégrée à la syntaxe des processus 163
3 La sémantique opérationnelle de SCPL 166
4 La sémantique dénotationnelle de SCPL 168
5 Traduction du Π-calcul dans SCPL 171

Chapitre 10 CPL :un cadre commun pour l'étude sémantique de programmes 177
1 Spécification du problème en Prolog et traduction dans CPL 177
2 Un programme Prolog plus efficace 178
3 Un programme Pascal 179
4 L'utilisation de CPL comme langage de programmation parallèle 181

Conclusion 185

Bibliographie 189

Annexe A Permutabilité d'inférences dans CLL 197

Annexe B Elimination des coupures dans CLL 205

Annexe C Descente des inférences produisant les A-formules dans CLL↓ 211

Annexe D Transposition des résultats de permutabilité de CLL dans LL 215

Annexe E Réduction de l'interface relative d'un processus à une interface linéaire 217

Introduction

L'exigence de plus en plus grande des utilisateurs de disposer de programmes corrects par rapport aux spécifications à satisfaire a conduit à rechercher des méthodes de programmation basées sur la *théorie de la démonstration.*

Construction de preuves et programmation

Une première approche s'inscrivant dans le cadre de *la programmation fonctionnelle*, consiste à traduire une spécification par une formule et à extraire d'une *preuve constructive* de cette formule le *programme* qui va réaliser la spécification [Nordström *et al.*, 1990; Constable, 1991; Galmiche, 1992; Coquand *et al.*, 1994]. Ce programme se présente sous forme d'une fonction et son exécution va consister à lui soumettre en argument des données, elles-mêmes sous forme de fonctions, pour calculer la valeur correspondante retournée. A un niveau logique, cela se traduit par une composition de preuves par la règle de la coupure puis par une *réduction de ces coupures* dans la preuve résultant de cette composition. Cette vision des choses puise ses racines dans l'isomorphisme de Curry-Howard [Howard, 1980] mais elle a vraiment acquis ses lettres de noblesse avec différentes théories comme la théorie des types de Martin-Löf [Martin-Löf, 1982] ou le calcul des constructions [Coquand and Huet, 1988].
Dans cette thèse, nous avons choisi une autre approche qui s'inscrit dans le cadre de *la programmation logique* [Miller *et al.*, 1991]. Dans cette approche, un *programme* est représenté par une *formule logique.* Ce que nous appellerons son environnement d'exécution est aussi représenté à l'aide de formules logiques. L'ensemble, programme et environnement, va constituer ce que nous appellerons une requête qui va se présenter aussi sous forme d'un énoncé logique (qui peut se traduire par une formule ou un séquent). L'exécution de la requête va consister à construire une preuve de cet énoncé.
Le noyau logique pur de Prolog par exemple, peut être vu de cette façon dans le cadre de la logique intuitionniste. Un programme est représenté par la conjonction P des implications traduisant les clauses de Horn qui le constituent. Un environnement d'exécution de P est un but qui est représenté par une conjonction B de formules atomiques. La requête associée peut alors se traduire par le séquent $P \vdash B$ et son exécution par la recherche d'une preuve de celui-ci dans le calcul des séquents intuitionniste.
Alors que dans la première approche, *la construction de preuves* correspond à *la synthèse de programmes*, elle traduit ici *l'exécution de programmes.*
L'avantage d'une telle approche est qu'elle rend l'écriture des programmes relativement aisée car proche des spécifications. Un même objet peut être vu comme une spécification ou un programme logique selon qu'il est considéré sous l'angle de sa signification logique ou de son exécution. Par ailleurs, le fait de se situer d'emblée dans un cadre logique, est un atout pour la vérification de la correction des programmes. Mais le revers de la médaille est qu'il est souvent difficile de contrôler l'exécution des programmes logiques et de faire en sorte qu'ils soient le plus efficaces possibles. On peut toujours améliorer l'efficacité par l'introduction dans l'écriture des programmes de prédicats de contrôle comme cela se fait avec Prolog, mais on sort alors du cadre logique pur. Voyons dans quelle mesure il n'est pas possible *d'intégrer le contrôle au niveau logique.*

Expression du changement d'état en logique

Si nous concevons l'exécution d'un programme comme une suite de changements d'états d'une mémoire, intégrer le contrôle au niveau logique, c'est être capable d'exprimer la notion de changement d'état en logique. Voyons tout d'abord ce qu'il en est en logique classique ou intuitionniste.
Considérons un exemple simple, celui d'un programme qui modifie la valeur d'une variable x, la faisant passer de 2 à 5. On peut modéliser un tel programme par l'implication $(x = 2) \Rightarrow (x = 5)$, son entrée et sa sortie par les formules respectives $(x = 2)$ et $(x = 5)$ (elles constituent l'environnement d'exécution). L'exécution du programme sera alors représenté par une preuve du séquent :

$$(x = 2) \Rightarrow (x = 5),\ (x = 2) \ \vdash\ (x = 5).$$

La partie gauche peut être vue comme un état initial constitué du programme et de son entrée et la partie droite comme un état final constitué par la sortie du programme. La preuve du séquent peut alors être considérée comme une action permettant de passer de l'état initial à l'état final. Le séquent est

prouvable en logique intuitionniste ce qui peut être interprété comme le fait que l'action est possible. Malheureusement celui-ci aussi :

$$(x = 2) \Rightarrow (x = 5),\ (x = 2) \vdash (x = 2) \wedge (x = 5).$$

La proposition $(x = 2)$, posée comme vraie en hypothèse, le reste ensuite indéfiniment alors que l'on souhaiterait qu'elle soit remplacée par $(x = 5)$.
A travers cet exemple, la logique intuitionniste (et par là même la logique classique aussi) révèle son incapacité à exprimer simplement une transition d'états. Cela est lié au caractère *monotone* de la vérité dans cette logique au sens où si elle peut être construite dans une preuve, elle ne peut pas y être détruite.
A l'opposé, un trait essentiel de *la logique linéaire* [Girard, 1987, 1989] est sa *capacité à exprimer le dynamisme*, c'est-à-dire à prendre en compte la notion de changement d'état. Ainsi reprenons l'exemple précédent. L'implication linéaire se notant $\multimap$, l'exécution du programme est représentée par la preuve du séquent :

$$(x = 2) \multimap (x = 5),\ (x = 2) \vdash (x = 5).$$

Mais en logique linéaire, les formules sont considérées comme des *ressources* qui sont *consommées* dans les preuves au moment de leur utilisation. Elles ne peuvent donc servir qu'une seule fois sauf si elles sont précédées de l'opérateur modal ! qui indique qu'il s'agit de ressources infinies. C'est pourquoi le séquent

$$(x = 2) \multimap (x = 5),\ (x = 2) \vdash (x = 2) \otimes (x = 5)$$

n'est pas prouvable ($\otimes$ représente une forme de conjonction en logique linéaire). Cette aptitude de la logique linéaire à modéliser le changement d'état est bien illustrée par les travaux de [Masseron *et al.*, 1993; Masseron, 1993] sur la *planification*. Il y est montré qu'on peut voir la construction de plans comme celle de preuves dans une théorie issue d'un fragment particulier de la logique linéaire.

Programmation logique en logique linéaire

Cette possibilité qu'offre la logique linéaire de gérer dans une preuve les formules comme des ressources dont on sait a priori qu'elles sont consommables ou infinies, a donné l'occasion de définir des langages de programmation logique à partir de fragments de celle-ci.
[Miller *et al.*, 1987] ont défini un cadre général pour une extension de Prolog, qui repose sur les *preuves uniformes*. Utilisé en logique intuitionniste, il leur permet d'avoir des buts qui soient des implications. Un programme peut ainsi se trouvé enrichi de faits nouveaux en cours d'exécution. [Hodas and Miller, 1991,1994] ont ensuite appliqué ce cadre à un fragment de la logique linéaire intuitionniste. La distinction entre ressources consommables et ressources infinies a alors permis d'enrichir les capacités d'expression du langage le rendant propre à des applications telles que l'analyse syntaxique de langages naturels, la gestion d'une base de données ou la programmation orientée objets. [Harland and Pym, 1991,1992, 1994] ont effectué un travail analogue dans un fragment voisin.
[Andreoli and Pareschi, 1990b,1991b] ont cherché eux aussi à étendre la capacité d'expression de Prolog mais dans une autre direction par *l'introduction de têtes multiples dans les clauses*. Ils ont ainsi été amenés à travailler dans un fragment restreint de la logique linéaire classique (CLL). Le langage qu'ils ont conçu, LO [Andreoli and Pareschi, 1991b], peut être vu comme une extension de Prolog. Mais il peut être aussi considéré comme un langage de programmation orientée objets [Andreoli and Pareschi, 1991a; Andreoli *et al.*, 1991] dans la mesure où les buts successifs engendrés au cours de la recherche peuvent être considérés comme des agents qui évoluent, peuvent donner naissance à de nouveaux agents héritant de leurs propriétés ; ces agents peuvent aussi communiquer par le biais d'un mécanisme extra-logique, celui du partage d'un tableau.

Logique linéaire et parallélisme

Les travaux de [Andreoli and Pareschi, 1991a; Andreoli *et al.*, 1991] révèlent que le parallélisme est une dimension naturelle de la programmation logique en logique linéaire.
Pour mettre en évidence ce lien, prenons un nouvel exemple : celui d'un programme qui consiste à modifier

l'état de deux variables x et y, la première passant de la valeur 2 à 5 et la seconde de 3 à 7 par exemple. Ses deux instructions peuvent être représentées par les deux implications linéaires $(x = 2) \multimap (x = 5)$ et $(y = 3) \multimap (y = 7)$. Une exécution peut être modélisée par la construction d'une preuve du séquent :

$$(x = 2) \multimap (x = 5),\ (y = 3) \multimap (y = 7),\ (x = 2),\ (y = 3)\ \vdash (x = 5) \otimes (y = 7)$$

Les deux instructions qui composent le programme sont totalement indépendantes donc elles peuvent être exécutées en parallèle. La syntaxe d'un langage impératif tel que Pascal, masquerait cette indépendance en imposant un ordre entre les instructions mais ce n'est pas le cas en logique linéaire. D'un point de vue syntaxique, il n'y a déjà pas d'ordre entre les formules qui les représentent dans le séquent. Ensuite dans la construction de la preuve qui traduit l'exécution du programme, les deux formules sont réduites de façon totalement indépendante si bien que la preuve peut être dissociée en deux : une preuve du séquent $(x = 2) \multimap (x = 5),\ (x = 2) \vdash (x = 5)$ et une preuve du séquent $(y = 3) \multimap (y = 7),\ (y = 3) \vdash (y = 7)$. Considérons maintenant un autre programme qui fait passer successivement une variable x de la valeur 2 à 5 puis 7. Son exécution peut être représenté en logique linéaire par la construction d'une preuve du séquent :

$$(x = 2) \multimap (x = 5),\ (x = 5) \multimap (x = 7),\ (x = 2)\ \vdash (x = 7)$$

Les deux instructions du programme ne sont pas indépendantes : la première, représentée par $(x = 2) \multimap (x = 5)$, doit être nécessairement exécutée avant l'autre, représentée par $(x = 5) \multimap (x = 7)$. Mais il est inutile de traduire ce lien dans la syntaxe des deux formules. Le lien apparaîtra dans la preuve par une impossibilité de dissocier la réduction des deux formules. Il peut être exprimé en termes de communication par la transmission de l'information $(x = 5)$ entre la formule $(x = 2) \multimap (x = 5)$ représentant l'émetteur et la formule $(x = 5) \multimap (x = 7)$ représentant le récepteur.
Ainsi apparaît l'aptitude de la logique linéaire à exprimer la logique interne de l'exécution d'un programme, l'*indépendance* entre deux instructions se manifestant par le *parallélisme* et la *causalité* par la *communication*. Les travaux sur la modélisation des *réseaux de Petri* en logique linéaire illustrent cette capacité à représenter le parallélisme. [Asperti, 1987; Gunter and V., 1989; Martí-Oliet and Meseguer, 1989] ont montré que le fragment multiplicatif (MLL) était un cadre adapté à la représentation des réseaux de Petri. Plus récemment, [Engberg and Winskel, 1994] ont établi que ceux-ci pouvaient même constituer un modèle de la logique linéaire intuitionniste toute entière.

Logique linéaire et calcul de processus

Les calculs de processus tels que CSP [Hoare, 1985], CCS [Milner, 1980] ou le Π-calcul [Milner *et al.*, 1992] jouent un rôle important dans la modélisation du parallélisme. Or, de la programmation logique en logique linéaire telle que nous l'avons présentée, à un calcul de processus , il n'y a qu'un pas. Les *formules* au lieu d'être vues comme des programmes, le seront comme des *processus*. Les *opérations logiques* sur ces formules seront considérées comme des *opérations algébriques* sur les processus. Enfin, la *construction de preuves* représentera la *réduction des processus* au lieu de l'exécution des programmes.
[Miller, 1992] a ouvert cette voie en montrant que le Π-calcul pouvait, avec certaines restrictions, être considéré comme une théorie en logique linéaire. Dans un fragment de CLL, [Kobayashi and Yonezawa, 1992, 1994a] ont développé ACL, un calcul de processus asynchrone qu'ils ont récemment étendu à l'ordre supérieur sous le nom de HACL [Kobayashi and Yonezawa, 1994b]. Les travaux de [Volpe, 1994], même s'ils sont tournés davantage vers l'application à un langage de programmation logique, sont très proches. V. Saraswat et P. Lincoln ont au départ une motivation différente des auteurs précédents, celle de représenter la programmation parallèle avec contraintes à l'aide de la logique linéaire. Mais le calcul qu'ils ont conçu, HLcc [Lincoln and Saraswat, 1992], se situe encore dans la même famille.
Notre thèse se situe dans le fil de ces travaux. Nous nous proposons d'étudier dans quelle mesure il est possible d'*interpréter la construction de preuves en logique linéaire comme un calcul de processus parallèles, d'une manière qui soit la plus générale possible.* Cette étude va déboucher sur la proposition de *CPL (Concurrent Programming Logic)*, un calcul de processus parallèles basée sur cette idée.
La volonté d'être le plus méthodique et le plus général possible, nous a amenés à un *examen préalable et systématique de la construction de preuves en logique linéaire* indépendamment de l'application envisagée,

et ce pour plusieurs raisons :

- Le problème d'identifier la construction de preuves à un calcul de processus ne peut pas être posé comme tel dans la logique linéaire toute entière. *Le choix d'un fragment logique adéquat est essentiel*: il faut qu'on puisse y *normaliser les preuves* de telle façon qu'elles prennent un sens dans le calcul de processus et qu'elles puissent être construites facilement pour que le calcul puisse être mis en œuvre pratiquement.

- Une fois le fragment logique choisi, le calcul défini comme une construction de preuves normales dans celui-ci, la connaissance de la construction de preuves en logique linéaire doit nous permettre d'élaborer des stratégies particulières de calcul. Nous verrons ainsi qu'il est une forme de stratégie qui joue un rôle important : *les stratégies synchrones.*

- Enfin, nous nous attacherons à définir une *sémantique dénotationnelle* issue de la sémantique des phases de la logique linéaire [Girard, 1987] qui vise à exprimer la capacité d'interaction des processus avec le monde extérieur et qui repose sur la déduction en logique linéaire.

Pour toutes ces raisons, une étude préalable de la construction de preuves en logique linéaire s'imposait. Naturellement, cette étude nous a emmené au-delà de ce dont nous avions strictement besoin pour notre application. Elle s'inscrit plus généralement dans *la théorie de la démonstration basée sur le calcul des séquents.*

Construction de preuves en logique linéaire

Le formalisme utilisé pour définir le système déductif de la logique linéaire [Girard, 1987] est celui du *calcul des séquents* [Gentzen, 1935; Kleene, 1968; Gallier, 1986]. Il est tout à fait adapté à la représentation d'un calcul de processus dans la mesure où un séquent $P \vdash P'$ peut être interprété comme résumant la réduction d'un processus P en un processus P'.
Sur le plan purement logique, l'avantage d'un tel formalisme par rapport à celui de la *déduction naturelle* est que chaque règle d'inférence est *globale*: elle permet de prendre en compte l'ensemble des formules présentes dans une preuve et pas seulement celles qui vont être modifiées par la règle. Le revers de la médaille est qu'un tel formalisme est redondant [Wallen, 1990]. Notamment, *l'ordre des inférences* au sein d'une preuve n'est pas toujours pertinent. L'idée qui en découle, est qu'il est possible de normaliser les preuves en permutant certaines de leurs inférences entre lesquelles l'ordre n'est pas essentiel. [Kleene, 1952] avait déjà montré tout l'intérêt de la permutabilité d'inférences dans les preuves en logique classique et intuitionniste présentées dans le formalisme du calcul des séquents. [Wallen, 1990] a généralisé ces résultats pour des logiques non classiques.
Et cette notion se retrouve dans la plupart des travaux sur la déduction en logique linéaire.

- [Hodas and Miller, 1991, 1994], pour étendre Prolog comme nous l'avons indiqué précédemment, ont été amenés à étudier la construction ascendante de preuves dans un fragment particulier de la logique linéaire intuitionniste (ILL). Ils ont montré qu'elles pouvaient y être normalisées sous forme de *preuves uniformes.* [Harland and Pym, 1991,1992, 1994] ont mené une étude similaire dans un fragment voisin.

- [Andreoli and Pareschi, 1990b; Andreoli, 1992] ont étudié la construction ascendante de preuves dans la logique linéaire complète présentée à l'aide de séquents sans partie gauche (CLL). L'utilisation de la permutabilité d'inférences a permis de définir une classe de preuves complète relativement à l'ensemble des preuves : les *"focusing proofs"* [Andreoli, 1992]. Leur construction de haut en bas présente deux propriétés caractéristiques : *la réversibilité* qui permet de décomposer certaines formules dès qu'elles apparaissent dans certains séquents et le *"focusing"* qui, dès qu'on commence à décomposer d'autres formules, permet d'enchaîner par la décomposition de leurs sous-formules.

- [Lincoln and Shankar, 1994] se sont concentrés sur le problème de la gestion des substitutions relatives aux variables quantifiées, dans la construction ascendante de preuves de CLL. Reprenant la technique de *skolémisation dynamique* utilisée pour la logique intuitionniste [Shankar, 1992],

ils ont pu rendre totalement permutable l'introduction des quantificateurs existentiels et universels dans les preuves. L'inconvénient est que cela restreint la permutabilité de certaines autres inférences.

- [Tammet, 1993], en plus de chercher à construire des preuves de bas en haut (autrement dit en chaînage arrière), a aussi choisi de les construire en partant des axiomes pour arriver à la conclusion (c'est-à-dire en chaînage avant). La méthode utilisée s'apparente à la *résolution* et s'inspire de la méthode inverse de Maslov [Lifschitz, 1989]. Mais au-delà de la forme, le fond repose encore sur la permutabilité d'inférences dans le calcul des séquents. [Mints, 1993] a étendu ces résultats au premier ordre.

Le fil conducteur de ces travaux est *l'utilisation de la permutabilité d'inférences dans les preuves.* C'est ce qui nous a amenés à *systématiser* cette utilisation. Notre démarche n'est pas avant tout dans l'obtention de tel ou tel résultat pointu qui viendrait s'ajouter à ceux qui viennent d'être exposés. Elle est de montrer que ces résultats qui peuvent apparaître indépendants les uns des autres, se rattache en fait à *une méthode générale de recherche de preuve, basée sur la permutabilité d'inférences, qui s'applique à tout fragment de la logique linéaire et à tout sens de construction des preuves.*

Plan de la thèse

Partie I: construction de preuves en logique linéaire

Le chapitre 1 est une introduction à la logique linéaire tournée plus particulièrement vers la présentation de son système déductif et de ses principaux sous-systèmes.

Dans le chapitre 2, nous nous attacherons à définir clairement les concepts de base d' *inférences permutables* et de *montée et descente d'une inférence dans une preuve.*
Ensuite, nous aborderons la *normalisation des preuves* proprement dite qui se résume à un mouvement d'inférences, avec deux facteurs déterminant ce mouvement : le *fragment logique* dans lequel se situent les preuves et le *sens de construction* envisagé [Galmiche and Perrier, 1994b]. L'élimination des coupures peut être considérée comme une première phase dans ce processus de normalisation.

Dans le chapitre 3, nous appliquerons les principes dégagés juste avant, à la normalisation des preuves dans CLL en vue de les construire en *chaînage arrière.* Nous allons découper le processus en deux phases : la première concerne uniquement les contractions et les affaiblissements et va déboucher sur une spécialisation du système d'inférence qui va devenir CLL$\uparrow$; la deuxième porte sur le déplacement des inférences logiques et nous permettra d'obtenir les preuves normales de CLL$\uparrow$. Nous verrons que, pour l'essentiel, nous retrouverons les "focusing proofs" de [Andreoli, 1992]. Mais comme nous l'avons déjà dit, ce qui nous importe ici, n'est pas tant le résultat obtenu que la mise en valeur de la méthode utilisée.

Dans le chapitre 4, nous allons reprendre CLL mais dans une perspective opposée : la construction des preuves en *chaînage avant.* Là encore, nous allons découper le processus de normalisation en plusieurs phases. Reprenant la notion de *séquent affaiblissable* introduite par [Tammet, 1993], nous allons faire une première modification du système d'inférence qui va nous permettre de rendre mobiles les affaiblissements de tous ordres, produits au cours d'une preuve.
Dans le nouveau système, CLL$\downarrow$, nous allons chercher à rendre les affaiblissements contrôlables dans un processus de construction des preuves en chaînage avant. [Tammet, 1993] a apporté une première réponse à ce problème crucial. L'utilisation systématique de la permutabilité d'inférences va nous permettre d'aller plus loin en faisant appel à des notions nouvelles telles que *formules de base* et *formules d'affaiblissement* ainsi que *base d'affaiblissement.*
Cela va se traduire par une deuxième modification du système d'inférence qui va devenir le système CLL$\downarrow_\Delta$, spécialisé dans la démonstration d'un séquent particulier $\vdash \Delta$. Outre le contrôle des affaiblissements, ce système va aussi intégrer celui des contractions. Il ne restera plus ensuite qu'à déplacer les inférences logiques pour obtenir des preuves normales de CLL$\downarrow_\Delta$.

Dans les trois précédents chapitres, nous avons envisagé les preuves sous un angle statique. Il s'agissait d'objets auxquels nous avons fait subir un processus de transformation : la normalisation. Dans le chapitre 5, nous allons les considérer sous un angle dynamique, celui de leur construction. Les propriétés des preuves normales vont se traduire au niveau de la recherche par des principes qui vont permettre de réduire l'indéterminisme lié à l'ordre d'application des règles d'inférence.
La montée maximum de certaines inférences dans les preuves donnera lieu aux principes duaux de *composition immédiate* pour la recherche descendante et de *décomposition en chaîne* pour la recherche ascendante. Symétriquement, la descente maximum d'autres inférences va se traduire par les principes duaux de *composition en chaîne* pour la recherche descendante et de *décomposition immédiate* pour la recherche ascendante [Galmiche and Perrier, 1994a].

Partie II : programmation parallèle en logique linéaire

Dans le chapitre 6, nous exposons la démarche qui va nous permettre d'utiliser notre connaissance de la construction des preuves en logique linéaire pour élaborer CPL, un calcul de processus qui s'appuie sur celle-ci.
Nous montrerons tout d'abord les choix auxquels nous avons été confrontés dans la *délimitation d'un fragment logique approprié.* Nous verrons dans quelle mesure ils rejoignent ou se différencient de ceux effectués par [Miller, 1992; Kobayashi and Yonezawa, 1992; Lincoln and Saraswat, 1992]. Le fragment logique étant fixé, nous utiliserons la méthode développée dans la partie I, pour *normaliser les preuves* dans ce fragment et spécialiser le système d'inférence pour en faire le *système déductif de CPL.* Nous ferons enfin une brève étude de la permutabilité d'inférences dans celui-ci qui aidera ensuite à l'élaboration des stratégies de calcul ainsi qu'à l'étude de la sémantique dénotationnelle.

Le chapitre 7 est consacré à l'étude du *langage* et de la *sémantique opérationnelle* de CPL. Nous commencerons par décrire les mécanismes de communication ainsi que les opérations algébriques sur les processus que le langage permet. Nous illustrerons ses possibilités d'expression par des exemples variés. A partir du système déductif de CPL, nous étudierons ensuite sa sémantique opérationnelle sous deux formes : *logique* en tant que construction de preuves et à travers une *relation de transition* qui en découle.

Le chapitre 8 est consacré à l'étude de la *sémantique dénotationnelle de CPL.* Elle vise à exprimer la capacité d'interaction d'un processus avec le monde extérieur par opposition à son mécanisme interne de réduction. Elle est issue de la sémantique des phases de la logique linéaire [Girard, 1987] et basée sur les concepts de *co-processus* et d'*interface* dont nous donnerons une représentation logique. Nous verrons ensuite qu'en s'appuyant sur la déduction en logique linéaire, il est possible d'élaborer un calcul sur les interfaces basé sur les notions de *réduction*, et de *relativisation* d'une interface. Nous étudierons enfin comment s'articulent les relations entre interfaces et les relations entre processus correspondants étant donné qu'il y a une dualité entre les deux.

Dans le chapitre 9, nous nous intéressons à la *synchronisation de la communication* dans les réductions de processus et plus généralement dans les preuves de CPL. Nous verrons qu'elle s'effectue en rapprochant les inférences produisant les messages de celles les consommant.
Cette opération qui n'est pas toujours possible, nous permettra d'obtenir des *preuves synchrones.* En nous intéressant ensuite seulement à celles-ci, nous allons restreindre CPL à un *calcul synchrone* : SCPL (Synchronous Concurrent Programming Logic). Nous verrons en particulier que nous pourrons y coder le Π-calcul [Milner, 1991; Milner *et al.*, 1992].

Dans le chapitre 10, nous montrerons comment CPL fournit un cadre approprié pour traduire dans un même langage des programmes écrits dans des styles très différents : programmes logiques, programmes impératifs ou programmes parallèles. Avec sa sémantique dénotationnelle, il peut ainsi fournir un *cadre commun pour l'étude sémantique de programmes* écrits dans des langages les plus divers.

Première partie

Construction de preuves en logique linéaire

Chapitre 1

Introduction à la logique linéaire

Ce chapitre constitue un rappel des principaux traits de la logique linéaire, une présentation de son système déductif et des principaux sous-systèmes de celui-ci.

1 La logique linéaire

Bien que sa formulation soit récente [Girard, 1987], la logique linéaire se situe dans le prolongement de logiques plus anciennes : celle de Lambek [Lambek, 1958], les logiques directe [Ketonen and Weyhrauch, 1984] et pertinente [Urquhart, 1984; Dunn, 1986]. Elle est issue d'une réflexion sur la sémantique de l'implication intuitionniste basée sur les espaces cohérents. J.Y. Girard a introduit ceux-ci pour simplifier les domaines de Scott [Girard, 1986]. Il découvrit alors que l'implication intuitionniste n'est pas une opération primitive mais qu'elle peut être décomposée en deux autres qui vont devenir les opérations $\multimap$ et ! de la logique linéaire. Comment la logique linéaire se situe-t-elle par rapport à celles dont elle est issue?
La *logique classique* est le reflet d'un monde *statique* où les vérités sont éternelles.
La *logique intuitionniste* [Heyting, 1966; Brower, 1975; ?] marqua une première rupture dans cette conception en introduisant une idée *constructive* de la vérité : ce qui est vrai, est ce qui peut être calculé comme tel. Elle place donc les preuves au centre de la notion de vérité. Mais elle ne rompt que partiellement avec une vision statique de la réalité dans la mesure où la vérité y reste monotone : au cours d'une preuve, une assertion qui a été établie, peut ensuite être utilisée à volonté, elle reste vraie jusqu'au bout.
La *logique linéaire* se veut par contre, une logique du *dynamisme* [Girard, 1989; Troelstra, 1992]. Les preuves peuvent être vues comme des *actions* transformant des assertions, pouvant être interprétées comme des états, en d'autres.
Pour illustrer cette idée, considérons l'exemple suivant. Le séquent

$$(x = 2) \multimap (x = 5),\ (x = 2) \vdash (x = 5)$$

est prouvable en logique linéaire tandis que

$$(x = 2) \multimap (x = 5),\ (x = 2) \vdash (x = 2) \otimes (x = 5)$$

ne l'est pas (rappelons que $\multimap$ et $\otimes$ représentent respectivement l'implication linéaire et une forme de conjonction). On peut considérer ces deux séquents comme deux actions, leur partie gauche représentant un état initial et leur partie droite, l'état final correspondant après exécution de l'action. La ressource $(x = 2)$ ayant été consommée dans l'action, cela entraîne sa disparition dans l'état final et c'est donc pour cela que le second séquent n'est pas prouvable.
Ainsi, on a pu dire de la logique linéaire qu'elle est une *logique consciente de ses ressources*. C'est pourquoi, elle a déjà pu trouver des applications qui font appel à cette qualité dans des domaines aussi divers que la programmation fonctionnelle, la programmation logique, le parallélisme et la programmation orientée objets (le lecteur trouvera une liste exhaustive de références dans [Alexiev, 1993]).
Dans son papier initial, [Girard, 1987] propose deux sémantiques de la logique linéaire : celle des *espaces cohérents* et celle des *phases*. Nous nous intéresserons à cette dernière à l'occasion de la définition d'une sémantique dénotationnelle pour CPL dans le chapitre 8.
D'autres sémantiques ont vu le jour depuis, basées sur les *jeux* [Lafont and Streicher, 1991; Blass, 1992;

Abramsky and Jagadeesan, 1992], les *catégories* [Seely, 1989; de Paiva, 1989; Barr, 1991] ou les *modèles de Kripke* [Allwein and Dunn, 1992].
Dans cette thèse, c'est l'aspect déductif de la logique linéaire qui nous intéresse avant tout. Or, de ce point de vue, [Girard, 1987] en propose deux présentations : sous forme d'un *calcul de séquents* et sous forme de *réseaux de preuve* [Gallier, 1992a,1992b]. La deuxième s'apparente au formalisme de la déduction naturelle mais les preuves y ont la forme d'arbres reliés entre eux par leurs feuilles. Nous opterons pour la première qui est la plus appropriée pour représenter un calcul de processus, les séquents permettant de traduire naturellement les changements d'états des processus.

2 Le calcul des séquents linéaire

Avant d'en présenter les règles, précisons quelques notations. Nous utiliserons les méta-variables A, B, C pour représenter des formules atomiques, F, G, H pour des formules quelconques et Δ, Γ, Σ pour des multi-ensembles de formules ; ces méta-variables pourront être éventuellement indexées.
Les règles de déduction définissant tout calcul des séquents, sont en général classées en trois groupes : *identité, structure, logique*. Et ce sont les règles structurelles qui déterminent la nature d'une logique.

2.1 L'abandon des règles d'affaiblissement et de contraction

En logique linéaire, les formules sont considérées comme des ressources qui sont produites et consommées. Une formule ne peut y être utilisée qu'une fois et une seule. C'est pourquoi *les règles du calcul des séquents linéaire s'obtiennent à partir de celles du calcul des séquents classique par suppression des règles structurelles d'affaiblissement et de contraction.*
Les règles d'affaiblissement gauche et droite se présentent habituellement ainsi :

$$\frac{\Gamma \vdash \Delta}{F, \Gamma \vdash \Delta}\ w_L \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash F, \Delta}\ w_R$$

L'abandon de la règle w_L signifie que dans une démonstration, toutes les hypothèses sont consommées et celui de w_R qu'il n'est pas possible d'introduire dans une conclusion une alternative arbitraire.
Les règles de contraction, quant à elles, ont la forme suivante :

$$\frac{F, F, \Gamma \vdash \Delta}{F, \Gamma \vdash \Delta}\ c_L \qquad \frac{\Gamma \vdash F, F, \Delta}{\Gamma \vdash F, \Delta}\ c_R$$

L'abandon de la règle c_L veut dire que dans une démonstration, une ressource ne peut pas être utilisée plusieurs fois et celui de c_R qu'il n'est pas possible dans une conclusion de contracter plusieurs alternatives identiques.
La règle d'échange est la seule règle structurelle maintenue mais de façon implicite : alors qu'en logique classique,les parties gauche et droite des séquents sont des *ensembles* de formules, en logique linéaire, ce sont des *multi-ensembles* de formules. Si la règle d'échange est abandonnée, on obtient une logique linéaire non commutative [Yetter, 1990, Abrusci, 1990, 1991].

2.2 La négation linéaire

Contrairement à la logique intuitionniste, la modification des règles structurelles ne change pas les propriétés de la négation par rapport à la logique classique. Elle est définie par les règles suivantes :

$$\frac{\Gamma \vdash F, \Delta}{F^\perp, \Gamma \vdash \Delta}\ \perp_L \qquad \frac{F, \Gamma \vdash \Delta}{\Gamma \vdash F^\perp, \Delta}\ \perp_R$$

Ainsi, elle conserve un caractère *involutif*, c'est-à-dire que $F^{\perp\perp}$ est égal à F. Cela confère donc à la logique linéaire une *symétrie* que n'a pas la logique intuitionniste. Cela permet, entre autres, de transformer tous les séquents en séquents sans partie gauche où l'on ne distingue plus hypothèses et conclusion. Par exemple : $F, G^\perp \vdash H$ pourra s'écrire $\vdash F^\perp, G, H$. Dans la suite, nous allons utiliser cette propriété pour simplifier le système d'inférence.

2.3 La dissociation des aspects multiplicatif et additif des opérateurs logiques

Une des conséquence de l'abandon des règles d'affaiblissement et de contraction est la dissociation du *et* et du *ou* classiques chacun en un opérateur *multiplicatif* et un opérateur *additif*. Ceci va donner lieu a deux fragments distincts de la logique linéaire : le fragment multiplicatif et le fragment additif.

2.3.1 Le fragment multiplicatif

La caractéristique de celui-ci est que les règles d'inférences n'y modifient pas l'étendue des ressources manipulées, elles agissent seulement sur leur structuration ; de là, le caractère *intensionnel* ou *multiplicatif* de ces règles. Le *et multiplicatif* s'appelle *times* (notation $\otimes$). On lui associe un élément neutre qui est la constante 1. Les deux opérateurs sont alors définis par les règles suivantes :

$$\frac{F,\ G,\ \Gamma \vdash \Delta}{F \otimes G,\ \Gamma \vdash \Delta}\ \otimes_L \qquad \frac{\Gamma_1 \vdash F,\ \Delta_1 \qquad \Gamma_2 \vdash G,\ \Delta_2}{\Gamma_1,\ \Gamma_2 \vdash F \otimes G,\ \Delta_1,\ \Delta_2}\ \otimes_R$$

$$\frac{\Gamma \vdash \Delta}{1,\ \Gamma \vdash \Delta}\ 1_L \qquad \frac{}{\vdash 1}\ 1_R$$

Le *ou multiplicatif* s'appelle *par* (notation $\wp$). On lui associe un élément neutre qui est la constante $\perp$. Les deux opérateurs sont alors définis par les règles suivantes :

$$\frac{F,\ \Gamma_1 \vdash \Delta_1 \qquad G,\ \Gamma_2 \vdash \Delta_2}{F \wp G,\ \Gamma_1,\ \Gamma_2 \vdash \Delta_1,\ \Delta_2}\ \wp_L \qquad \frac{\Gamma \vdash F,\ G,\ \Delta}{\Gamma \vdash F \wp G,\ \Delta}\ \wp_R$$

$$\frac{}{\perp \vdash}\ \perp_L \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \perp,\ \Delta}\ \perp_R$$

Le fragment multiplicatif comprend aussi l'implication linéaire qui peut être considéré comme un connecteur dérivé de $\wp$ et $^{\perp}$ de la façon suivante : $F \multimap G \equiv F^{\perp} \wp G$. Mais elle peut être aussi définie directement par les règles suivantes :

$$\frac{\Gamma_1 \vdash F,\ \Delta_1 \qquad G,\ \Gamma_2 \vdash \Delta_2}{F \multimap G,\ \Gamma_1,\ \Gamma_2 \vdash \Delta_1,\ \Delta_2}\ \multimap_L \qquad \frac{F,\ \Gamma \vdash G,\ \Delta}{\Gamma \vdash F \multimap G,\ \Delta}\ \multimap_R$$

2.3.2 Le fragment additif

Il se caractérise par des règles d'inférence qui modifient l'étendue des ressources : superposition de contextes ou introduction d'une formule totalement nouvelle, d'où le caractère *extensionnel* ou *additif* de ses règles. Ainsi le *et* additif s'appelle *with* et s'écrit &. On lui associe un élément neutre qui est la constante $\top$. Ils sont définis par les règles suivantes :

$$\frac{F,\ \Gamma \vdash \Delta}{F \& G,\ \Gamma \vdash \Delta}\ \&1_L \qquad \frac{G,\ \Gamma \vdash \Delta}{F \& G,\ \Gamma \vdash \Delta}\ \&2_L \qquad \frac{\Gamma \vdash F, \Delta \qquad \Gamma \vdash G,\ \Delta}{\Gamma \vdash F \& G,\ \Delta}\ \&$$

$$\frac{}{\Gamma \vdash \top,\ \Delta}\ \top_R$$

De façon duale, on trouve le *ou* additif qu'on appelle *plus* (notation $\oplus$). L'élément neutre correspondant est la constante 0. Les deux opérateurs sont définis par les règles suivantes :

$$\frac{F,\ \Gamma \vdash \Delta \qquad G,\ \Gamma \vdash \Delta}{F \oplus G,\ \Gamma \vdash \Delta}\ \oplus_L \qquad \frac{\Gamma \vdash F,\ \Delta}{\Gamma \vdash F \oplus G,\ \Delta}\ \oplus 1_R \qquad \frac{\Gamma \vdash G,\ \Delta}{\Gamma \vdash F \oplus G,\ \Delta}\ \oplus 2_R$$

$$\frac{}{0,\ \Gamma \vdash \Delta}\ 0_L$$

On peut montrer que l'ensemble des formules de la logique linéaire muni d'une des quatre opérations binaires ainsi définies constitue un monoïde commutatif. Cette propriété sera souvent utilisée de façon implicite par la suite.

2.4 Les opérateurs exponentiels

Pour retrouver le pouvoir d'expression de la logique classique, il était nécessaire de réintroduire les règles d'affaiblissement et de contraction mais de façon contrôlée en marquant les formules auxquelles elles sont applicables à l'aide des opérateurs modaux ! et ?. D'où les règles suivantes :

$$\frac{F,\ \Gamma \vdash \Delta}{!F,\ \Gamma \vdash \Delta}\ !_L \qquad \frac{!\Gamma \vdash F,\ ?\Delta}{!\Gamma \vdash !F,\ ?\Delta}\ !_R \qquad \frac{\Gamma \vdash \Delta}{!F,\ \Gamma \vdash \Delta}\ w!_L \qquad \frac{!F,\ !F,\ \Gamma \vdash \Delta}{!F,\ \Gamma \vdash \Delta}\ c!_L$$

$$\frac{F,\ !\Gamma \vdash ?\Delta}{?F,\ !\Gamma \vdash ?\Delta}\ ?_L \qquad \frac{\Gamma \vdash F,\ \Delta}{\Gamma \vdash ?F,\ \Delta}\ ?_R \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash ?F,\ \Delta}\ w?_R \qquad \frac{\Gamma \vdash ?F,\ ?F,\ \Delta}{\Gamma \vdash ?F,\ \Delta}\ c?_R$$

Lorsqu'on trouve ci-dessus des expressions de la forme $?\Gamma$ *ou* $!\Gamma$, *cela indique des multi-ensembles obtenus en ajoutant devant chaque formule de* Γ, *l'opérateur ? ou !.*

2.5 La règle de la coupure

Elle est fondamentale car elle donne le pouvoir d'utiliser des lemmes dans les preuves et s'énonce de la façon suivante :

$$\frac{\Gamma_1 \vdash F,\ \Delta_1 \qquad F,\ \Gamma_2 \vdash \Delta_2}{\Gamma_1,\ \Gamma_2 \vdash \Delta_1,\ \Delta_2}\ cut$$

Mais paradoxalement, elle n'a de sens que s'il est possible de l'éliminer dans les preuves (Hauptsatz de Gentzen). En effet, elle a pour seul rôle de permettre l'introduction de lemmes dans les démonstrations. Elle ne doit pas ajouter de sens particulier par rapport aux autres règles qui, elles, servent à définir la logique. [Girard, 1987] a montré que l'élimination des coupures était possible en logique linéaire.

On a récapitulé l'ensemble des règles constituant le système déductif de la logique linéaire que nous noterons LL dans la figure 1.1. Le fragment propositionnel de LL sera noté LL_0.

Remarque 1.2.1 *On peut ranger les règles d'inférences de LL en deux groupes : celles qui exigent des conditions sur les contextes et que l'on peut qualifier de* sensibles au contexte *et celles qui ne le sont pas. Dans le premier, on trouve tout d'abord la règle & qui exige l'identité des deux contextes initiaux. On trouve aussi la règle ! qui exige que le contexte ait la forme* $?\Delta$. *Enfin, ce groupe comprend aussi la règle* $\forall$ *qui exige que la variable qui est quantifiée ne soit pas libre dans le contexte.*
Cette remarque est importante parce qu'elle explique premièrement la difficulté à passer d'un formalisme sous forme de calcul des séquents à un formalisme du type déduction naturelle pour représenter la logique linéaire. C'est tout le débat sur les réseaux de preuve qui dépasse le cadre de cette thèse (voir à ce sujet [Girard, 1987; Bellin, 1991; Gonthier et al., *1992]).*

3 Les principaux sous-systèmes

Du point de vue de la construction de preuves, LL est un cadre trop large pour y développer des stratégies de recherche de preuves efficaces. A ce sujet, il est à noter que, contrairement à la logique propositionnelle classique, LL_0 n'est pas décidable [Lincoln *et al.*, 1990]. Il est donc capital de pouvoir se restreindre à certains fragments de LL [Chirimar and Lipton, 1991; Bellin and Ketonen, 1992]. Quelques critères semblent importants pour leur définition :

- LL doit être une *extension conservative* de ce fragment, c'est-à-dire que tout séquent appartenant à celui-ci et prouvable dans LL, doit avoir une preuve totalement incluse dans celui-ci ;
- la recherche de preuve doit y être *efficace* ; ce point est crucial et va être développé dans le chapitre suivant ;
- son *pouvoir d'expression* doit être suffisamment étendu pour pouvoir y spécifier les problèmes qu'on souhaite résoudre.

FIG. 1.1 - *Système d'inférence de LL*

groupe identité

$$\frac{}{A \vdash A}\ id \qquad \frac{\Gamma_1 \vdash F, \Delta_1 \qquad F, \Gamma_2 \vdash \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\ cut$$

groupe logique

négation

$$\frac{\Gamma \vdash F, \Delta}{F^\perp, \Gamma \vdash \Delta}\ \perp_L \qquad \frac{F, \Gamma \vdash \Delta}{\Gamma \vdash F^\perp, \Delta}\ \perp_R$$

opérateurs multiplicatifs

$$\frac{F, G, \Gamma \vdash \Delta}{F \otimes G, \Gamma \vdash \Delta}\ \otimes_L \qquad \frac{\Gamma_1 \vdash F, \Delta_1 \qquad \Gamma_2 \vdash G, \Delta_2}{\Gamma_1, \Gamma_2 \vdash F \otimes G, \Delta_1, \Delta_2}\ \otimes_R \qquad \frac{\Gamma \vdash \Delta}{1, \Gamma \vdash \Delta}\ 1_L \qquad \frac{}{\vdash 1}\ 1_R$$

$$\frac{F, \Gamma_1 \vdash \Delta_1 \qquad G, \Gamma_2 \vdash \Delta_2}{F \wp G, \Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\ \wp_L \qquad \frac{\Gamma \vdash F, G, \Delta}{\Gamma \vdash F \wp G, \Delta}\ \wp_R \qquad \frac{}{\perp \vdash}\ \perp_L \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \perp, \Delta}\ \perp_R$$

$$\frac{\Gamma_1 \vdash F, \Delta_1 \qquad G, \Gamma_2 \vdash \Delta_2}{F \multimap G, \Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\ \multimap_L \qquad \frac{F, \Gamma \vdash G, \Delta}{\Gamma \vdash F \multimap G, \Delta}\ \multimap_R$$

opérateurs additifs

$$\frac{F, \Gamma \vdash \Delta}{F \& G, \Gamma \vdash \Delta}\ \&1_L \qquad \frac{G, \Gamma \vdash \Delta}{F \& G, \Gamma \vdash \Delta}\ \&2_L \qquad \frac{\Gamma \vdash F, \Delta \qquad \Gamma \vdash G, \Delta}{\Gamma \vdash F \& G, \Delta}\ \&_R \qquad \frac{}{\Gamma \vdash \top, \Delta}\ \top_R$$

$$\frac{F, \Gamma \vdash \Delta \qquad G, \Gamma \vdash \Delta}{F \oplus G, \Gamma \vdash \Delta}\ \oplus_L \qquad \frac{\Gamma \vdash F, \Delta}{\Gamma \vdash F \oplus G, \Delta}\ \oplus 1_R \qquad \frac{\Gamma \vdash G, \Delta}{\Gamma \vdash F \oplus G, \Delta}\ \oplus 2_R \qquad \frac{}{0, \Gamma \vdash \Delta}\ 0_L$$

opérateurs exponentiels

$$\frac{F, \Gamma \vdash \Delta}{!F, \Gamma \vdash \Delta}\ !_L \qquad \frac{!\Gamma \vdash F, ?\Delta}{!\Gamma \vdash !F, ?\Delta}\ !_R \qquad \frac{\Gamma \vdash \Delta}{!F, \Gamma \vdash \Delta}\ w!_L \qquad \frac{!F, !F, \Gamma \vdash \Delta}{!F, \Gamma \vdash \Delta}\ c!_L$$

$$\frac{F, !\Gamma \vdash ?\Delta}{?F, !\Gamma \vdash ?\Delta}\ ?_L \qquad \frac{\Gamma \vdash F, \Delta}{\Gamma \vdash ?F, \Delta}\ ?_R \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash ?F, \Delta}\ w?_R \qquad \frac{\Gamma \vdash ?F, ?F, \Delta}{\Gamma \vdash ?F, \Delta}\ c?_R$$

quantificateurs

$$\frac{F[t/x], \Gamma \vdash \Delta}{\forall x F, \Gamma \vdash \Delta}\ \forall_L \qquad \frac{\Gamma \vdash F[y/x], \Delta}{\Gamma \vdash \forall x F, \Delta}\ \forall_R$$

$$\frac{F[y/x], \Gamma \vdash \Delta}{\exists x F, \Gamma \vdash \Delta}\ \exists_L \qquad \frac{\Gamma \vdash F[t/x], \Delta}{\Gamma \vdash \exists x F, \Delta}\ \exists_R$$

Pour les règles $\forall_R$ et $\exists_L$, la variable y ne doit pas être libre dans la conclusion

Les deux derniers critères sont souvent contradictoires si bien que l'on peut privilégier plus ou moins l'un ou l'autre.

3.1 La logique linéaire classique (CLL)

Comme nous l'avons déjà vu, les propriétés de la négation linéaire vont nous permettre de simplifier le système complet par une restriction aux séquents sans partie gauche. Tout séquent de la forme $\Gamma \vdash \Delta$ peut être remplacé par $\vdash \Gamma^{\perp}, \Delta$ (où $\Gamma^{\perp}$ représente le multi-ensemble des négations des formules de Γ). Cela permet de diviser le nombre de règles par deux mais pour que LL en soit une extension conservative, il faut aussi redéfinir la négation à l'aide de règles indiquant comment la descendre au niveau atomique dans les formules composées. Nous obtenons alors le système CLL décrit par la figure 1.2.
Il est bien entendu équivalent à LL, ce qui signifie que tout séquent $\Gamma \vdash \Delta$ est prouvable dans LL si et seulement si $\vdash \Gamma^{\perp}, \Delta$ est prouvable dans CLL. L'implication linéaire n'y apparaît pas explicitement vu que $F \multimap G$ est considéré comme une abréviation de $F^{\perp} \wp\, G$, cela permet à LL d'être une extension conservative de CLL. La restriction propositionnelle de CLL sera notée CLL_0.

3.2 Les fragments multiplicatif (MLL) et multiplicatif-additif (MALL)

Le fragment multiplicatif de CLL, noté MLL, s'obtient par une restriction syntaxique sur les formules : les seuls opérateurs permis sont 1, $\perp$, $^{\perp}$, $\otimes$, $\wp$, $\forall$, $\exists$. A cause de la propriété de la sous-formule, CLL est une extension conservative d'un tel fragment. L'implication linéaire, du fait qu'elle est composée à partir de $\wp$ et de $^{\perp}$ est aussi un opérateur implicite de MLL. La restriction propositionnelle de MLL sera notée MLL_0, fragment NP-complet [Kanovich, 1991] comme le fragment encore plus restreint construit uniquement à partir des constantes $\perp$ et 1 et des opérateurs multiplicatif [Lincoln and Winkler, 1992]. L'intérêt de MLL est que l'étendue des ressources atomiques est conservée dans les preuves si bien que leur construction y est particulièrement simple.
En étendant MLL aux opérateurs additifs 0, $\top$, $\&$, $\oplus$, on obtient le fragment de CLL que nous qualifierons de multiplicatif-additif et qui est noté MALL. Pour les mêmes raisons que pour MLL, CLL est une extension conservative de MALL. La restriction propositionnelle de MALL sera notée MALL_0. Le premier fragment est NEXPTIME-hard [Lincoln and Scedrov, 1992] alors que le second est PSPACE-complet [Lincoln *et al.*, 1990].
Notons que dans MALL, l'affaiblissement et la contraction des ressources sont permis implicitement au travers des opérateurs additifs mais de manière finie en ce sens que la forme syntaxique d'un séquent quelconque de MALL permet de déterminer un ensemble maximum et fini de ressources nécessaire à sa preuve. Par exemple, si le séquent $\vdash A,\ B\&C,\ D\&E$ est prouvable, la formule A sera au maximum dupliquée deux fois car il y a deux connecteurs $\&$ dans le séquent.

3.3 La logique linéaire intuitionniste (ILL)

De la même façon que la logique intuitionniste s'obtient à partir de la logique classique en limitant le nombre de formules dans la partie droite des séquents à une au maximum, on définit la logique linéaire intuitionniste, notée ILL, à partir de la logique linéaire toute entière. ILL joue un rôle important dans les applications à la programmation fonctionnelle, la programmation logique et le parallélisme [Alexiev, 1993].
LL est-elle une extension conservative de ce fragment? Si l'on passe en revue toutes les règles d'inférence, on constate que dans la construction d'une preuve de bas en haut, certaines risquent de nous faire sortir de ILL ; ce sont : $^{\perp}{}_L$, $\multimap_L$, $\wp_R$, $c?_R$. Pour ne pas avoir à utiliser certaines de ces règles, il suffit de remplacer dans les séquents à prouver certaines formules par des formules équivalentes.
Ainsi, on remplacera les formules de la forme $F^{\perp}$, $F \wp G$, $?F$ respectivement par :

$$F \multimap \perp, \quad (F \multimap \perp) \multimap G, \quad (!F) \multimap \perp.$$

Malheureusement, on n'a pas éliminé la règle $\multimap_L$ et si l'on étudie les choses de plus près, on constate que l'on sort de ILL uniquement lorsque l'application de la règle $\multimap_L$ est consécutive à celle de 0_L ou de $\perp_L$. Ainsi les séquents $\vdash (((A \multimap \perp) \multimap \perp) \multimap A$ et $\vdash ((A \multimap B) \multimap 0) \multimap (A \otimes \top)$ ne sont pas prouvables dans ILL [Lincoln, 1992].

FIG. 1.2 - *Le système d'inférence de CLL*

groupe identité

$$\frac{}{\vdash A, A^{\perp}}\ id \qquad\qquad \frac{\vdash F, \Delta_1 \quad \vdash F^{\perp}, \Delta_2}{\vdash \Delta_1, \Delta_2}\ cut$$

groupe logique

négation

$$F^{\perp\perp} = F$$

$$(F \otimes G)^{\perp} = F^{\perp} \wp G^{\perp} \qquad (F \wp G)^{\perp} = F^{\perp} \otimes G^{\perp}$$

$$1^{\perp} = \perp \qquad \perp^{\perp} = 1$$

$$(F \& G)^{\perp} = F^{\perp} \oplus G^{\perp} \qquad (F \oplus G)^{\perp} = F^{\perp} \& G^{\perp}$$

$$\top^{\perp} = 0 \qquad 0^{\perp} = \top$$

$$(!F)^{\perp} = ?F^{\perp} \qquad (?F)^{\perp} = !F^{\perp}$$

$$(\forall x F)^{\perp} = \exists x F^{\perp} \qquad (\exists x F)^{\perp} = \forall x F^{\perp}$$

opérateurs multiplicatifs

$$\frac{\vdash F_1, \Delta_1 \quad \vdash F_2, \Delta_2}{\vdash F_1 \otimes F_2, \Delta_1, \Delta_2}\ \otimes \qquad \frac{}{\vdash 1}\ 1 \qquad \frac{\vdash F_1, F_2, \Delta}{\vdash F_1 \wp F_2, \Delta}\ \wp \qquad \frac{\vdash \Delta}{\vdash \perp, \Delta}\ \perp$$

opérateurs additifs

$$\frac{\vdash F_1, \Delta \quad \vdash F_2, \Delta}{\vdash F_1 \& F_2, \Delta}\ \& \qquad \frac{}{\vdash \top, \Delta}\ \top \qquad \frac{\vdash F_1, \Delta}{\vdash F_1 \oplus F_2, \Delta}\ \oplus_1 \qquad \frac{\vdash F_2, \Delta}{\vdash F_1 \oplus F_2, \Delta}\ \oplus_2$$

opérateurs exponentiels

$$\frac{\vdash F, ?\Delta}{\vdash !F, ?\Delta}\ ! \qquad \frac{\vdash F, \Delta}{\vdash ?F, \Delta}\ ? \qquad \frac{\vdash \Delta}{\vdash ?F, \Delta}\ w? \qquad \frac{\vdash ?F, ?F, \Delta}{\vdash ?F, \Delta}\ c?$$

quantificateurs

$$\frac{\vdash F[y/x], \Delta}{\vdash \forall x F, \Delta}\ \forall \qquad \frac{\vdash F[t/x], \Delta}{\vdash \exists x F, \Delta}\ \exists$$

avec y non libre dans la conclusion

Si l'on exclut les constantes 0 et $\perp$, on obtient un fragment dont LL est une extension conservative. La restriction des règles d'inférence à celui-ci forment alors un système, noté ILL', défini par la figure 1.3 et que l'on appellera par abus de langage système de la logique linéaire intuitionniste, même s'il n'en est qu'une restriction. La partie propositionnelle d'ILL' sera notée ILL'_0.

FIG. 1.3 - *Système d'inférence d'ILL'*

groupe identité

$$\frac{}{A \vdash A}\ id \qquad\qquad \frac{\Gamma_1 \vdash F \quad F, \Gamma_2 \vdash G}{\Gamma_1, \Gamma_2 \vdash G}\ cut$$

groupe logique

opérateurs multiplicatifs

$$\frac{F_1, F_2, \Gamma \vdash G}{F_1 \otimes F_2, \Gamma \vdash G}\ \otimes_L \qquad \frac{\Gamma_1 \vdash F_1 \quad \Gamma_2 \vdash F_2}{\Gamma_1, \Gamma_2 \vdash F_1 \otimes F_2}\ \otimes_R \qquad \frac{\Gamma \vdash F}{1, \Gamma \vdash F}\ 1_L \qquad \frac{}{\vdash 1}\ 1_R$$

$$\frac{\Gamma_1 \vdash F_1 \quad F_2, \Gamma_2 \vdash G}{F_1 \multimap F_2, \Gamma_1, \Gamma_2 \vdash G}\ \multimap_L \qquad \frac{F_1, \Gamma \vdash F_2}{\Gamma \vdash F_1 \multimap F_2}\ \multimap_R$$

opérateurs additifs

$$\frac{F_1, \Gamma \vdash G}{F_1 \& F_2, \Gamma \vdash G}\ \&1_L \qquad \frac{F_2, \Gamma \vdash G}{F_1 \& F_2, \Gamma \vdash G}\ \&2_L \qquad \frac{\Gamma \vdash F_1 \quad \Gamma \vdash F_2}{\Gamma \vdash F_1 \& F_2}\ \&_R \qquad \frac{}{\Gamma \vdash \top}\ \top_R$$

$$\frac{F_1, \Gamma \vdash G \quad F_2, \Gamma \vdash G}{F_1 \oplus F_2, \Gamma \vdash G}\ \oplus_L \qquad \frac{\Gamma \vdash F_1}{\Gamma \vdash F_1 \oplus F_2}\ \oplus 1_R \qquad \frac{\Gamma \vdash F_2}{\Gamma \vdash F_1 \oplus F_2}\ \oplus 2_R$$

opérateur exponentiel

$$\frac{F, \Gamma \vdash G}{!F, \Gamma \vdash G}\ !_L \qquad \frac{!\Gamma \vdash F}{!\Gamma \vdash !F}\ !_R \qquad \frac{\Gamma \vdash G}{!F, \Gamma \vdash G}\ w!_L \qquad \frac{!F, !F, \Gamma \vdash G}{!F, \Gamma \vdash G}\ c!_L$$

quantificateurs

$$\frac{F[t/x], \Gamma \vdash G}{\forall x F, \Gamma \vdash G}\ \forall_L \qquad \frac{\Gamma \vdash F[y/x]}{\Gamma \vdash \forall x F}\ \forall_R$$

$$\frac{F[y/x], \Gamma \vdash G}{\exists x F, \Gamma \vdash G}\ \exists_L \qquad \frac{\Gamma \vdash F[t/x]}{\Gamma \vdash \exists x F}\ \exists_R$$

avec y non libre dans la conclusion pour les règles $\forall_R$ et $\exists_L$

Chapitre 2

Concepts pour la normalisation de preuves en logique linéaire

Introduction

Dans ce chapitre, nous nous proposons d'étudier la première phase qui va nous permettre de construire efficacement des preuves sans coupures dans un fragment donné de LL : *la normalisation des preuves* issues de ce fragment. Elle vise à définir une classe de preuves suffisamment restreinte pour que leur construction soit efficace, tout en restant complète pour le fragment considéré : tout séquent prouvable doit pouvoir admettre une preuve normale. Pour y parvenir, nous proposons d'utiliser systématiquement la *permutabilité d'inférences* au sein des preuves [Kleene, 1952].
Cette notion n'est pas nouvelle et on la retrouve de façon informelle dans la plupart des travaux sur la déduction en logique linéaire [Harland and Pym, 1990; Hodas and Miller, 1991; Bellin, 1991; Andreoli, 1992; Lincoln, 1992; Tammet, 1993]. Mais nous voulons la définir rigoureusement car la nature de la règle & qui entraîne la superposition de deux contextes complique les choses. Il en est de même pour la notion de *mouvement d'une inférence dans une preuve* qui en découle et qui est un élément constitutif du processus de normalisation.
Pour établir ces concepts, il nous est nécessaire d'analyser l'évolution d'une formule dans une preuve en calcul des séquents. En effet, en déduction naturelle, une formule est liée d'un côté aux composantes dont elle est issue, et de l'autre, à la formule dans la composition de laquelle elle entre. Dans le calcul des séquents, une formule qui a été produite par une inférence, peut rester passive lors des inférences suivantes et ne devenir active en entrant dans la composition d'une nouvelle formule que bien longtemps après. Nous définirons le vocabulaire pour décrire ce phénomène.
Cela va nous permettre ensuite d'établir de façon précise ce qu'on entend par inférences permutables dans une preuve. A partir de cela, nous allons mener une étude exhaustive de la *permutabilité de deux inférences* dans CLL en fonction de leurs types. Par itération de la permutation de deux inférences dans une preuve vers le bas ou vers le haut, nous pourrons définir *la montée et la descente d'une inférence dans une preuve.*
Nous étudierons ensuite deux questions : dans quel sens peut-on déplacer plus facilement une inférence et jusqu'où ? Nous pouvons en venir ensuite tout naturellement au *processus de normalisation des preuves* proprement dit qui n'est qu'un ensemble de mouvement d'inférences. Nous nous sommes intéressés à deux questions : quels facteurs peuvent déterminer ce processus et les mouvements individuels qui le composent, sont-ils cohérents entre eux ?
Nous achevons ce chapitre par une première application de ces concepts à *l'élimination des coupures* qui constitue une première phase du processus de normalisation.

1 Vocabulaire relatif à l'évolution d'une formule dans une preuve

Afin de définir rigoureusement les notions de permutabilité et de mouvement d'inférences, il est nécessaire de fixer un peu de vocabulaire relatif au calcul des séquents qui permette de décrire l'évolution des formules dans les preuves. Etant donné que nous allons par la suite travailler dans CLL, nous nous limiterons donc à des séquents sans partie gauche. Il serait facile d'étendre ce vocabulaire à des séquents avec parties gauche et droite. Etudions d'abord ce qui se passe au niveau local d'une inférence, l'impact qu'elle a sur l'évolution des formules d'une preuve. Cela nous amène à introduire les concepts de *formule principale*, *formule active* et *contexte*.

1.1 Formules principales, formules actives et contextes d'une inférence

Rappelons tout d'abord qu'une *inférence* est une instance d'une règle d'inférence et que le *type* d'une inférence est le nom de la règle correspondante. Par ailleurs, quand nous parlons de formule pour un multi-ensemble, il s'agit en fait d'occurrence de formule ; par exemple, le multi-ensemble { A, A} contient deux formules qui sont identiques syntaxiquement.

Définition 2.1.1 *Une* formule principale *d'une inférence qui n'est pas une contraction, est une formule de sa conclusion qui n'existait pas dans ses prémisses. Dans le cas d'une contraction, il s'agit de la formule résultant de la contraction des deux formules du prémisse.*
La partie principale *d'une inférence I, notée $\Delta_p(I)$, est le multi-ensemble de ses formules principales.*

Exemple 2.1.1

inférence I	*partie principale $\Delta_p(I)$*
$\dfrac{}{\vdash \top,\ \Delta}\ \top$	$\{\top\} \cup \Delta$
$\dfrac{\vdash F,\ \Delta_1 \quad \vdash F^{\perp},\ \Delta_2}{\vdash \Delta_1,\ \Delta_2}\ cut$	$\emptyset$
$\dfrac{\vdash F,\ G,\ \Delta}{\vdash F \wp G,\ \Delta}\ \wp$	$\{F \wp G\}$

Définition 2.1.2 *Une* formule active *d'une inférence qui n'est pas une contraction, est une formule d'un de ses prémisses qui n'existe plus dans sa conclusion. Dans le cas d'une contraction, il y a deux formules actives, celles qui sont contractées.*
La partie active *du $k^{ième}$ prémisse d'une inférence I, notée $\Delta_a^k(I)$, est le multi-ensemble de ses formules actives. On parlera aussi de* parties actives gauche et droite *au lieu de parties actives du premier et du second prémisses.*

Exemple 2.1.2

inférence I	*partie active $\Delta_a^1(I)$*	*partie active $\Delta_a^2(I)$*
$\dfrac{}{\vdash \top,\ \Delta}\ \top$		
$\dfrac{\vdash \Delta}{\vdash ?F,\ \Delta}\ w?$	$\emptyset$	
$\dfrac{\vdash F,\ G,\ \Delta}{\vdash F \wp G,\ \Delta}\ \wp$	$\{F, G\}$	
$\dfrac{\vdash F,\ \Delta_1 \quad \vdash G,\ \Delta_2}{\vdash F \otimes G,\ \Delta_1,\ \Delta_2}\ \otimes$	$\{F\}$	$\{G\}$

Définition 2.1.3 *Le* contexte *du $k^{ième}$ prémisse d'une inférence I, noté $\Delta_c^k(I)$, est le complémentaire de sa partie active. On parlera aussi de* contextes gauche et droite *au lieu de contextes du premier et du second prémisses. On dira aussi qu'ils constituent les* contextes initiaux *de l'inférence.*
Le contexte final *d'une inférence I, noté $\Delta_c(I)$, est le complémentaire de sa partie principale.*

Exemple 2.1.3

inférence I	*contexte initial $\Delta_c^1(I)$*	*contexte initial $\Delta_c^2(I)$*	*contexte final $\Delta_c(I)$*
$\dfrac{}{\vdash \top,\ \Delta}\ \top$			$\emptyset$
$\dfrac{\vdash \Delta}{\vdash ?F,\ \Delta}\ w?$	Δ		Δ
$\dfrac{\vdash F,\ \Delta \quad \vdash G,\ \Delta}{\vdash F\&G,\ \Delta}\ \&$	Δ	Δ	Δ
$\dfrac{\vdash F,\ \Delta_1 \quad \vdash G,\ \Delta_2}{\vdash F\otimes G,\ \Delta_1,\ \Delta_2}\ \otimes$	Δ_1	Δ_2	$\Delta_1 \cup \Delta_2$

Les choses étant fixées au niveau local, nous allons pouvoir maintenant élargir notre champ aux preuves dans leur globalité pour définir le vocabulaire qui va nous permettre d'y décrire l'histoire des formules de leur apparition jusqu'à leur disparition par composition.

1.2 Evolution d'une formule dans une déduction marquée

Commençons par rappeler les définitions classiques de *déduction* et de *preuve* dans le calcul des séquents.

Définition 2.1.4 *Une* déduction *de CLL est un arbre binaire étiqueté par des séquents de CLL et tel que tout nœud a pour étiquette la conclusion d'une inférence dont les prémisses sont les étiquettes de ses fils.*
Une conclusion intermédiaire *d'une déduction est l'étiquette d'un de ses nœuds.*
Une hypothèse *est une conclusion intermédiaire étiquetant une feuille d'une déduction.*
La conclusion (finale) *d'une déduction est la conclusion intermédiaire attachée à sa racine.*
Une preuve *est une déduction sans hypothèses.*

Voyons maintenant les problèmes qui se posent lorsque l'on cherche à suivre l'évolution d'une formule dans une preuve. Partons d'un exemple. Considérons la preuve suivante :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A,\ A^\perp}\ id}{\vdash A,\ ?A^\perp}\ ?}{\vdash A,\ ?A^\perp,\ ?A^\perp}\ w?}{\vdash A,\ ?A^\perp \oplus F,\ ?A^\perp}\ \oplus_1$$

Si l'on cherche à établir l'histoire de la sous-formule $?A^\perp$ de $?A^\perp \oplus F$, on se trouve devant une ambiguïté : elle peut avoir été introduite soit par l'inférence ?, soit par l'inférence w?. La solution consiste à marquer les formules dans la preuve. Pour notre exemple, il y a deux marquages possibles d'où les deux preuves marquées suivantes issues de la preuve initiale.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A,\ A_1^\perp}\ id}{\vdash A,\ ?A_1^\perp}\ ?}{\vdash A,\ ?A_1^\perp,\ ?A_2^\perp}\ w?}{\vdash A,\ ?A_1^\perp \oplus F,\ ?A_2^\perp}\ \oplus_1 \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A,\ A_1^\perp}\ id}{\vdash A,\ ?A_1^\perp}\ ?}{\vdash A,\ ?A_1^\perp,\ ?A_2^\perp}\ w?}{\vdash A,\ ?A_2^\perp \oplus F,\ ?A_1^\perp}\ \oplus_1$$

Définition 2.1.5 *Une* inférence marquée *est une inférence munie d'un marquage qui permet d'identifier toute formule d'un de ses prémisses avec une formule (sous-formule si l'on considère une formule active) de sa conclusion.*
Une dérivation marquée *est une dérivation où chaque inférence est marquée.*

La notion de marquage va donner un sens précis à l'évolution d'une formule au sein d'une déduction, les étapes clés étant sa naissance et sa disparition en tant que telle.

Définition 2.1.6 *Dans une déduction marquée, on dit qu'une formule F d'une conclusion intermédiaire* a été introduite *par une inférence I si et seulement si F est une formule principale de I.*

Définition 2.1.7 *Dans une déduction marquée, on dit qu'une formule F d'une conclusion intermédiaire* a été activée *dans une inférence I si et seulement si F est une formule active de I.*

Exemple 2.1.4 *Considérons la preuve suivante :*

$$\dfrac{\dfrac{\dfrac{}{\vdash A, A^\perp}\,id}{\vdash A \oplus B, A^\perp}\,\oplus_1 \qquad \dfrac{\dfrac{}{\vdash B, B^\perp}\,id}{\vdash A \oplus B, B^\perp}\,\oplus_2}{\vdash A \oplus B, A^\perp \& B^\perp}\,\&$$

Pour cette preuve, il n'y a qu'un marquage possible qui va de soi. L'intérêt d'une telle preuve est qu'elle illustre le fait qu'une formule peut être introduite par plusieurs inférences à cause de la règle & qui entraîne une superposition de contextes. Par contre, il est clair qu'une formule peut être activée dans au plus une inférence. Ainsi, ici, $A \oplus B$ est introduite par les deux inférences $\oplus_1$ et $\oplus_2$ et $A^\perp$ est activée dans l'inférence &.

2 Permutabilité d'inférences

Le vocabulaire ainsi fixé, nous allons pouvoir maintenant définir de façon rigoureuse la propriété de permutabilité de deux inférences au sein d'une déduction [Kleene, 1952]. Cette question a déjà été abordée dans certains travaux sur la construction de preuves en logique linéaire [Bellin, 1991; Lincoln, 1992; Lincoln and Shankar, 1994] mais nous en faisons ici une étude exhaustive.

2.1 Définitions

A première vue, la notion semble simple : il s'agit de pouvoir permuter deux inférences consécutives dans une déduction sans perturber le reste de celle-ci. L'exemple suivant illustre tout à fait cette idée. Soit la déduction $\mathcal{D}_1$:

$$\dfrac{\dfrac{\mathcal{D}'_1\left\{\begin{array}{c}\vdots\\ \vdash F_1, G_1, \Delta_1\end{array}\right. \qquad \mathcal{D}''_1\left\{\begin{array}{c}\vdots\\ \vdash G_2, \Delta_2\end{array}\right.}{\vdash F_1, G_1 \otimes G_2, \Delta_1, \Delta_2}\,\otimes}{\vdash F_1 \oplus F_2, G_1 \otimes G_2, \Delta_1, \Delta_2}\,\oplus_1$$

En permutant les inférences $\otimes$ et $\oplus_1$, on obtient une nouvelle déduction $\mathcal{D}_2$ qui reste correcte.

$$\dfrac{\dfrac{\mathcal{D}'_1\left\{\begin{array}{c}\vdots\\ \vdash F_1, G_1, \Delta_1\end{array}\right.}{\vdash F_1 \oplus F_2, G_1, \Delta_1}\,\oplus_1 \qquad \mathcal{D}''_1\left\{\begin{array}{c}\vdots\\ \vdash G_2, \Delta_2\end{array}\right.}{\vdash F_1 \oplus F_2, G_1 \otimes G_2, \Delta_1, \Delta_2}\,\otimes$$

Dans la déduction $\mathcal{D}_1$, l'*objet de la permutation* est la déduction constituée de l'inférence $\otimes$ suivie de $\oplus_1$. Elle a pour hypothèses $\vdash F_1, G_1, \Delta_1$ et $\vdash G_2, \Delta_2$ et comme conclusion $\vdash F_1 \oplus F_2, G_1 \otimes G_2, \Delta_1, \Delta_2$. Le *résultat de la permutation* dans $\mathcal{D}_2$, est la déduction constituée de l'inférence $\oplus_1$ suivie de $\otimes$ qui a mêmes hypothèses et conclusion. Ainsi, la permutation n'a entraîné aucune perturbation au-dessus et au-dessous de son objet.
Considérons maintenant une déduction $\mathcal{D}_3$ où on a remplacé l'inférence $\otimes$ par une inférence &.

$$\dfrac{\dfrac{\mathcal{D}'_3\left\{\begin{array}{c}\vdots\\ \vdash F_1, G_1, \Delta\end{array}\right. \qquad \mathcal{D}''_3\left\{\begin{array}{c}\vdots\\ \vdash F_1, G_2, \Delta\end{array}\right.}{\vdash F_1, G_1 \& G_2, \Delta}\ \&}{\vdash F_1 \oplus F_2, G_1 \& G_2, \Delta}\ \oplus_1$$

Si l'on permute maintenant l'inférence & avec l'inférence $\oplus_1$, on obtient une déduction $\mathcal{D}_4$:

$$\dfrac{\dfrac{\mathcal{D}'_3\left\{\begin{array}{c}\vdots\\ \vdash F_1, G_1, \Delta\end{array}\right.}{\vdash F_1 \oplus F_2, G_1, \Delta}\ \oplus_1 \qquad \dfrac{\mathcal{D}''_3\left\{\begin{array}{c}\vdots\\ \vdash F_1, G_2, \Delta\end{array}\right.}{\vdash F_1 \oplus F_2, G_2, \Delta}\ \oplus_1}{\vdash F_1 \oplus F_2, G_1 \& G_2, \Delta}\ \&$$

qui est correcte mais on remarque que, contrairement au premier exemple, l'inférence $\oplus_1$ en permutant avec &, a été dupliquée. Cela montre la différence de comportement par rapport aux contextes entre les deux conjonctions de la logique linéaire.
Dans l'autre sens, la permutation avec une inférence de type & provoque une superposition de deux inférences. Elle est donc plus difficile à réaliser puisqu'*il faut avoir deux inférences identiques qui précède celle de type &*. Ainsi dans la déduction $\mathcal{D}_5$ ci-dessous, l'inférence $\oplus_1$ n'est pas permutable avec & car l'autre inférence précédant immédiatement & n'est pas de type $\oplus_1$; elle est de type $\oplus_2$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A, A^\perp}\ id \qquad \dfrac{}{\vdash B, B^\perp}\ id}{\vdash A \otimes B, A^\perp, B^\perp}\ \otimes}{\vdash (A \otimes B) \oplus (C \otimes B), A^\perp, B^\perp}\ \oplus_1 \qquad \dfrac{\dfrac{\dfrac{}{\vdash C, C^\perp}\ id \qquad \dfrac{}{\vdash B, B^\perp}\ id}{\vdash C \otimes B, C^\perp, B^\perp}\ \otimes}{\vdash (A \otimes B) \oplus (C \otimes B), C^\perp, B^\perp}\ \oplus_2}{\vdash (A \otimes B) \oplus (C \otimes B), A^\perp \& C^\perp, B^\perp}\ \&$$

Remarquons qu'il existe des cas où l'on peut tout de suite affirmer que l'ordre de deux inférences consécutives ne peut pas être changé : lorsqu'une formule principale de la première est active dans la seconde.

Définition 2.2.1 *Dans une déduction de CLL, une inférence qui n'est pas un axiome, est* en position de permutation *avec une autre qui la suit immédiatement, si et seulement si aucune formule principale de la première n'est active dans la seconde.*

Ainsi dans le dernier exemple évoqué, l'inférence $\otimes$ de gauche n'est pas en position de permutation avec $\oplus_1$ qui suit car la formule $A \otimes B$ est principale dans la première et active dans la seconde. Par contre, l'inférence $\oplus_1$ est en position de permutation avec & mais elle n'est pas permutable pour autant.
Ces explications préalables permettent de comprendre pourquoi on ne peut éviter de définir précisément la notion de permutabilité d'inférences.

Définition 2.2.2 *Soit dans une déduction $\mathcal{D}$, deux inférences I_1 et I_2 en position de permutation.*
I_1 *est* permutable avec I_2 *si et seulement s'il existe deux inférences I'_1 et I'_2 telles que :*

(i) les types de I_1 et I_2 sont respectivement ceux de I'_1 et I'_2 ;

(ii) la conclusion de I'_2 coïncide avec un prémisse de I'_1 ;

(iii) si I_1 est de type &, alors il existe une inférence J'_2 de même type que I_2 et dont la conclusion coïncide avec le second prémisse de I'_1 ;

(iv) si I_2 est de type &, alors l'autre inférence J_1 située juste avant dans $\mathcal{D}$, a le même type que I_1 ;

(v) la déduction formée par I_1 (et J_1 dans le cas où I_2 est de type &) suivie de I_2, considérée comme

l'objet de la permutation, *et la déduction formée par I'_2 (et J'_2 dans le cas où I_1 est de type &) suivie de I'_1, considérée comme le* résultat de la permutation, *ont même conclusion et même ensemble d'hypothèses (modulo pour ces dernières, un renommage des variables libres).*

Pour illustrer cette définition, reprenons les exemples précédents. Ainsi dans la déduction $\mathcal{D}_3$, les inférences I_1 et I_2 sont respectivement les inférences de type & et $\oplus_1$; elles constituent l'objet de la permutation. Les inférences I'_2, J'_2 et I'_1 sont respectivement les inférences gauche et droite de type $\oplus_1$ et celle de type & dans la déduction $\mathcal{D}_4$; elles constituent le résultat de la permutation. Il est facile de vérifier les 5 conditions qui permettent de conclure que I_1 est permutable avec I_2.
Dans la déduction $\mathcal{D}_5$, les inférences I_1, J_1 et I_2 sont respectivement les inférences de type $\oplus_1$, $\oplus_2$ et & ; or, elles ne sont pas permutables car la condition (iv) de la définition n'est pas vérifiée : I_1 et J_1 n'ont pas même type.
A partir de cette notion, on en déduit directement la notion de transformation d'une déduction par permutation d'inférences.

Définition 2.2.3 *Une déduction $\mathcal{D}'$ est* obtenue par permutation d'une inférence I_1 avec une inférence I_2 dans une déduction *$\mathcal{D}$ si et seulement si $\mathcal{D}'$ est obtenue en remplaçant dans $\mathcal{D}$ l'objet de la permutation de I_1 avec I_2 par le résultat de cette même permutation (modulo un renommage des variables libres et une duplication de branches de $\mathcal{D}$ au-dessus de l'objet de la permutation).*

2.2 Etude de la permutabilité

Nous pouvons maintenant mener une étude systématique de cette propriété dans CLL. Si l'on remarque que les types $\oplus_1$ et $\oplus_2$ se comporte de ce point de vue de la même façon, on peut les réunir en un seul type afin de diminuer le nombre de cas à traiter. Les conclusions d'une telle étude peuvent être résumées par le théorème suivant.

Théorème 2.2.1 *(de permutabilité) Soit t_1 et t_2 deux types du tableau ci-dessous.*

1. *la case (t_1, t_2) du tableau est vide si et seulement si toute inférence de type t_1 en position de permutation avec une inférence de type t_2 dans une déduction, est permutable avec celle-ci ;*
2. *la case (t_1, t_2) du tableau contient np ou $-$ si et seulement si il existe une inférence de type t_1 en position de permutation avec une inférence de type t_2 dans une déduction mais qui ne soit pas permutable avec cette dernière (le tiret $-$ indique que la non permutabilité est toute relative et qu'elle peut être contournée comme nous le verrons dans la prochaine section) ;*
3. *la case (t_1, t_2) du tableau contient une croix $\times$ si et seulement si il n'existe pas de déduction où une inférence de type t_1 soit en position de permutation avec une inférence de type t_2.*

$t_2 \backslash t_1$	$\otimes$	$\wp$	$\perp$	&	$\oplus$	?	$w?$	$c?$	!	$\forall$	$\exists$	*cut*
$\otimes$									np			
$\wp$	np								np			np
$\perp$									np			
&	np	$-$	$-$	$-$	np	np	np	$-$	np	$-$	np	np
$\oplus$									np			
?												
$w?$												
$c?$	np											np
!	$\times$	$\times$	$\times$	$\times$	$\times$	np			$\times$	$\times$	$\times$	np
$\forall$									np		np	np
$\exists$									np			
cut									np			

Preuve 2.2.1 *Elle est donnée en annexe A* □

3 Mouvements d'inférences

L'étude de la *permutabilité d'inférences* nous permet de connaître précisément le degré de mobilité des divers types d'inférences dans une preuve. Le but est d'ordonner au maximum les inférences au sein d'une preuve. Ceci se fera alors par des mouvements vers le haut et vers le bas qui ne sont en fait que des itérations de permutations élémentaires.
Ceci nous amène à définir cette notion de *mouvement d'une inférence dans une déduction.*

3.1 Montée d'une inférence dans une déduction

On va considérer ici un premier mouvement, la montée d'une inférence.

3.1.1 Définition

Intuitivement, cela consiste à itérer vers le haut l'opération de permutation de deux inférences dans une déduction. Le mouvement peut être compliqué si l'inférence qui est montée dans la déduction, doit pour cela permuter avec d'autres de type &: elle est alors dupliquée autant de fois qu'elle franchit ce genre d'obstacles. Le mouvement, unique au départ, peut ainsi par la suite se subdiviser en plusieurs mouvements parallèles. La nécessité de prendre en compte la complexité de ce phénomène nous amène à la définition suivante.

Définition 2.3.1 *Soit une déduction $\mathcal{D}$ de CLL et une inférence I de $\mathcal{D}$.*
Une déduction $\mathcal{D}'$ de CLL est obtenue par montée de I dans $\mathcal{D}$ *si et seulement si il existe une suite finie $\mathcal{D}_0, \cdots, \mathcal{D}_n$ de déductions de CLL et une suite finie $Inf_0, \cdots, Inf_n$ d'ensembles d'inférences telles que :*

1. *$\mathcal{D}_0 \equiv \mathcal{D}$, $\mathcal{D}_n \equiv \mathcal{D}'$ et $Inf_0 \equiv \{I\}$;*
2. *pour tout entier i tel que $0 \leq i < n$, $\mathcal{D}_{i+1}$ est obtenue à partir de $\mathcal{D}_i$ par permutation d'une inférence I_{1_i} avec une inférence I_{2_i} de Inf_i et l'ensemble Inf_{i+1} est obtenu en remplaçant dans Inf_i l'inférence I_{2_i} par l'inférence (ou les inférences si I_{1_i} est de type &) correspondante de $\mathcal{D}_{i+1}$.*

Les éléments de Inf_n sont appelés les inférences résultant de la montée de I dans $\mathcal{D}$.

Illustrons cette définition par un exemple.

Exemple 2.3.1 *Considérons la preuve $\mathcal{P}_0$ de CLL*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A,\ A^\perp}\,id}{\vdash A\oplus B,\ A^\perp}\,\oplus_1 \quad \dfrac{\dfrac{}{\vdash B,\ B^\perp}\,id}{\vdash A\oplus B,\ B^\perp}\,\oplus_2}{\vdash A\oplus B,\ A^\perp\&B^\perp}\,\& \quad \dfrac{}{\vdash C,\ C^\perp}\,id}{\vdash (A\oplus B)\otimes C,\ A^\perp\&B^\perp,\ C^\perp}\,\otimes}{\vdash (A\oplus B)\otimes C,\ A^\perp\&B^\perp,\ C^\perp,\ ?C}\,w?$$

Nous allons chercher à monter au maximum l'inférence w? dans la preuve. Donc, pour reprendre la notation de la définition, l'ensemble Inf_0 se réduit à { w?}.
Par une première permutation, nous obtenons la preuve $\mathcal{P}_1$ suivante :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A,\ A^\perp}\,id}{\vdash A\oplus B,\ A^\perp}\,\oplus_1 \quad \dfrac{\dfrac{}{\vdash B,\ B^\perp}\,id}{\vdash A\oplus B,\ B^\perp}\,\oplus_2}{\vdash A\oplus B,\ A^\perp\&B^\perp}\,\&}{\vdash A\oplus B,\ A^\perp\&B^\perp,\ ?C}\,w? \quad \dfrac{}{\vdash C,\ C^\perp}\,id}{\vdash (A\oplus B)\otimes C,\ A^\perp\&B^\perp,\ C^\perp,\ ?C}\,\otimes$$

Inf_1 se réduit toujours à $\{ w? \}$. Une nouvelle permutation avec cette fois une inférence & entraîne une duplication de l'inférence $w?$ et nous obtenons la preuve $\mathcal{P}_2$ suivante :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A,\ A^\perp}\,id}{\vdash A \oplus B,\ A^\perp}\,\oplus_1}{\vdash A \oplus B,\ A^\perp,\ ?C}\,w? \qquad \dfrac{\dfrac{\dfrac{}{\vdash B,\ B^\perp}\,id}{\vdash A \oplus B,\ B^\perp}\,\oplus_2}{\vdash A \oplus B,\ A^\perp,\ ?C}\,w?}{\vdash A \oplus B,\ A^\perp \& B^\perp,\ ?C}\,\& \qquad \dfrac{}{\vdash C,\ C^\perp}\,id}{\vdash (A \oplus B) \otimes C,\ A^\perp \& B^\perp,\ C^\perp,\ ?C}\,\otimes$$

Inf_2 est maintenant constitué de deux inférences $w?$. Par deux dernières permutations, celles-ci peuvent encore être montées juste au-dessous des axiomes. Elles constituent alors le résultat de la montée qui s'achève ici. La preuve obtenue se présente alors ainsi :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A,\ A^\perp}\,id}{\vdash A,\ A^\perp,\ ?C}\,w?}{\vdash A \oplus B,\ A^\perp,\ ?C}\,\oplus_1 \qquad \dfrac{\dfrac{\dfrac{}{\vdash B,\ B^\perp}\,id}{\vdash B,\ B^\perp,\ ?C}\,w?}{\vdash A \oplus B,\ A^\perp,\ ?C}\,\oplus_2}{\vdash A \oplus B,\ A^\perp \& B^\perp,\ ?C}\,\& \qquad \dfrac{}{\vdash C,\ C^\perp}\,id}{\vdash (A \oplus B) \otimes C,\ A^\perp \& B^\perp,\ C^\perp,\ ?C}\,\otimes$$

3.1.2 Quelles inférences monter et jusqu'où ?

L'observation du tableau accompagnant le théorème 2.2.1, permet de répondre en partie à la première. Les types d'inférences faciles à monter sont ceux correspondant à des lignes du tableau contenant un minimum de symboles *np*.

Ainsi les inférences de type ? et $w?$ ne peuvent pas être bloquées. Et celles de type cut, $\otimes$, $\oplus$, $\exists$, $\perp$ peuvent être bloquées uniquement par des inférences de type !.

Jusqu'où monter une inférence ? Les limites peuvent être de trois sortes :

1. on se trouve au sommet de la déduction (dans le cas d'une déduction avec hypothèses) ;
2. l'inférence qui précède celle qui est montée, n'est plus en position de permutation avec elle ; c'est le cas lorsque l'on parvient à monter l'inférence juste au-dessous d'une autre introduisant une de ses formules actives ou lorsqu'on réussit à la monter juste après un axiome ;
3. l'inférence qui précède celle qui est montée, est toujours en position de permutation avec elle mais n'est plus permutable.

Ces limites dépendent bien entendu du type de l'inférence qui est montée, ce qui va donner lieu à plusieurs théorèmes que nous allons maintenant établir. Afin d'éviter le premier des trois cas qui viennent d'être mentionnés, nous nous restreindrons aux preuves, étant donné que par la suite, nous ne travaillerons que rarement avec des déductions comportant des hypothèses.

3.1.3 Les résultats

a) Montée de w?

Commençons par les affaiblissements (inférences de type $w?$). D'après le théorème 2.2.1, il n'y a aucun obstacle à leur montée et comme ils ne comportent aucune formule active, ils peuvent être montés jusqu'aux axiomes dans les preuves. D'où le théorème suivant :

Théorème 2.3.1 *Si $\mathcal{P}$ est une preuve de CLL et si I est une inférence de type w? de $\mathcal{P}$, alors il existe une preuve $\mathcal{P}'$ de CLL obtenue par une montée de I dans $\mathcal{P}$ et telle que toutes les inférences résultant de cette montée sont immédiatement précédées par un axiome.*

Preuve 2.3.1 *Elle s'effectue par induction sur la profondeur de $\mathcal{P}$.*
Le cas de base est trivial.
Le cas d'induction demande à être développé uniquement lorsque I est la dernière inférence de $\mathcal{P}$. Alors d'après le théorème 2.2.1, on peut permuter l'avant-dernière inférence de $\mathcal{P}$ avec I. On obtient une preuve dont la profondeur n'a pas été augmentée. On peut donc appliquer l'hypothèse d'induction aux preuves partielles extraites de celle-ci et se terminant par l'inférence I qui vient d'être remontée d'un niveau (il y en a au maximum deux). On obtient ainsi la preuve cherchée. □

L'exemple 2.3.1 est une application du théorème 2.3.1.

b) Montée de ?

Pour ce qui est des inférences de type ?, la seule différence avec les affaiblissements, est qu'elles comportent une formule active donc elles vont être stoppées dans leur montée par les inférences introduisant cette formule active. D'où le théorème :

Théorème 2.3.2 *Si $\mathcal{P}$ est une preuve de CLL et si I est une inférence de type ? de $\mathcal{P}$, alors il existe une preuve $\mathcal{P}'$ de CLL obtenue par une montée de I dans $\mathcal{P}$ et telle que toutes les inférences résultant de cette montée sont immédiatement précédées par une inférence introduisant leur formule active.*

Preuve 2.3.2 *Elle s'effectue de la même façon que celle du théorème 2.3.1.* □

c) Montée du cut

Apparemment, selon le théorème 2.2.1, la montée d'une coupure (inférence de type *cut*) dans une preuve peut être stoppée par une inférence de type !. Mais comme l'illustre l'exemple suivant, cet obstacle peut être levé et cela donne lieu à un théorème analogue au théorème 2.3.2.

Exemple 2.3.2 *Nous allons chercher à monter au maximum la coupure dans la preuve ci-dessous.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash B,\ B^\perp}\,id}{\vdash B,\ ?B^\perp}\,?}{\vdash B,\ ?B^\perp,\ ?A^\perp}\,w?}{\vdash !B,\ ?B^\perp,\ ?A^\perp}\,! \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A,\ A^\perp}\,id}{\vdash A,\ ?A^\perp}\,?}{\vdash !A,\ ?A^\perp}\,! \qquad \dfrac{}{\vdash B,\ B^\perp}\,id}{\vdash !A,\ (?A^\perp)\otimes B,\ B^\perp}\,\otimes}{\vdash !B,\ (?A^\perp)\otimes B,\ ?B^\perp,\ B^\perp}\,cut$$

Nous ne pouvons pas permuter la coupure avec l'inférence ! mais par contre c'est possible avec l'inférence $\otimes$. Nous obtenons alors la preuve suivante :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash B,\ B^\perp}\,id}{\vdash B,\ ?B^\perp}\,?}{\vdash B,\ ?B^\perp,\ ?A^\perp}\,w?}{\vdash\ !B,\ ?B^\perp,\ ?A^\perp}\,! \qquad \dfrac{\dfrac{\dfrac{}{\vdash A,\ A^\perp}\,id}{\vdash A,\ ?A^\perp}\,?}{\vdash\ !A,\ ?A^\perp}\,!}{\vdash\ !B,\ ?B^\perp,\ ?A^\perp}\,cut \qquad \dfrac{}{\vdash B,\ B^\perp}\,id}{\vdash\ !B,\ ?B^\perp,\ (?A^\perp)\otimes B,\ B^\perp}\,\otimes$$

Maintenant, la permutation de la coupure avec l'inférence ! est possible et nous obtenons la preuve :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash B,\ B^\perp}\,id}{\vdash B,\ ?B^\perp}\,?}{\vdash B,\ ?B^\perp,\ ?A^\perp}\,w? \qquad \dfrac{\dfrac{\dfrac{}{\vdash A,\ A^\perp}\,id}{\vdash A,\ ?A^\perp}\,?}{\vdash\ !A,\ ?A^\perp}\,!}{\vdash B,\ ?B^\perp,?A^\perp}\,cut}{\vdash\ !B,\ ?B^\perp,\ ?A^\perp}\,! \qquad \dfrac{}{\vdash B,\ B^\perp}\,id}{\vdash\ !B,\ ?B^\perp,\ (?A^\perp)\otimes B,\ B^\perp}\,\otimes$$

La montée s'arrête ici car les inférences précédant la coupure ne sont plus en position de permutation avec elle (dans la prochaine section, nous verrons toutefois comment on peut aller plus loin pour faire disparaître carrément la coupure).

Théorème 2.3.3 *Si $\mathcal{P}$ est une preuve de CLL et si I est une inférence de type cut de $\mathcal{P}$, alors il existe une preuve $\mathcal{P}'$ de CLL obtenue par une montée de I dans $\mathcal{P}$ et telle que toutes les inférences résultant de cette montée sont immédiatement précédées par les inférences introduisant leurs formules actives.*

Preuve 2.3.3 *Elle s'effectue par induction sur le nombre d'inférences précédant I dans $\mathcal{P}$.*
- Le cas de base est trivial.
- Considérons maintenant le cas d'induction et l'on doit distinguer deux éventualités selon le type des inférences précédant I.

1. Une inférence précédant immédiatement I n'est pas de type ! et n'introduit pas une formule active de I. *Il suffit alors de permuter celle-ci avec I en appliquant le théorème 2.2.1 et d'appliquer ensuite l'hypothèse d'induction.*

2. Les inférences précédant immédiatement I sont de type ! ou introduisent une formule active de I. *Si elles introduisent toutes les deux les formules actives de I, on a la preuve cherchée. Sinon, avec I, elles forment la configuration suivante :*

$$\dfrac{\dfrac{\vdash ?F,\ G,\ ?\Delta_n}{\vdash ?F,\ !G,\ ?\Delta_n}\,! \qquad \dfrac{\vdash\ F^\perp,\ ?\Delta'_n}{\vdash\ !F^\perp,\ ?\Delta'_n}\,!}{\vdash\ !G,\ ?\Delta_n,\ ?\Delta'_n}\,cut$$

On peut donc permuter dans la preuve $\mathcal{P}$ l'inférence gauche juste avant I avec I. On obtient à la place de la configuration précédente, celle-ci :

$$\dfrac{\dfrac{\vdash ?F,\ G,\ ?\Delta_n \qquad \dfrac{\dfrac{\vdash F^{\perp},\ ?\Delta'_n}{\vdash !F^{\perp},\ ?\Delta'_n}\,!}{\vdash !F^{\perp},\ ?\Delta'_n}\,!}{\vdash G,\ ?\Delta_n,\ ?\Delta'_n}\,cut}{\vdash !G,\ ?\Delta_n,\ ?\Delta'_n}\,!$$

On peut ensuite appliquer l'hypothèse d'induction à la nouvelle preuve puisque le nombre d'inférences précédant I a été diminué de un.

□

d) Montée de $\otimes$, $\oplus$ et $\exists$

Pour ce qui des inférences de type $\otimes$, $\oplus$ et $\exists$, les obstacles potentiels à leur montée que peuvent présenter les inférences de type ! ne peuvent pas être levés de la même façon comme le montre l'exemple suivant.

Exemple 2.3.3 *Nous allons chercher à monter l'inférence $\otimes$ dans la preuve suivante :*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A,\ A^{\perp}}\,id}{\vdash A\wp A^{\perp}}\,\wp}{\vdash A\wp A^{\perp},\ ?B}\,w? \qquad \dfrac{\dfrac{\dfrac{}{\vdash C,\ C^{\perp}}\,id}{\vdash C,\ ?C^{\perp}}\,?}{\vdash !C,\ ?C^{\perp}}\,!}{\vdash (A\wp A^{\perp})\otimes ?C^{\perp},\ !C,\ ?B}\,\otimes$$

Par permutation de l'inférence $w?$ avec $\otimes$, nous obtenons :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A,\ A^{\perp}}\,id}{\vdash A\wp A^{\perp}}\,\wp \qquad \dfrac{\dfrac{\dfrac{}{\vdash C,\ C^{\perp}}\,id}{\vdash C,\ ?C^{\perp}}\,?}{\vdash !C,\ ?C^{\perp}}\,!}{\vdash (A\wp A^{\perp})\otimes ?C^{\perp},\ !C}\,\otimes}{\vdash (A\wp A^{\perp})\otimes ?C^{\perp},\ !C,\ ?B}\,w?$$

La permutation de l'inférence ! avec $\otimes$ n'est pas possible donc le mouvement s'arrête là.

Pour la montée des inférences de type $\otimes$, $\oplus$ et $\exists$, on a donc le théorème suivant :

Théorème 2.3.4 *Si $\mathcal{P}$ est une preuve de CLL et si I est une inférence de $\mathcal{P}$ de type $\otimes$, $\oplus$ ou $\exists$, alors il existe une preuve $\mathcal{P}'$ de CLL obtenue par une montée de I dans $\mathcal{P}$ et telle que toutes les inférences résultant de cette montée sont immédiatement précédées par une inférence de type ! ou introduisant une de leurs formules actives.*

Preuve 2.3.4 *Elle s'effectue de la même façon que celle du théorème 2.3.3 par induction sur le nombre d'inférences précédant I dans $\mathcal{P}$.* □

e) Montée de c?

Si l'on observe le tableau du théorème 2.2.1, on constate que les contractions (inférences de type c?), lorsqu'elles sont montées dans une preuve, peuvent être bloquées par une inférence de type *cut* ou $\otimes$. Toutefois, ce mouvement , même s'il est difficile, peut être intéressant comme nous le verrons dans la prochaine section. Il donne alors lieu au théorème suivant :

Théorème 2.3.5 *Si $\mathcal{P}$ est une preuve de CLL et si I est une inférence de $\mathcal{P}$ de type c?, alors il existe une preuve $\mathcal{P}'$ de CLL obtenue par une montée de I dans $\mathcal{P}$ et telle que toutes les inférences I' résultant de cette montée sont immédiatement précédées par une inférence introduisant une de leurs formules actives ou une inférence de type cut ou $\otimes$; dans ce dernier cas, les deux formules actives de I' sont chacune dans un des prémisses de l'inférence immédiatement avant.*

Preuve 2.3.5 *Elle peut s'effectuer de la même façon que celle du théorème 2.3.3 par induction sur le nombre d'inférences précédant I dans $\mathcal{P}$ ou comme pour le théorème 2.3.1 par induction sur la structure de $\mathcal{P}$.* □

f) Montée de $\wp$

Les inférences de type $\wp$, du fait qu'elles ont un seul prémisse avec deux formules actives, se comportent comme les contractions. La seule différence est qu'elles peuvent être bloquées par des inférences de type !. D'où le théorème :

Théorème 2.3.6 *Si $\mathcal{P}$ est une preuve de CLL et si I est une inférence de $\mathcal{P}$ de type $\wp$, alors il existe une preuve $\mathcal{P}'$ de CLL obtenue par une montée de I dans $\mathcal{P}$ et telle que toutes les inférences I' résultant de cette montée sont immédiatement précédées par une inférence introduisant une de leurs formules actives, une inférence de type ! ou enfin une inférence de type cut ou $\otimes$; dans ce dernier cas, les deux formules actives de I' sont chacune dans un des prémisses de l'inférence immédiatement avant.*

Preuve 2.3.6 *Elle peut s'effectuer de la même façon que le théorème précédant.* □

g) Montée de !

D'après le tableau du théorème 2.2.1, il paraît difficile de monter les inférences de type ! dans les preuves par permutations successives étant donné la forme nécessaire que doit avoir le contexte. En revanche, on peut envisager une *montée par bonds* d'une conclusion intermédiaire de la forme $\vdash F, ?\Delta$ à la suivante de la même forme. Mais si l'on rencontre une inférence de type &, les choses se compliquent comme le montre la preuve suivante :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A,\ A^{\perp}}\,\mathrm{id}}{\vdash A,\ ?A^{\perp}}\,? \qquad \dfrac{}{\vdash A,\ A^{\perp}}\,\mathrm{id}}{\vdash A^{\perp}\&A^{\perp},\ A}\,\&}{\dfrac{\vdash ?(A^{\perp}\&A^{\perp}),\ A}{\vdash ?(A^{\perp}\&A^{\perp}),\ !A}\,!}\,?$$

Il n'est pas possible de monter l'inférence ! juste après la conclusion intermédiaire $\vdash A, ?A^{\perp}$ car il faudrait disposer d'une autre de même forme dans la branche droite au-dessus de l'inférence &. Une autre difficulté

peut surgir si la formule active de l'inférence ! à remonter est de la forme $?F$. Considérons la preuve :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A,\ A^{\perp}}\,\mathrm{id}}{\vdash A,\ ?A^{\perp}}\,?}{\vdash ?A,\ ?A^{\perp}}\,?}{\vdash ?A,\ !?A^{\perp}}\,!}{\vdash ?A,\ ?!?A^{\perp}}\,?}{\vdash !?A,\ ?!?A^{\perp}}\,?$$

Il n'est pas possible de monter la dernière inférence ! juste après la deuxième inférence ? car cela invaliderait la première inférence !.

3.2 Descente d'une inférence dans une déduction

3.2.1 Définition

Considérons maintenant le mouvement symétrique du précédent. Intuitivement, il provient d'une itération vers le bas de la permutation de deux inférences. Comme pour la montée, ce mouvement peut être compliqué par la rencontre d'une inférence & mais l'effet est contraire : l'inférence en train d'être descendue, fusionne alors avec une autre provenant de l'autre branche de la déduction au-dessus de l'inférence &. Donc, partant d'un ensemble d'inférences, le mouvement se termine avec une seule. C'est pourquoi, la descente est définie comme le contraire de la montée.

Définition 2.3.2 *Soit une déduction $\mathcal{D}$ de CLL et une inférence I de $\mathcal{D}$.*
Une déduction $\mathcal{D}'$ de CLL est obtenue par descente de I dans $\mathcal{D}$ *si et seulement si il existe une déduction $\mathcal{D}'$ de CLL et une inférence I' de $\mathcal{D}'$ telles que $\mathcal{D}$ est obtenue par montée de I' dans $\mathcal{D}'$, I étant une des inférences résultant de cette montée.*

Il est évident que I' est unique et on l'appellera *l'inférence résultant du mouvement de descente de I dans $\mathcal{D}$*. Illustrons cette définition par un exemple.

Exemple 2.3.4 *Nous allons chercher à descendre l'inférence $\forall$ dans la preuve suivante :*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash a(x),\ a(x)^{\perp}}\,id}{\vdash \exists y\ a(y),\ a(x)^{\perp}}\,\exists}{\vdash \exists y\ a(y),\ \forall z\ a(z)^{\perp}}\,\forall \qquad \dfrac{}{\vdash B,\ B^{\perp}}\,id}{\vdash (\exists y\ a(y)) \otimes B,\ \forall z\ a(z)^{\perp},\ B^{\perp}}\,\otimes}{\vdash (\exists y\ a(y)) \otimes B,\ \forall z\ a(z)^{\perp},\ (B^{\perp} \oplus c(x))}\,\oplus_1}{\vdash (\exists y\ a(y)) \otimes B,\ (\forall z\ a(z)^{\perp}) \wp (B^{\perp} \oplus c(x))}\,\wp$$

Une première permutation avec l'inférence $\otimes$ nous permet d'obtenir la preuve :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash\ a(x),\ a(x)^{\perp}}\,id}{\vdash\ \exists y\ a(y),\ a(x)^{\perp}}\,\exists \qquad \dfrac{}{\vdash\ B,\ B^{\perp}}\,id}{\vdash\ (\exists y\ a(y)) \otimes B,\ a(x)^{\perp},\ B^{\perp}}\,\otimes}{\vdash\ (\exists y\ a(y)) \otimes B,\ \forall z\ a(z)^{\perp},\ B^{\perp}}\,\forall}{\vdash\ (\exists y\ a(y)) \otimes B,\ \forall z\ a(z)^{\perp},\ (B^{\perp} \oplus c(x))}\,\oplus_1}{\vdash\ (\exists y\ a(y)) \otimes B,\ (\forall z\ a(z)^{\perp}) \wp (B^{\perp} \oplus c(x))}\,\wp$$

Nous continuons par une permutation avec l'inférence $\oplus_1$ qui nécessite un renommage préalable de la variable x dans la preuve au-dessus de la conclusion de $\forall$ pour éviter une violation de la condition associée à la règle $\forall$. Nous obtenons alors la preuve :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash\ a(u),\ a(u)^{\perp}}\,id}{\vdash\ \exists y\ a(y),\ a(u)^{\perp}}\,\exists \qquad \dfrac{}{\vdash\ B,\ B^{\perp}}\,id}{\vdash\ (\exists y\ a(y)) \otimes B,\ a(u)^{\perp},\ B^{\perp}}\,\otimes}{\vdash\ (\exists y\ a(y)) \otimes B,\ a(u)^{\perp},\ (B^{\perp} \oplus c(x))}\,\oplus_1}{\vdash\ (\exists y\ a(y)) \otimes B,\ \forall z\ a(z)^{\perp},\ (B^{\perp} \oplus c(x))}\,\forall}{\vdash\ (\exists y\ a(y)) \otimes B,\ (\forall z\ a(z)^{\perp}) \wp (B^{\perp} \oplus c(x))}\,\wp$$

La descente s'arrête là car l'inférence $\forall$ n'est plus en position de permutation avec celle qui suit.

3.2.2 Quelles inférences descendre et jusqu'où ?

Pour ce qui est de quelles inférences descendre, reportons-nous une fois de plus au tableau du théorème 2.2.1. Un type d'inférence est d'autant plus facile à descendre dans une déduction que la colonne correspondante du tableau contient un minimum de symboles *np*.

Ainsi, les inférences de type $\wp$, &, c?, $\forall$ et $\perp$ peuvent être descendues aisément dans les déductions .

Il faut signaler toutefois que pour les contractions, cela demande d'étendre un peu la définition de la permutabilité de deux inférences.
Pour ce qui est des limites à ce mouvement de descente, on retrouve les trois cas symétriques de ceux rencontrés pour la montée :

1. l'inférence peut être descendue jusqu'en bas de la déduction ;

2. l'inférence qui est descendue, ne se trouve plus à un moment donné en position de permutation avec celle qui la suit immédiatement ; cela signifie que sa formule principale devient active ;

3. l'inférence qui est descendue, est toujours en situation de permutation avec celle qui la suit immédiatement mais elle n'est plus permutable avec elle.

a) Descente $\wp$, &, $\forall$ et $\perp$

Le comportement des divers types d'inférences dans le mouvement de descente est beaucoup plus homogène que dans celui de montée d'où une simplification des théorèmes.

Théorème 2.3.7 *Soit $\mathcal{D}$ une déduction ne comportant pas de formule qui ait comme connecteur de tête $\wp$, &, $\forall$ ou $\perp$ et qui soit introduite par un axiome de type $\top$.*
Si I est une inférence de $\mathcal{D}$ de type $\wp$, &, $c?$, $\forall$ ou $\perp$, alors il existe une déduction $\mathcal{D}'$ obtenue par une descente de I dans $\mathcal{D}$ telle que l'inférence résultant de cette descente est soit la dernière de $\mathcal{D}'$, soit elle est suivie immédiatement par une autre où sa formule principale est active.

Preuve 2.3.7 *Elle s'effectue par induction sur la structure de $\mathcal{D}$.*
Le cas de base est trivial.
Le cas d'induction nous amène à envisager deux éventualités selon que la formule principale F de I est active dans $\mathcal{D}$ ou pas.

1. F est active dans $\mathcal{D}$.
   On applique l'hypothèse d'induction à la déduction partielle extraite de $\mathcal{D}$ qui a comme conclusion un prémisse de sa dernière inférence et qui contient I. On obtient alors la déduction cherchée.

2. F n'est pas active dans $\mathcal{D}$.
   Soit I' la dernière inférence de $\mathcal{D}$. Selon le type de I', nous sommes amenés à distinguer deux cas :

   (a) `I' n'est pas de type &.`
   On applique l'hypothèse d'induction à la déduction partielle extraite de $\mathcal{D}$ qui a comme conclusion un prémisse de I' et qui contient I. On applique ensuite le théorème 2.2.1 qui permet d'amener I en bas de la déduction par une dernière permutation.

   (b) `I' est de type &.`
   Selon le type de I, on doit encore distinguer deux cas :

   i. *I n'est pas une contraction*
   Dans la branche de $\mathcal{D}$ ne contenant pas I, soit I'' l'inférence introduisant F. Elle est nécessairement du même type que I car, étant donné la restriction effectuée sur $\mathcal{D}$, elle ne peut pas être de type $\top$.
   Appliquons l'hypothèse d'induction aux deux déductions partielles extraites de $\mathcal{D}$ qui se terminent par les prémisses de I'; cela permet de descendre I et I'' en bas de ces deux déductions. Elles sont alors identiques donc on peut les permuter avec I' et on obtient la déduction cherchée.

   ii. *I est une contraction*
   Appliquons l'hypothèse d'induction à la déduction partielle extraite de $\mathcal{D}$ qui se termine par un prémisse de I' et qui contient I; cela permet de descendre I qui forme alors avec I' la configuration suivante :

$$\dfrac{\dfrac{\Pi_1\left\{\begin{array}{c}\vdots\\ \vdash ?F', ?F', G, \Delta\end{array}\right.}{\vdash ?F', G, \Delta}\,c? \qquad \left.\begin{array}{c}\vdots\\ \vdash ?F', H, \Delta\end{array}\right\}\Pi_2}{\vdash ?F',\ G\&H,\ \Delta}\,\&$$

   Dans cette configuration, ?F' est la formule principale F de I. En introduisant un affaiblissement suivi d'une contraction à la fin de $\mathcal{P}_2$, on peut alors permuter I avec I' (bien sûr, on commet ainsi une légère entorse à notre définition de la permutabilité mais qui se

justifie) et on obtient alors la configuration :

$$\dfrac{\dfrac{\Pi_1\left\{\begin{array}{c}\vdots\\ \vdash ?F', ?F', G, \Delta\end{array}\right. \quad \dfrac{\left.\begin{array}{c}\vdots\\ \vdash ?F', H, \Delta\end{array}\right\}\Pi_2}{\vdash ?F',\ ?F',\ H,\ \Delta}\,w?}{\vdash ?F',\ ?F',\ G\&H,\ \Delta}\,\&}{\vdash ?F',\ G\&H,\ \Delta}\,c?$$

On a donc ainsi la déduction souhaitée.

□

Remarque 2.3.1 *La restriction dans les hypothèses du théorème précédent relative aux axiomes de type* $\top$*, n'est pas gênante car il est toujours possible de la contourner par l'insertion d'inférences supplémentaires. Par exemple ci-dessous, l'axiome à gauche peut toujours être remplacé par la configuration à droite.*

$$\dfrac{}{\vdash F\&G, \Delta}\,\top \qquad\qquad \dfrac{\dfrac{}{\vdash F, \Delta}\,\top \quad \dfrac{}{\vdash G, \Delta}\,\top}{\vdash F\&G, \Delta}\,\&$$

Remarque 2.3.2 *Ce théorème explique aussi la distinction effectuée dans le tableau du théorème 2.2.1, entre les cases marquées "np" et celles marquées "-". Ces dernières indiquent que les inférences* I_1 *et* I_2 *correspondantes ne sont pas immédiatement permutables mais qu'elles le deviennent après descente juste au-dessus de* I_2 *d'une troisième inférence dans la branche parallèle à* I_1*.*

b) Descente de w?

Même si le mouvement de descente des affaiblissements peut être bloqué par une inférence de type &, il est intéressant de l'envisager pour une utilisation ultérieure. D'où le théorème suivant :

Théorème 2.3.8 *Si* $\mathcal{D}$ *est une déduction et* I *une inférence de* $\mathcal{D}$ *de type* $w?$*, alors il existe une déduction* $\mathcal{D}'$ *obtenue par une descente de* I *dans* $\mathcal{D}$ *telle que l'inférence résultant de cette descente est soit la dernière de* $\mathcal{D}'$*, soit elle est suivie immédiatement par une autre où sa formule principale est active ou bien qui est de type &.*

Preuve 2.3.8 *Par induction sur le nombre n d'inférences qui sépare I de celle où sa formule principale est active ou bien du bas de la déduction* $\mathcal{D}$ *si cette formule principale n'est pas active dans* $\mathcal{D}$*.*
Si n= 0, le résultat est immédiat sinon on permute I avec l'inférence suivante en utilisant le théorème 2.2.1. Puis on applique l'hypothèse de récurrence à la déduction ainsi obtenue.□

c) Descente de !

Pour ce qui est des inférences !, elle peuvent d'une façon semblable à ce qui se passe en sens inverse, être descendues par bonds mais elles ne rencontrent pas les mêmes obstacles, ce que traduit le théorème suivant :

Théorème 2.3.9 *S'il existe une déduction* $\mathcal{D}$ *de CLL de conclusion* $\vdash !F, ?\Delta$*, alors on obtient une nouvelle déduction à partir de* $\mathcal{D}$ *en supprimant l'inférence introduisant* $!F$*, en remplaçant dans toutes les conclusions intermédiaires qui suivent,* $!F$ *par* F *et en ajoutant à la fin de la déduction une nouvelle inférence de type ! introduisant* $!F$*.*

Preuve 2.3.9 *Elle est triviale.* □

On remarque qu'on a pu classer les types d'inférences en deux catégories : ceux dont les inférences sont faciles à monter et ceux dont les inférences sont faciles à descendre. Ce classement est à rapprocher de celui effectué par [Andreoli, 1992] sur les connecteurs logiques : les connecteurs *asynchrones* qui peuvent être éliminés à n'importe quel moment qui suit leur apparition lors de la construction des preuves de bas en haut, correspondent aux types d'inférences faciles à descendre ; les connecteurs *synchrones* qui ne peuvent être éliminés qu'à des moments précis, correspondent eux aux types d'inférences faciles à monter. Mais la correspondance n'est pas totale. Ainsi, nous nous situons au niveau de types d'inférences ce qui nous amène à différencier les types ?, w? et c? qui ne se situent pas tous dans la même catégorie. Andreoli, lui, se situe au niveau des connecteurs logiques et il considère "?" comme asynchrone . Par ailleurs, il situe "!" dans les connecteurs synchrones alors que nous rangeons le type ! dans ceux qui sont aisés à descendre. Par la suite, nous reviendrons sur cette classification : dans la prochaine section, nous verrons qu'elle est relative et qu'elle dépend du sens de construction des preuves choisi ; le chapitre 5 nous permettra de mieux comprendre le vocabulaire utilisé par Andreoli et le lien entre sa classification et la nôtre.

4 Vers des preuves normales

Dans la précédente section, nous avons envisagé les mouvements d'inférences dans une preuve isolément. Il s'agit maintenant de les combiner pour transformer une preuve en une preuve normale, c'est-à-dire une preuve où les inférences sont rangées suivant un certain ordre.

4.1 Facteurs de normalisation

La normalisation d'une preuve est la combinaison de la montée de certaines inférences avec la descente d'autres. La première question qui se pose alors, est : quelles inférences monter et quelles inférences descendre ? Deux facteurs influent essentiellement sur ce choix : le *fragment de LL* dans lequel se situe la preuve et le *sens de construction des preuves* par rapport auquel on se place, de haut en bas ou de bas en haut. Nous allons développer maintenant ces deux aspects.

4.1.1 Influence du fragment logique

Dans la précédente section, nous avons vu qu'un premier critère pour choisir quelles inférences monter et quelles inférences descendre, est un *critère de facilité*. Dans CLL, nous avons même pu ranger les types d'inférences en deux classes $T\uparrow$ et $T\downarrow$ correspondant respectivement aux inférences faciles à monter et à celles faciles à descendre. Nous avons alors la partition suivante :

$T\uparrow = \{cut,\ \otimes,\ \oplus,\ ?,\ w?,\ \exists\}$;
$T\downarrow = \{\wp,\ \&,\ c?,\ !,\ \forall,\ \bot\}$.

Mais il est aisé de voir qu'un tel résultat est relatif au fragment de LL dans lequel on se place. Ainsi, plus le fragment est restreint, plus la mobilité des inférences dans les preuves est accrue et plus la normalisation des preuves peut y être poussée loin.
Considérons par exemple le fragment de CLL d'où sont bannis les connecteurs & et ! qui sont les principaux freins à la mobilité des inférences dans une preuve si l'on observe le tableau accompagnant le théorème 2.2.1. L'application du théorème 2.2.1 permet de redéfinir les deux classes $T\uparrow$ et $T\downarrow$ ainsi :

$T\uparrow = \{cut,\ \otimes,\ \oplus,\ ?,\ w?,\ \exists,\ \bot\}$;
$T\downarrow = \{\wp,\ \&,\ \oplus,\ ?,\ w?,\ c?,\ !,\ \forall,\ \bot\}$.

Ces deux classes ne sont plus disjointes ce qui nous amènera à utiliser d'autres critères pour déterminer le sens du mouvement de certaines inférences dont le type appartient aux deux.

4.1.2 Influence du sens de construction des preuves

Dans cette section, nous envisageons des preuves déjà construites que nous transformons pour les normaliser. Mais il ne s'agit pas d'un objectif en soi. Ce qui nous intéresse, c'est la construction de preuves que ce soit pour la rendre plus efficace en vue de sa mécanisation ou lui donner un sens dans une application telle qu'un calcul de processus.
Or, cette construction peut se faire de bas en haut en partant de la conclusion finale pour remonter

jusqu'aux axiomes ou le contraire, de haut en bas. Dans le premier sens, les règles d'inférence sont appliquées à partir d'une conclusion connue qui est remplacée par des prémisses correspondants. Dans l'autre sens, ce sont des prémisses connus qui sont remplacés par une conclusion correspondante. Dans un cas comme dans l'autre, il n'est pas toujours facile de déterminer quelle règle utiliser.

a) Traitement de w?

Considérons la preuve $\mathcal{P}_1$ de CLL suivante :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A,\ A^\perp}\,id}{\vdash A,\ A^\perp,\ ?C}\,w? \qquad \dfrac{}{\vdash B,\ B^\perp}\,id}{\vdash A\otimes B,\ A^\perp,\ B^\perp,\ ?C}\,\otimes}{\vdash (A\otimes B)\wp A^\perp,\ B^\perp,\ ?C}\,\wp}{\vdash (A\otimes B)\wp A^\perp\wp B^\perp,\ ?C}\,\wp}{\vdash (A\otimes B)\wp A^\perp\wp B^\perp\wp ?C}\,\wp$$

Imaginons que nous cherchions à construire $\mathcal{P}_1$ de haut en bas. A partir de la conclusion intermédiaire $\vdash A,\ A^\perp$, il est impossible de prévoir qu'il faut ensuite effectuer l'inférence $w?$ étant donné que sa partie active est vide. Nous avons donc là une source importante d'indéterminisme.
Essayons maintenant de descendre au maximum l'inférence $w?$ dans la preuve $\mathcal{P}_1$. Nous obtenons la preuve $\mathcal{P}_2$ suivante :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A,\ A^\perp}\,id \qquad \dfrac{}{\vdash B,\ B^\perp}\,id}{\vdash A\otimes B,\ A^\perp,\ B^\perp}\,\otimes}{\vdash (A\otimes B)\wp A^\perp,\ B^\perp}\,\wp}{\vdash (A\otimes B)\wp A^\perp\wp B^\perp}\,\wp}{\vdash (A\otimes B)\wp A^\perp\wp B^\perp,\ ?C}\,w?}{\vdash (A\otimes B)\wp A^\perp\wp B^\perp\wp ?C}\,\wp$$

L'inférence $w?$ se situe maintenant juste avant l'inférence $\wp$ qui utilise sa formule principale $?C$ comme formule active. Or, dans une construction de haut en bas de la preuve, c'est particulièrement intéressant. A partir de la conclusion intermédiaire $\vdash (A\otimes B)\wp A^\perp\wp B^\perp$, il est maintenant possible de prévoir qu'il faut effectuer ensuite une inférence $w?$. En effet, nous cherchons à construire une *preuve sans coupures* donc on peut utiliser la *propriété de la sous-formule* qui nous dit ici qu'il reste à composer la formule $(A\otimes B)\wp A^\perp\wp B^\perp$ avec $?C$ pour obtenir la conclusion finale. La solution est alors vite trouvée.

> *Ainsi, dans une perspective de construction des preuves de haut en bas, il faut descendre au maximum les affaiblissements dans les preuves.*

Maintenant, si l'on se place dans une optique de construction ascendante des preuves, il vaut mieux chercher à construire la preuve $\mathcal{P}_1$ que $\mathcal{P}_2$ car la présence de la formule $?C$ ne permet pas de dire si elle a été introduite par une inférence de type $?$, $w?$ ou $c?$. Remonter les affaiblissements au niveau des axiomes, comme le permet le théorème 2.3.1, permet de limiter le choix : avant d'atteindre un axiome, on n'utilisera jamais la règle d'affaiblissement.

> *Donc, dans une perspective de construction des preuves de bas en haut, il faut monter au maximum les affaiblissements dans les preuves, c'est-à-dire jusqu'aux axiomes.*

En généralisant, on peut dire que dans la perspective d'une construction des preuves de bas en haut, on a intérêt à descendre les inférences qui sont faciles à déterminer à partir de leur conclusion et à remonter celles qui peuvent difficilement être prévues à partir de celle-ci ; il s'agit de repousser la difficulté jusqu'à ce qu'on ait l'information suffisante pour la traiter. Dans une perspective de construction des preuves de haut en bas, c'est le contraire : il faut remonter les inférences qu'on peut facilement prévoir à partir de leurs prémisses et descendre celles pour lesquelles c'est difficile.

b) Traitement de c?

Mis à part les affaiblissements, ce principe s'applique aussi aux contractions. La règle c? se présente ainsi :

$$\frac{\vdash ?F, ?F, \Delta}{\vdash ?F, \Delta}\ c?$$

La présence de deux formules $?F$ dans le prémisse montre qu'une contraction est plus facile à prévoir depuis son prémisse que depuis sa conclusion.

Donc, quel que soit le sens de construction des preuves envisagé, il faut monter au maximum les contractions dans les preuves.

c) Traitement de $\perp$

La règle $\perp$ est la troisième qui est sensible au sens de construction des preuves. Rappelons sa définition :

$$\frac{\vdash \Delta}{\vdash \perp, \Delta}\ \perp$$

La présence de la constante $\perp$ dans la conclusion permet de prévoir sans ambiguïté l'application de la règle $\perp$ dans une construction des preuves de bas en haut. Par contre, dans le sens contraire, l'absence de formule active nous laisse avec autant d'indéterminisme que la règle $w?$.

En résumé, quel que soit le sens de construction des preuves envisagé, il faut descendre au maximum les inférences de type $\perp$ dans les preuves.

4.1.3 Problème de la cohérence des mouvements d'inférences au sein d'une preuve

Quand on effectue plusieurs mouvements d'inférences au sein d'une même preuve, il faut s'assurer qu'ils ne se contredisent pas mutuellement. Supposons, pour simplifier le problème que les inférences déplacées puissent toujours permuter avec celles avec lesquelles elles sont en position de permutation.
Rappelons ce que nous avons dit dans la section précédente. Dans le mouvement vers le haut, si une inférence a des formules actives, elle peut alors être montée juste au-dessous de celles produisant ces formules actives, sinon elle peut être montée jusqu'aux axiomes. Dans le mouvement vers le bas, si une inférence a sa formule principale active dans la preuve, alors elle peut être descendue immédiatement au-dessus de celle où cette formule est active, sinon elle peut être descendue en bas de la preuve.
La cohérence de l'ensemble des mouvements peut être mise à mal de quatre façons :

1. **dans leur montée, plusieurs inférences peuvent être bloquées par le même axiome de type id ou 1.**
   Considérons par exemple la preuve :

$$\dfrac{\dfrac{\dfrac{}{\vdash A,\ A^{\perp}}\,id \qquad \dfrac{}{\vdash B,\ B^{\perp}}\,id}{\vdash A \otimes B,\ A^{\perp},\ B^{\perp}}\,\otimes \qquad \dfrac{}{\vdash C,\ C^{\perp}}\,id}{\vdash A \otimes B,\ A^{\perp} \otimes C,\ B^{\perp},\ C^{\perp}}\,\otimes$$

   Dans cette preuve, si l'on cherche à monter au maximum les deux inférences $\otimes$, elles ne pourront pas toutes les deux venir se placer juste après celles produisant leurs formules actives étant donné que le même axiome produit une formule active de l'une, A, et une formule active de l'autre, $A^{\perp}$. Une façon de choisir consiste par exemple à décider de placer juste après l'axiome celle qui utilise comme

formule active, l'atome positif A. Dans d'autres cas, il n'est pas possible de régler le problème ainsi ; on tolère alors un certain indéterminisme qui n'est pas gênant puisqu'il s'agit d'indéterminisme "don't care" : la preuve reste correcte quel que soit l'ordre des inférences concernées.

2. **dans leur montée, plusieurs inférences peuvent être bloquées par le même axiome de type $\top$.**
   Il est facile de simplifier la preuve en supprimant parmi les inférences bloquées, toutes celles qui n'ont pas la formule $\top$ comme formule active. Il ne restera alors plus éventuellement qu'une seule inférence où la constante $\top$ est active. Considérons la preuve :

$$\dfrac{\dfrac{\dfrac{}{\vdash \top,\ A}\,\top \qquad \dfrac{}{\vdash B,\ B^{\perp}}\,id}{\vdash \top,\ A\otimes B,\ B^{\perp}}\,\otimes \qquad \dfrac{}{\vdash C,\ C^{\perp}}\,id}{\vdash \top\otimes C,\ A\otimes B,\ B^{\perp},\ C^{\perp}}\,\otimes$$

Si l'on veut monter toutes les deux inférences $\otimes$ juste après celles qui introduisent leurs formules actives, il y a conflit car elles ont toutes deux une formule active introduite par l'axiome $\top$. La solution consiste à supprimer celle des deux inférences qui est inutile ; ici, il s'agit de celle introduisant $A \otimes B$. La preuve simplifiée se présente alors ainsi :

$$\dfrac{\dfrac{}{\vdash \top,\ A\otimes B,\ B^{\perp}}\,\top \qquad \dfrac{}{\vdash C,\ C^{\perp}}\,id}{\vdash \top\otimes C,\ A\otimes B,\ B^{\perp},\ C^{\perp}}\,\otimes$$

On peut généraliser cette simplification à l'aide du théorème suivant :

Théorème 2.4.1 *Tout séquent prouvable de CLL possède une preuve vérifiant la propriété :*

> *Pour tout axiome de type $\top$, toutes ses formules principales sauf une égale à la constante $\top$, si elles sont actives dans la preuve, le sont dans une inférence qui ne peut pas être montée jusqu'à cet axiome.*

Preuve 2.4.1 *Dans cette démonstration, nous dirons que dans une preuve de CLL, un axiome de type $\top$ possède la propriété $\mathcal{SIMPLIF}$ si toutes ses formules principales sauf une égale à la constante $\top$, si elles sont actives dans la preuve, le sont dans une inférence qui ne peut pas être montée jusqu'à cet axiome.*
Considérons une preuve quelconque $\mathcal{P}$ de CLL. Nous allons montrer par induction sur la structure de $\mathcal{P}$ qu'il existe une preuve qui a même conclusion que $\mathcal{P}$ et dont tous les axiomes $\top$ vérifient la propriété $\mathcal{SIMPLIF}$.
Soit I la dernière inférence de $\mathcal{P}$. Par hypothèse d'induction, chaque prémisse de I (s'il en existe) possède une preuve dont tous les axiomes $\top$ vérifient la propriété $\mathcal{SIMPLIF}$.
En prolongeant ces preuves par l'inférence I, on obtient une nouvelle preuve $\mathcal{P}'$ qui a même conclusion que $\mathcal{P}$. Mais malheureusement, l'inférence I peut entraîner des violations de $\mathcal{SIMPLIF}$ dans le cas où elle possède au moins une formule active F introduite par un axiome de type $\top$ qui a comme autre formule principale la constante $\top$.
Montons alors I au maximum dans la preuve $\mathcal{P}'$. Nous obtenons une preuve $\mathcal{P}''$. Pour toute inférence I' résultant de cette montée et violant la propriété $\mathcal{SIMPLIF}$, nous allons montrer qu'il est possible de modifier localement $\mathcal{P}''$ pour supprimer cette anomalie. Pour cela, nous sommes amenés à faire une analyse de cas selon le type de I'.

(a) I' est de type $\otimes$.
Elle forme alors avec les inférences qui la précèdent, une preuve partielle extraite de $\mathcal{P}''$ qui se

présente ainsi :

$$\dfrac{\dfrac{}{\vdash\ \top,\ F_1,\ \Delta_1}\,\top \qquad \begin{array}{c}\vdots\\ \vdash\ F_2,\ \Delta_2\end{array}}{\vdash\ F_1 \otimes F_2,\ \top,\ \Delta_1,\ \Delta_2}\,\otimes$$

On peut remplacer cette sous-preuve par le seul axiome :

$$\dfrac{}{\vdash\ F_1 \otimes F_2,\ \top,\ \Delta_1,\ \Delta_2}\,\top$$

Etant donné la manière dont $\mathcal{P}''$ a été obtenue, cet axiome vérifie la propriété $\mathcal{SIMPLIF}$ donc l'anomalie a été supprimée.

(b) l est de type &.
Elle forme alors avec les inférences qui la précèdent, une preuve partielle extraite de $\mathcal{P}''$ qui se présente ainsi :

$$\dfrac{\dfrac{}{\vdash\ \top,\ F_1,\ \Delta}\,\top \qquad \begin{array}{c}\vdots\\ \vdash\ \top,\ F_2,\ \Delta\end{array}}{\vdash\ \top,\ F_1 \& F_2,\ \Delta}\,\&$$

On peut remplacer cette sous-preuve par le seul axiome :

$$\dfrac{}{\vdash\ F_1 \& F_2,\ \top,\ \Delta}\,\top$$

Etant donné la manière dont $\mathcal{P}''$ a été obtenue, cet axiome vérifie la propriété $\mathcal{SIMPLIF}$ donc l'anomalie a été supprimée.

(c) l a un seul prémisse.
Elle constitue avec l'axiome $\top$ qui la précède, la configuration suivante :

$$\dfrac{\dfrac{}{\vdash\ \top,\ \Delta_a,\ \Delta}\,\top}{\vdash\ \top,\ \Delta_p,\ \Delta}$$

On peut facilement remplacer ces deux inférences par un seul axiome :

$$\dfrac{}{\vdash\ \top,\ \Delta_p,\ \Delta}\,\top$$

Comme dans les deux cas précédents, cet axiome vérifie la propriété $\mathcal{SIMPLIF}$.

□

Le théorème qui vient d'être démontré, prouve qu'il est toujours possible d'éviter que les axiomes de type $\top$ provoque des contradictions dans un mouvement de montée de plusieurs inférences.

3. **dans leur descente, deux inférences peuvent être bloquées par une même troisième de type $\wp$ ou $c?$.**
   Pourquoi ces deux types d'inférences seulement ? Parce que les inférences correspondantes possèdent un prémisse qui contient deux formules actives. Les deux formules actives peuvent avoir été produites par deux inférences différentes qui ne peuvent pas être descendues toutes les deux juste avant celle activant leur formule principale. Comme dans le cas précédent, soit on effectue un choix, soit on laisse subsister un indéterminisme.

4. **dans leur descente, plusieurs inférences peuvent arriver en bas de la preuve**
   Ce cas se pose si la conclusion de la preuve contient plus d'une formule. On le traite comme les autres.

4.2 L'élimination des coupures : une première phase de la normalisation

L'élimination des coupures peut être considérée comme une première phase dans la normalisation des preuves telle que nous l'entendons ici. En effet, elle repose essentiellement sur leur montée au maximum au sein des preuves. L'élimination des coupures dans les preuves est essentielle pour la démonstration automatique de théorèmes car elle permet d'y appliquer la *propriété de la sous-formule* [Gallier, 1986].
[Girard, 1987] a déjà démontré le théorème d'élimination des coupures dans LL, en prouvant que, quelle que soit la stratégie choisie pour remonter les coupures dans une preuve afin de les éliminer au niveau des axiomes, le calcul termine. Pour cela, il n'a pas travaillé directement avec des séquents mais avec des réseaux de preuves. Nous restons dans CLL et nous allons montrer qu'il existe une stratégie particulière qui termine.
Notre démonstration part du théorème 2.3.3 qui permet de monter une coupure dans une preuve juste au-dessous des deux inférences produisant ses formules actives. Pour franchir cette barrière, nous utilisons alors le fait que les deux formules actives sont la négation l'une de l'autre donc qu'elles sont produites par deux règles d'inférence duales ($\otimes$ et $\wp$ par exemple). Cette symétrie nous permet ici d'éliminer les deux inférences produisant les deux formules actives de la coupure qui était bloquée. On peut ensuite appliquer de nouveau le théorème 2.3.3 et ainsi de suite jusqu'à ce qu'on arrive aux axiomes.
Le problème est que l'on est pas sûr qu'un tel processus termine. A cause des contractions, l'étendue de la preuve peut croître au fur et à mesure du processus. Il est donc difficile de trouver un paramètre entier simple de la preuve qui diminue strictement avec la remontée des coupures : ni la profondeur de celle-ci, ni le nombre de ses inférences ne conviennent. [Lincoln, 1992] résout le problème en généralisant la règle de la coupure ainsi lorsqu'elle porte sur exponentiels :

$$\frac{\vdash ?F, \cdots, ?F, \Delta_1 \qquad \vdash !F^{\perp}, \Delta_2}{\vdash \Delta_1, \Delta_2} \; cut^*$$

Cette nouvelle règle évite de dupliquer une partie de la preuve pour résoudre le cas où une coupure est bloquée par une contraction produisant une de ses formules actives. D'une certaine façon, elle traduit une stratégie particulière de montée des coupures dans les preuves : elle lie entre eux les mouvements dans des branches parallèles issues de duplications.
Nous avons choisi de ne pas modifier le système d'inférence mais plutôt de définir une fonction entière d'une preuve qui décroisse strictement au cours du processus de montée des coupures lorsque l'on applique une stratégie particulière. Nous utilisons la notion de *complexité des coupures dans une preuve.*

4.2.1 Complexité des coupures dans une preuve

Commençons par définir la complexité d'une formule au sein d'une preuve.

Définition 2.4.1 *Soit $\mathcal{P}$ une preuve de CLL et F une formule présente dans une conclusion intermédiaire de $\mathcal{P}$; la* complexité de la formule F *est l'entier c(F) défini inductivement ainsi :*

1. *si F est introduite par un axiome ou une inférence de type w? ou $\perp$, alors c(F) = 1 ;*
2. *si F est introduite par une seule inférence qui a des formules actives, alors c(F) s'obtient en incrémentant de 1 la complexité de la formule active qui est la plus élevée ;*
3. *si F est introduite par plusieurs inférences, on choisit pour c(F) la complexité des occurrences de F introduites par une seule inférence qui est la plus élevée.*

Ceci nous amène ensuite à la définition de la complexité d'une coupure dans une preuve.

Définition 2.4.2 *La complexité d'une coupure I dans une preuve de CLL est l'entier c(I) qui est égal à la somme des complexités de ses formules actives.*

Nous en venons enfin à la définition de la *complexité des coupures dans une preuve.* Au premier abord, on pourrait penser qu'il suffit de choisir le maximum des complexités de ses coupures. Malheureusement, une telle fonction ne décroît pas nécessairement au cours du processus de montée des coupures. Nous

avons choisi une stratégie consistant à éliminer d'abord les coupures situées le plus haut possible dans la preuve. D'où la définition suivante :

Définition 2.4.3 *La complexité des coupures dans une preuve $\mathcal{P}$ est l'entier $c(\mathcal{P})$ qui est égal au maximum des complexités des coupures de $\mathcal{P}$ qui ne sont précédées par aucune autre dans $\mathcal{P}$. Elle est nulle si la preuve $\mathcal{P}$ ne comporte aucune coupure.*

4.2.2 L'élimination des coupures proprement dite

Le théorème principal consiste à montrer que la fonction que nous venons de définir décroît strictement au cours de la montée des coupures dans une preuve.

Théorème 2.4.2 *Soit $\mathcal{P}_1$ une preuve de CLL qui comporte au moins une coupure. Alors il existe une preuve $\mathcal{P}_2$ qui a la même conclusion que $\mathcal{P}_1$ et une complexité des coupures strictement plus faible.*

Preuve 2.4.2 *Voir en annexe B* □

Le théorème d'élimination des coupures proprement dit est un corollaire du précédent.

Théorème 2.4.3 *Tout séquent prouvable de CLL admet une preuve sans coupures.*

Preuve 2.4.3 *Soit une preuve $\mathcal{P}$ de CLL. Considérons la propriété P(n) suivante :*

> *"il existe une preuve $\mathcal{P}_n$ qui a même conclusion que $\mathcal{P}$ et dont la complexité des coupures est au moins inférieure de n à celle de $\mathcal{P}$."*

Montrons par récurrence que P(n) est vraie pour tout n tel que $0 \leq n \leq c(\mathcal{P})$.
P(0) est vraie car il suffit de prendre $\mathcal{P}$ pour $\mathcal{P}_0$.
Supposons maintenant la propriété vraie pour n quelconque tel que $0 \leq n < c(\mathcal{P})$. Montrons que P(n+1) est alors vraie. Par hypothèse de récurrence, il existe une preuve $\mathcal{P}_n$ qui a même conclusion que $\mathcal{P}$ et dont la complexité des coupures est inférieure de n à celle de $\mathcal{P}$. On peut alors appliquer à celle-ci le théorème 2.4.3 et la remplacer par une preuve de complexité des coupures strictement inférieure. La conclusion finale n'a pas changé. On a donc la preuve $\mathcal{P}_{n+1}$ cherchée. La propriété P(n+1) est vraie.
On vient donc de montrer que P(n) est vraie pour tout n tel que $0 \leq n \leq c(\mathcal{P})$. En particulier $P(c(\mathcal{P}))$ est vraie ce qui signifie qu'il existe une preuve $\mathcal{P}_{c(\mathcal{P})}$ qui a même conclusion que $\mathcal{P}$ et qui est sans coupures, ce qu'il fallait démontrer.□

Comme nous l'avons déjà souligné, la propriété de la sous-formule est liée à l'absence de coupures dans les preuves et joue un rôle essentiel dans la démonstration automatique de théorèmes. C'est pourquoi nous la rappelons ici.

Théorème 2.4.4 *Toute formule présente dans une conclusion intermédiaire d'une preuve sans coupures de CLL est une sous-formule d'une formule de la conclusion finale.*

Preuve 2.4.4 *Elle s'effectue facilement par induction en faisant une analyse de cas selon le type de la dernière inférence de la preuve.*□

Conclusion

Nous disposons de deux concepts de base : *la permutabilité de deux inférences* et *le mouvement d'une inférence* au sein d'une preuve. Avec ces deux outils, nous avons pu dresser un tableau complet de la permutabilité d'inférences et des mouvements possibles d'une inférence dans une preuve de CLL.
Nous avons défini le processus de normalisation d'une preuve comme la combinaison de mouvements d'inférences en son sein. Nous avons dégagé les deux facteurs qui déterminent ce processus : le fragment logique considéré et le sens de construction des preuves choisi. Nous avons également cerné les sources éventuelles d'incohérence dans le mouvement d'ensemble.
Une première mise en application de ces principes a consisté à éliminer les coupures dans les preuves.

Désormais, nous considérerons uniquement des preuves sans coupures de façon à pouvoir appliquer la propriété de la sous-formule dans leur construction.
Il va falloir maintenant aller plus loin dans l'application. Nous avons choisi de le faire dans le cadre de CLL tout entier où nous avons voulu montrer que le choix du sens de construction des preuves influe fortement sur la forme que prend leur normalisation : dans le chapitre suivant, le chapitre 3, nous traiterons de la normalisation des preuves dans CLL en vue de les construire en chaînage arrière (de bas en haut) et dans le chapitre 4, ce sera en vue de les construire en chaînage avant (de haut en bas).

Chapitre 3

Normalisation dans CLL et preuves en chaînage arrière

Introduction

La plupart des travaux sur la déduction en logique linéaire se situent dans le cadre d'une construction de preuves en chaînage arrière.
[Hodas and Miller, 1991, 1994] ont montré que, dans un fragment d'ILL, les preuves pouvaient être normalisées sous forme de *preuves uniformes*, c'est-à-dire où les règles gauches sont appliquées seulement lorsque les règles droites ne peuvent plus l'être. Ils ont mis en évidence la propriété de *"backchaining"* qui permet de pousser à bout la décomposition d'une formule de la partie gauche d'un séquent lorsqu'elle est entamée. [Harland and Pym, 1990, 1991, 1994] ont abouti à des résultats semblables dans un fragment voisin.
[Andreoli, 1992] a mis en évidence deux propriétés dans la construction de preuves dans CLL : la *réversibilité* qui permet de décomposer certaines formules dès qu'elles apparaissent dans la construction ascendante des preuves et le *"focusing"* qui est le "backchaining" appliqué aux formules de la partie droite des séquents.
[Lincoln and Shankar, 1994] se sont intéressés à la gestion des quantificateurs au premier ordre où, par une technique de *skolémisation dynamique*, ils ont rendu permutables les inférences de type $\forall$ et $\exists$.
Les concepts définis dans le chapitre précédent appliqués à la normalisation des preuves dans CLL pour les construire de bas en haut, vont-ils nous permettre d'éclairer et de retrouver ces différents résultats? Peuvent-ils permettre d'aller plus loin? C'est ce que nous nous proposons d'étudier dans ce chapitre.
Nous allons déplacer les inférences dans les preuves selon les deux critères qui ont été mis en évidence précédemment : *la facilité à le faire* et *leur meilleur contrôle en chaînage arrière* . Pour cela, nous procéderons par étapes :

- nous commencerons par monter au maximum les contractions ;
- après, nous ferons de même avec les affaiblissements ;
- ces deux premiers mouvements vont nous permettre d'intégrer contractions et affaiblissements aux autres règles d'inférence et ainsi de spécialiser le système d'inférence en un système, CLL$\uparrow$, adapté à la construction ascendante de preuves ;
- enfin, dans CLL$\uparrow$, nous déplacerons les inférences logiques pour obtenir les preuves normales de CLL$\uparrow$.

1 Traitement des contractions.

1.1 Montée des contractions

Comme nous l'avons déjà dit, nous devons les monter au maximum dans les preuves. D'après le théorème 2.3.5, dans ce mouvement, elles peuvent rencontrer deux types d'obstacles : des inférences de type $\otimes$ et des inférences introduisant leurs formules actives. Nous allons montrer qu'on peut se ramener à deux cas se traduisant par deux configurations particulières :

- Une configuration que nous qualifierons de *$\otimes$-blocage* qui se présente ainsi :

$$\dfrac{\dfrac{\vdash G_1,\ ?F_1, \cdots,\ ?F_n,\ \Delta_1 \quad \vdash G_2,\ ?F_1, \cdots,\ ?F_n,\ \Delta_2}{\vdash G_1 \otimes G_2,\ ?F_1, \cdots,\ ?F_n,\ ?F_1, \cdots,\ ?F_n,\ \Delta_1,\ \Delta_2}\otimes}{\vdash G_1 \otimes G_2,\ ?F_1, \cdots,\ ?F_n,\ ?F_2, \cdots,\ ?F_n,\ \Delta_1,\ \Delta_2}\mathrm{c}?$$

$$\dfrac{\vdots}{G_1 \otimes G_2,\ ?F_1, \cdots,\ ?F_n,\ \Delta_1,\ \Delta_2}\mathrm{c}?$$

  Les contractions présentes dans cette configuration, sont bloquées dans leur montée car leurs deux formules actives sont réparties entre les deux prémisses de l'inférence $\otimes$.

- Une configuration que nous qualifierons de *?-blocage* et qui a la forme suivante :

$$\dfrac{\dfrac{\vdash F,\ ?F,\ \Delta}{\vdash ?F,\ ?F,\ \Delta}?}{\vdash ?F,\ \Delta}\mathrm{c}?$$

  La contraction présente dans cette configuration, ne peut pas être déplacée vers le haut car elle est bloquée par l'inférence ? qui introduit une de ses formules actives.

Théorème 3.1.1 *A partir de toute preuve sans coupures de CLL, on peut obtenir par une suite de montées de contractions et de modifications du marquage des formules, une preuve où toute contraction se trouve dans une configuration de $\otimes$-blocage ou de ?-blocage.*

Preuve 3.1.1 *Dans cette démonstration, nous dirons qu'une contraction dans une preuve de CLL, possède la propriété $\mathcal{C}-\mathcal{NORMAL}$ si elle se trouve dans une configuration de $\otimes$-blocage ou de ?-blocage.*
Considérons une preuve quelconque $\mathcal{P}$ sans coupures de CLL. Nous allons montrer par récurrence sur la profondeur de $\mathcal{P}$ qu'il existe une preuve de CLL qui a la même conclusion que $\mathcal{P}$ et dont toutes les contractions possèdent la propriété $\mathcal{C}-\mathcal{NORMAL}$.
Le cas de base est trivial.
Le cas d'induction demande à être développé uniquement lorsque la dernière inférence I de $\mathcal{P}$ est une contraction. Dans ce cas, nous appliquons l'hypothèse de récurrence à la preuve extraite de $\mathcal{P}$ qui a comme conclusion le prémisse de I et nous la remplaçons par une preuve dont toutes les contractions ont la propriété $\mathcal{C}-\mathcal{NORMAL}$.
Ensuite, en utilisant le théorème 2.3.5, nous montons au maximum I dans la preuve. Nous obtenons une preuve $\mathcal{P}'$ où toute inférence I' résultant de la montée ne possède pas forcément la propriété $\mathcal{C}-\mathcal{NORMAL}$. Selon sa position, elle peut même entraîner d'autres contractions à ne plus suivre cette propriété. Voyons comment réparer ces éventuelles anomalies en distinguant plusieurs cas selon le type de l'inférence I'' qui bloque I'. Ces cas découlent de l'application du théorème 2.3.5.

1. I'' est de type $w?$.
   Comme la formule principale de I'' est active dans I', ces deux inférences se neutralisent et il suffit de les supprimer toutes les deux pour éliminer cette configuration qui n'est pas dans la norme.

2. I'' est de type c?.
Le blocage de I' est dû à la façon dont la preuve $\mathcal{P}'$ a été marquée. En modifiant ce marquage, on peut permettre à une contraction de poursuivre sa montée. Pour être précis, nous devons distinguer deux cas selon le type de configuration dans lequel se trouve I'' car il faut rappeler que I'' possédait la propriété $\mathcal{C}-\mathcal{NORMAL}$ avant la montée de I.

   (a) *I'' se trouve dans une suite ininterrompue de contractions bloquées par une inférence de type $\otimes$.*
   La suite de contractions où I' s'est insérée, avec l'inférence $\otimes$ qui la bloque, constitue une configuration qui se présente ainsi :

$$\dfrac{\vdash ?F,\ ?F,\ G_1,\ ?F_1,\cdots,\ ?F_n,\ \Delta_1 \quad \vdash ?F,\ G_1,\ ?F_1,\cdots,\ ?F_n,\ \Delta_2}{\vdash ?F,\ ?F,\ ?F,\ G_1\otimes G_2,\ ?F_1,\cdots,\ ?F_n,\ ?F_1,\cdots,\ ?F_n,\ \Delta_1,\ \Delta_2}\otimes$$

$$\vdots$$

$$\dfrac{\dfrac{\vdash ?F,\ ?F,\ ?F,\ G_1\otimes G_2,\ ?F_k,\cdots,\ ?F_n,\ ?F_1,\cdots,\ ?F_n,\ \Delta_1,\ \Delta_2}{\vdash ?F,\ ?F,\ G_1\otimes G_2,\ ?F_k,\cdots,\ ?F_n,\ ?F_1,\cdots,\ ?F_n,\ \Delta_1,\ \Delta_2}\ c?\ I''}{\vdash ?F,\ G_1\otimes G_2,\ ?F_k,\cdots,\ ?F_n,\ ?F_1,\cdots,\ ?F_n,\ \Delta_1,\ \Delta_2}\ c?\ I'$$

$$\vdots$$

$$\dfrac{}{\vdash ?F, G_1\otimes G_2,\ ?F_1,\cdots,\ ?F_n,\ \Delta_1,\ \Delta_2}\ c?$$

   En modifiant le marquage, on peut monter la première contraction I'' juste au-dessus de l'inférence $\otimes$ et appliquer ensuite l'hypothèse d'induction à la branche dans laquelle est cette contraction. On supprime ainsi toutes les anomalies relatives à la propriété $\mathcal{C}-\mathcal{NORMAL}$.

   (b) *I'' est précédée immédiatement par une inférence de type? qui introduit une de ses formules actives.*
   Ces deux inférences avec la contraction I' qui suit, forment donc une déduction qui se présente ainsi :

$$\dfrac{\dfrac{\dfrac{\vdash F,\ ?F,\ ?F,\ \Delta}{\vdash ?F,\ ?F,\ ?F,\ \Delta}\ ?}{\vdash ?F,\ ?F,\ \Delta}\ c?\ I''}{\vdash ?F,\ \Delta}\ c?\ I'$$

   Là aussi le blocage est dû la façon dont la preuve $\mathcal{P}'$ a été marquée. Comme dans le cas précédent, ce marquage peut être modifié pour permettre à la première contraction de monter ce qui fera disparaître l'anomalie relativement à la propriété $\mathcal{C}-\mathcal{NORMAL}$.

1.2 Simplification du système d'inférence.

Le théorème précédent va nous permettre de simplifier le système d'inférence de CLL et de retrouver un résultat déjà proposé par [Andreoli, 1992] : comme il est possible au sein d'une preuve de CLL, de localiser les contractions dans deux types de configurations qui ont une forme bien précise, nous allons fusionner les inférences constituant chaque configuration.

1. la règle $\otimes'$ correspondant au $\otimes$-blocage qui a la forme suivante :

$$\dfrac{\vdash F_1,\ \Delta_1,\ ?\Delta \quad \vdash F_2,\ \Delta_2,\ ?\Delta}{\vdash F_1\otimes F_2,\ \Delta_1,\ \Delta_2,\ ?\Delta}\otimes'$$

où les formules de Δ_1 et Δ_2 n'ont pas ? comme connecteur de tête

La condition attachée à cette règle pourrait être supprimée ; elle a comme seule rôle de faciliter la construction des preuves de bas en haut.

2. la règle ?' correspondant au ?-blocage qui est ainsi définie :

$$\frac{\vdash F,\ ?F,\ \Delta}{\vdash ?F,\ \Delta}\ ?'$$

Etant donné le théorème précédent, l'introduction des règles $\otimes'$ et $?'$ dans le système d'inférence de CLL, va permettre de supprimer la règle $c?$.
Ces règles peuvent même se substituer dans tous les cas de figure aux règles $\otimes$ et ?. Elle risquent, dans une construction de bas en haut des preuves, d'engendrer des formules parasites de la forme $?F$ qui, toutefois, pourront toujours être éliminées par des affaiblissements. Considérons par exemple l'inférence de CLL :

$$\frac{\vdash A,\ ?F \quad \vdash B}{\vdash A \otimes B,\ ?F}\ \otimes$$

Dans le système simplifié, elle est remplacée par la configuration :

$$\frac{\vdash A,\ ?F \quad \dfrac{\vdash B}{\vdash B,\ ?F}\ w?}{\vdash A \otimes B,\ ?F}\ \otimes'$$

2 Traitement des affaiblissements

2.1 Mouvement de montée

Ce mouvement se justifie par une perspective de construction ascendante des preuves. Le théorème 2.3.1 montre que tout inférence de type $w?$ prise isolément, peut être montée jusqu'aux axiomes. Mais lorsque l'on envisage un mouvement général, se pose un problème de cohérence (section 4.1 du chapitre 2) : deux affaiblissements qui cherchent à se placer juste après le même axiome, ne vont pas pouvoir le faire.
Le théorème suivant résout ce type de problème en utilisant une notion d'absorption d'un affaiblissement par un axiome $\top$ qu'il est nécessaire de définir au préalable.

Définition 3.2.1 *Une preuve $\mathcal{P}'$ de CLL est obtenue par absorption d'un affaiblissement par un axiome $\top$ à partir d'une preuve $\mathcal{P}$ de CLL si l'on a remplacé dans $\mathcal{P}$:*

une sous-preuve de la forme $\dfrac{\dfrac{}{\vdash \top,\ \Delta}\ \top}{\vdash \top,\ ?F,\ \Delta}\ w?$ *par l'axiome* $\dfrac{}{\vdash \top,\ ?F,\ \Delta}\ \top$

Théorème 3.2.1 *A partir de toute preuve de CLL, il est possible par une suite de montées et d'absorptions d'affaiblissements par des axiomes $\top$, d'obtenir une preuve où tout affaiblissement se trouve dans une suite ininterrompue d'inférences de même type et précédée immédiatement par un axiome de type id ou 1.*

Preuve 3.2.1 *Dans cette démonstration, nous dirons qu'un affaiblissement, dans une preuve de CLL, possède la propriété $\mathcal{W}-\mathcal{NORMAL}$ s'il se trouve dans une configuration ayant la forme décrite dans l'énoncé du théorème.*
Considérons une preuve quelconque $\mathcal{P}$ sans coupures de CLL. Nous allons montrer par induction sur la structure de $\mathcal{P}$ qu'il existe une preuve de CLL qui a la même conclusion que $\mathcal{P}$ et dont tous les affaiblissements possèdent la propriété $\mathcal{W}-\mathcal{NORMAL}$.
Le cas de base est trivial.
Le cas d'induction demande à être développé uniquement lorsque la dernière inférence I de $\mathcal{P}$ est un affaiblissement. Dans ce cas, nous appliquons l'hypothèse de récurrence à la preuve extraite de $\mathcal{P}$ qui a comme conclusion le prémisse de I et nous la remplaçons par une preuve dont tous les affaiblissements ont la propriété $\mathcal{W}-\mathcal{NORMAL}$.
Ensuite, en utilisant le théorème 2.3.1, nous montons I jusqu'aux axiomes.
Si une inférence résultant de cette montée, se trouve juste après un axiome de type $\top$, nous procédons alors à une absorption de celle-ci par l'axiome $\top$ au sens ou nous l'avons défini au-dessus.
Sinon elle se trouve placée juste après un axiome de type id ou 1. Elle possède donc la propriété $\mathcal{W}-\mathcal{NORMAL}$ et elle n'est pas venue perturber les affaiblissements au-dessous qui la possédait déjà.
En conclusion, tous les affaiblissements de la preuve obtenue vérifient la propriété $\mathcal{W}-\mathcal{NORMAL}$. □

2.2 Simplification du système d'inférence

Comme pour les contractions, ce théorème va nous permettre de simplifier le système d'inférence de CLL en fusionnant les inférences des configurations où les affaiblissements peuvent être localisés en une seule inférence. Ainsi les règles $w?$, id et 1 vont être remplacées par deux nouvelles, id' et $1'$ ainsi définies :

$$\frac{}{\vdash A,\ A^{\perp},\ ?\Delta}\ id' \qquad\qquad \frac{}{\vdash 1,\ ?\Delta}\ 1'$$

Nous retrouvons ici une proposition de [Andreoli, 1992]

3 CLL↑ : un système pour la construction ascendante des preuves

Après normalisation des contractions et des affaiblissements dans CLL, nous obtenons un nouveau système d'inférence CLL↑ plus adapté à la construction ascendante de preuves. Nous avons récapitulé ses règles dans la figure 3.1. Le système CLL↑ est équivalent à CLL, ce qui peut s'exprimer ainsi :

Théorème 3.3.1 *Tout séquent $\vdash \Delta$ de CLL est prouvable si et seulement si il est prouvable dans CLL↑.*

Preuve 3.3.1 *Il faut montrer tout d'abord que pour toute preuve $\mathcal{P}$ de CLL, il existe une preuve de CLL↑ qui a la même conclusion. Nous procédons en quatre étapes.*
Tout d'abord, par application du théorème 3.1.1, nous transformons la preuve $\mathcal{P}$ en une preuve $\mathcal{P}_1$ où les contractions sont disposées de façon à pouvoir être traduites ensuite dans CLL↑.
Puis nous insérons juste avant les inférences de type $\otimes$ et ?, les affaiblissements qui vont rendre possible la traduction des premières dans CLL↑. Nous obtenons une preuve $\mathcal{P}_2$.
Ensuite, à l'aide du théorème 3.2.1, nous montons tous les affaiblissements dans la preuve $\mathcal{P}_2$. Nous obtenons une preuve $\mathcal{P}_3$.
Enfin, il ne reste plus qu'à remplacer les blocs formés d'inférences de type id, 1, w?, $\otimes$, ? et c? par les inférences id', 1', $\otimes'$ et?' de CLL↑ et obtient une preuve $\mathcal{P}_4$ de CLL↑ qui a même conclusion que $\mathcal{P}$.
Réciproquement, il faut montrer que pour toute preuve $\mathcal{P}$ de CLL↑, il existe une preuve de CLL qui a la même conclusion. Il suffit pour cela de montrer que les règles nouvelles de CLL↑, id', 1', $\otimes'$ et?', sont admissibles dans CLL. Nous ne le ferons pas ici car cela ne pose aucune difficulté.□

Le système CLL↑, en lui-même, n'est pas original. La plupart de ceux qui ont étudié la construction ascendante de preuves dans CLL, l'ont utilisé en tant que cadre mieux adapté à cette tâche [Andreoli, 1992;

FIG. 3.1 - *Le système d'inférence de CLL↑*

groupe identité

$$\frac{}{\vdash A, A^{\perp}, ?\Delta}\ id'$$

groupe logique

négation

$$F^{\perp\perp} = F$$

$$\begin{array}{rl} (F \otimes G)^{\perp} = F^{\perp} \wp G^{\perp} & (F \wp G)^{\perp} = F^{\perp} \otimes G^{\perp} \\ 1^{\perp} = \perp & \perp^{\perp} = 1 \\ (F \& G)^{\perp} = F^{\perp} \oplus G^{\perp} & (F \oplus G)^{\perp} = F^{\perp} \& G^{\perp} \\ \top^{\perp} = 0 & 0^{\perp} = \top \\ (!F)^{\perp} = ?F^{\perp} & (?F)^{\perp} = !F^{\perp} \\ (\forall x F)^{\perp} = \exists x F^{\perp} & (\exists x F)^{\perp} = \forall x F^{\perp} \end{array}$$

opérateurs multiplicatifs

$$\frac{\vdash F_1, \Delta_1, ?\Delta \qquad \vdash F_2, \Delta_2, ?\Delta}{\vdash F_1 \otimes F_2, \Delta_1, \Delta_2, ?\Delta}\ \otimes' \qquad \frac{}{\vdash 1, ?\Delta}\ 1' \qquad \frac{\vdash F_1, F_2, \Delta}{\vdash F_1 \wp F_2, \Delta}\ \wp \qquad \frac{\vdash \Delta}{\vdash \perp, \Delta}\ \perp$$

où les formules de Δ_1 et Δ_2 ne sont pas de la forme $?F'$

opérateurs additifs

$$\frac{\vdash F_1, \Delta \qquad \vdash F_2, \Delta}{\vdash F_1 \& F_2, \Delta}\ \& \qquad \frac{}{\vdash \top, \Delta}\ \top \qquad \frac{\vdash F_1, \Delta}{\vdash F_1 \oplus F_2, \Delta}\ \oplus_1 \qquad \frac{\vdash F_2, \Delta}{\vdash F_1 \oplus F_2, \Delta}\ \oplus_2$$

opérateurs exponentiels

$$\frac{\vdash F, ?\Delta}{\vdash !F, ?\Delta}\ ! \qquad \frac{\vdash F, ?F, \Delta}{\vdash ?F, \Delta}\ ?'$$

quantificateurs

$$\frac{\vdash F[y/x], \Delta}{\vdash \forall x F, \Delta}\ \forall \qquad \frac{\vdash F[t/x], \Delta}{\vdash \exists x F, \Delta}\ \exists$$

avec y non libre dans la conclusion

TAB. 3.1 - *Permutabilité d'inférences dans les preuves de CLL↑*

$t_2 \backslash t_1$	$\otimes'$	$\wp$	&	$\oplus$	?'	!	$\forall$	$\exists$	$\perp$
$\otimes'$							*np*		
$\wp$	*np*						*np*		
$\perp$							*np*		
&	*np*	–	–	–	*np*	*np*	*np*	–	*np*
$\oplus$							*np*		
?'									
!	×	×	×	×	×	*np*	×	×	×
$\forall$							*np*		*np*
$\exists$							*np*		

Tammet, 1993]. Ce qu'on met en lumière ici, c'est qu'en menant dans CLL un processus de normalisation des preuves basé sur la permutabilité d'inférences, on aboutit tout naturellement à un tel système.
Maintenant, notre processus de normalisation n'est pas achevé à ce stade. Après les contractions et les affaiblissements, il s'agit d'*ordonner les autres inférences.* Pour cela, il est nécessaire de transposer les théorèmes sur les permutations et mouvements d'inférences de CLL dans CLL↑. Nous nous contenterons de fournir le tableau récapitulant la permutabilité d'inférences dans CLL↑ (voir tableau 3.1). Il s'obtient très facilement à partir du tableau correspondant de CLL accompagnant le théorème 2.2.1 si l'on se rappelle que les inférences de type $\otimes'$ et ?' ne sont que des condensés de déductions formées d'une inférence $\otimes$ ou ? suivie de contractions.

Remarque 3.3.1 *Il faut faire attention au sens dans CLL↑ des concepts de formule principale et formule active. Ainsi, considérons l'inférence suivante :*

$$\frac{\vdash F_1,\ ?G \quad \vdash F_2,\ ?G}{\vdash F_1 \otimes F_2,\ ?G}\ \otimes'$$

Dans la conclusion, ?G est considérée comme formule principale et dans les prémisses comme formule active. D'après la définition 2.2.1, une telle inférence n'est donc jamais en position de permutation dans une preuve et n'entrera pas dans le cadre de l'étude de permutabilité dans CLL↑.

4 Normalisation des preuves dans CLL↑

Pour achever le processus de normalisation, il reste à ordonner si possible les inférences résultant de l'application des règles logiques. Il va falloir pour cela, en monter certaines et en descendre d'autres.
Pour les inférences de type $\perp$, on sait déjà (cf section 4 du chapitre 2) qu'il va falloir les descendre étant donné que nous nous situons dans une perspective de construction des preuves de bas en haut.
Pour les autres, le seul critère est celui de la *facilité à les déplacer vers le haut ou vers le bas.* Il suffit pour cela d'observer le tableau 3.1. Il est ensuite facile d'étayer cette observation par une série de théorèmes qui ne sont qu'une transposition dans CLL↑ des théorèmes sur les mouvements d'inférences dans les preuves de CLL (cf section 3 du chapitre 2).
En conclusion, dans CLL↑, le classement des types d'inférences est le suivant :

$$\boxed{\begin{array}{l} T\uparrow = \{\ \otimes', \oplus, ?', \exists\ \} \\ T\downarrow = \{\ \wp, \perp, \&, !, \forall\ \} \end{array}}$$

On peut alors définir la notion de *preuve normale.* Celle-ci repose sur *un lien particulier entre toute conclusion intermédiaire et l'inférence qui l'a produite.* Nous aurions très bien pu la définir en précisant

le lien entre toute conclusion intermédiaire et l'inférence dont elle est un prémisse. Ce serait beaucoup moins pertinent quand on sait que l'on va ensuite utiliser cette définition pour construire les preuves de bas en haut : pour chaque conclusion intermédiaire connue, on sera alors amené à rechercher quelle inférence peut l'avoir produite.

Définition 3.4.1 *Une preuve de CLL↑ est dite* normale *si et seulement si toutes ses conclusions intermédiaires sont normales.*
Une conclusion intermédiaire normale vérifie la propriété suivante :

Si *elle est de la forme* $\vdash !F, ?\Delta$, *alors elle est la conclusion d'une inférence de type !*
sinon si *elle contient une formule introduite par une inférence d'un type appartenant à* $T\downarrow$
  alors *elle est la conclusion d'une inférence de ce type*
  sinon si *si elle est conclusion d'une inférence de type ?'*
    alors *toute formule active de cette inférence qui n'est pas un littéral négatif,*
      est introduite par l'inférence juste avant
    sinon si *elle est la conclusion d'une inférence de type* $\otimes'$, $\oplus$ *ou* $\exists$,
      alors *toute formule active de cette inférence qui n'a pas ? comme connecteur de tête*
        et qui n'est pas un littéral négatif, est la formule principale
        de l'inférence située juste avant.

Remarque 3.4.1 *Le fait que l'on écarte, pour la condition sur les formules actives des inférences d'un type appartenant à* $T\uparrow$, *les littéraux négatifs tient à ce que, pour les axiomes de type id, nous avons choisi de placer l'inférence utilisant le littéral positif avant celle utilisant le littéral négatif lorsqu'il y a conflit (voir section 5 du chapitre 2).*

Une caractéristique particulière des preuves normales est pour les inférences de type ?', $\otimes'$, $\oplus$ et $\exists$ d'identifier dans certaines conditions, formule active et formule principale de l'inférence précédente. Ainsi dans une construction ascendante des preuves, cela va nous amener dès que nous commencerons à décomposer certaines formules, à poursuivre en décomposant leurs sous-formules. Nous reviendrons en détail sur ce phénomène lorsque nous aborderons, dans le prochain chapitre, la construction des preuves elles-même. Andreoli l'avait déjà mis en lumière en proposant une forme particulière de preuves normales pour CLL : les "focusing proofs" [Andreoli, 1992]. La forme que nous proposons ici en est très proche. Il y a néanmoins deux différences :

- les inférences de type ! ne sont pas traitées de façon complètement satisfaisante par Andreoli qui, dans la démonstration d'un séquent de la forme $\vdash !F, ?\Delta$ ne choisit pas nécessairement $!F$ comme formule principale de la dernière inférence, alors que c'est possible ; il se prive ainsi d'une possibilité de limiter l'indéterminisme ;
- Andreoli a poussé plus loin la modification du système d'inférence si bien que toute preuve engendrée par ce système est normale. Il s'agit là d'une différence secondaire de présentation.

Illustrons maintenant la définition d'une preuve normale par un exemple.

Exemple 3.4.1 *Considérons la preuve de CLL↑ suivante :*

$$\dfrac{\dfrac{}{\vdash B,\ B^\perp}\,id' \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A,\ A^\perp,\ ?B}\,id'}{\vdash A,\ A^\perp \wp ?B}\,\wp}{\vdash A\oplus B,\ A^\perp\wp ?B}\,\oplus_1 \qquad \dfrac{}{\vdash C,\ C^\perp}\,id'}{\vdash A\oplus B,\ (A^\perp\wp ?B)\otimes C,\ C^\perp}\,\otimes}{\vdash B\otimes(A\oplus B),\ (A^\perp\wp ?B)\otimes C,\ C^\perp,\ B^\perp}\,\otimes$$

Cette preuve n'est pas normale pour une double raison :

- *La conclusion intermédiaire* $\vdash A\oplus B,\ A^\perp\wp ?B$ *n'est pas normale car elle contient la formule* $A^\perp\wp ?B$ *qui a été introduite par une inférence* $\wp$ *alors que sa formule principale est* $A\oplus B$ *;*

- *La conclusion intermédiaire* $\vdash B \otimes (A \oplus B),\ (A^{\perp} \wp ?B) \otimes C,\ C^{\perp},\ B^{\perp}$ *n'est pas normale car dans l'inférence qui produit cette conclusion, la formule active* $A \oplus B$ *n'est pas introduite par l'inférence juste avant.*

On peut normaliser cette preuve en descendant l'inférence $\wp$ *et montant l'inférence* $\otimes$ *située le plus bas. On obtient la preuve normale suivante :*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash B,\ B^{\perp}}\,id' \qquad \dfrac{\dfrac{}{\vdash A,\ A^{\perp},\ ?B}\,id'}{\vdash A \oplus B,\ A^{\perp},\ ?B}\,\oplus}{\vdash B \otimes (A \oplus B),\ A^{\perp},\ ?B,\ B^{\perp}}\,\otimes}{\vdash B \otimes (A \oplus B),\ A^{\perp} \wp ?B,\ B^{\perp}}\,\wp \qquad \dfrac{}{\vdash C,\ C^{\perp}}\,id'}{\vdash B \otimes (A \oplus B),\ (A^{\perp} \wp ?B) \otimes C,\ B^{\perp},\ C^{\perp}}\,\otimes$$

Nous allons maintenant généraliser le processus de normalisation illustré par l'exemple ci-dessus à l'aide d'un théorème qui va montrer que la classe des preuves normales est complète par rapport à l'ensemble des preuves de CLL↑.
La démonstration du théorème fournit aussi une procédure de normalisation des preuves.

Théorème 3.4.1 *Tout séquent prouvable dans CLL↑, y admet une preuve normale.*

Preuve 3.4.1 *La démonstration que nous proposons ici, correspond à une stratégie qui consiste à normaliser chaque conclusion intermédiaire en parcourant la preuve de haut en bas. D'autres stratégies sont possibles.*
Considérons une preuve quelconque $\mathcal{P}$ *de CLL↑. Nous allons montrer par induction sur la structure de* $\mathcal{P}$ *qu'il existe une preuve normale de CLL↑ qui a même conclusion que* $\mathcal{P}$.
Soit I la dernière inférence de $\mathcal{P}$. *Par hypothèse d'induction, tout prémisse de I (s'il en existe) admet une preuve normale de CLL↑. En prolongeant ces preuves normales par I, on obtient une nouvelle preuve* $\mathcal{P}'$ *qui a même conclusion que* $\mathcal{P}$ *mais qui n'est pas nécessairement normale. Ceci dépend du type de I. C'est pourquoi nous allons procéder à une analyse de cas selon le type de I.*

1. I est un axiome.
   La preuve $\mathcal{P}$ *est normale.*

2. I est d'un type appartenant à $T\downarrow$.
   La preuve $\mathcal{P}'$ *est normale. Il faut en effet remarquer que si I n'est pas de type !, sa conclusion ne peut pas avoir la forme* $\vdash !F, ?\Delta$.

3. I est d'un type appartenant à $T\uparrow$.
   Nous allons monter au maximum I dans la preuve $\mathcal{P}'$ *mais seulement dans les branches où la formule active de I n'a pas ? comme connecteur de tête et où elle n'est pas un littéral négatif. En plus, si I est de type* $\otimes'$, *il y a une indétermination dans la définition du mouvement ; si on laisse le hasard la régler, plusieurs conclusions intermédiaires qui vont être franchies, risquent de perdre leur caractère de conclusions intermédiaires normales. Pour éviter ceci, nous nous imposons de décomposer le mouvement général en une suite de mouvements élémentaires définis ainsi :*

   - *Si une au moins des inférences qui précède immédiatement I, est d'un type appartenant à* $T\downarrow$, *on la permute avec I ;*
   - *Si toutes les inférences sont d'un type appartenant à* $T\uparrow$, *on monte I (dans l'une des deux branches seulement s'il y en a deux), juste au-dessus de la première conclusion intermédiaire qui soit produite par une inférence d'un type appartenant à* $T\downarrow$ *ou qui contienne une formule ayant ? comme connecteur de tête et active dans l'inférence qui suit immédiatement.*

On peut vérifier que chaque mouvement élémentaire, tel qu'il vient d'être défini, préserve le caractère normal de toute conclusion intermédiaire qui est franchie. Les seules qui, éventuellement, ne sont pas normales, sont les conclusions des inférences en train d'être montées.
Mais au terme du mouvement général, leurs formules actives ont? comme connecteur de tête, sont des littéraux négatifs ou sont introduites par une inférence située juste avant.
Les conclusions de ces inférences peuvent toutefois ne pas être normales si elles contiennent des formules introduites par des inférences d'un type appartenant à $T\downarrow$. Mais il est facile de corriger cette anomalie en descendant ces inférences juste après la conclusion intermédiaire en cause. Elle devient alors normale et les autres le restent.

□

Le théorème de normalisation peut être complété par un autre qui montre que l'ordre dans lequel on applique les règles d'inférence de $T\downarrow\setminus\{!\}$, dans la construction d'une preuve de bas en haut, est indifférent. [Andreoli, 1992] avait déjà mis en évidence cette propriété sous le nom de *réversibilité* car elle revient à dire que si la conclusion d'une inférence est prouvable, alors ses prémisses aussi.

Théorème 3.4.2 *Soit $\vdash F, \Delta$ un séquent prouvable de $CLL\uparrow$ tel que F ait un connecteur de tête appartenant à $T\downarrow\setminus\{!\}$. Alors il existe une preuve normale de $\vdash F, \Delta$ se terminant par une inférence dont la formule principale est F.*

Preuve 3.4.2 *On considère dans $CLL\uparrow$ une preuve quelconque de $\vdash F, \Delta$. On descend en bas de la preuve l'inférence introduisant F puis on normalise les sous-preuves des prémisses de cette inférence.*
En prolongeant les preuves normales obtenues par l'inférence finale, on obtient la preuve normale cherchée.□

Conclusion

Les preuves normales obtenues sont très voisines des "focusing proofs" de [Andreoli, 1992]. Mais l'intérêt de la démarche est de mettre en avant les rapports entre les notions de permutabilité d'inférences, mouvement d'inférences et normalisation de preuves. Il est l'illustration de la méthode qui va être par la suite appliquée à d'autres fragments ou avec un autre sens de construction des preuves.
Nous aurions pu aussi chercher à intégrer les résultats obtenus par [Lincoln and Shankar, 1994] concernant la gestion des variables quantifiées au premier ordre. Ils ont montré qu'on peut permuter les inférences de type $\exists$ avec celle de type $\forall$ dans les preuves en utilisant une technique de *skolémisation dynamique des variables*. Malheureusement, cette possibilité, comme le signalent eux-mêmes les auteurs, est partiellement incompatible avec les autres résultats de permutabilité. C'est ce qui explique le fait que nous ne les ayons pas pris en compte. Nous reviendrons toutefois sur ces travaux quand nous aborderons les stratégies ascendantes de construction de preuves dans le chapitre 5 et que nous verrons comment gérer la quantification au premier ordre.

Chapitre 4

Normalisation dans CLL et preuves en chaînage avant

Introduction

D'une façon générale, les méthodes de preuve en chaînage avant les plus connues sont la *résolution* de [Robinson, 1965] et la *méthode inverse* de Maslov [Lifschitz, 1989]. Même si elles se sont développées indépendamment l'une de l'autre, elles sont très proches. Elles consistent toutes deux à transformer une formule à démontrer en un ensemble de formules d'un format bien précis, appelées *clauses*. Puis à l'aide d'un nombre très limité de règles d'inférences ou *règles de résolution*, on engendre de nouvelles clauses jusqu'à ce qu'on en obtienne une particulière qui constitue le *but* attendu.
[Mints, 1990] a étendu la méthode inverse pour des logiques modales en proposant un schéma de transformation d'un système présenté dans le formalisme du calcul des séquents dans celui de la résolution. [Tammet, 1993] a appliqué ce schéma à la logique linéaire.
A la différence de ces travaux, nous allons conserver le cadre du calcul des séquents afin de pouvoir exploiter la propriété de permutabilité d'inférences liée à celui-ci. A l'opposé du chapitre précédent, nous nous placerons dans une perspective de construction des preuves en chaînage avant. La normalisation des preuves va encore consister à ordonner les inférences en leur sein de façon à rendre leur construction descendante la plus efficace possible. Mais en quoi le changement de sens de construction des preuves influe-t-il sur la forme des preuves normales? D'après ce que nous avons vu dans le paragraphe 4.1, il va falloir encore monter les contractions au maximum mais par contre le sens de déplacement des affaiblissements va se trouver inversé : ils devront être descendus au maximum. Nous verrons toutefois que ce mouvement seul ne permet pas d'assurer un bon contrôle de ceux-ci au cours de la construction d'une preuve de haut en bas ; ils sont encore une source importante d'indéterminisme.
Ce problème rejoint celui du contrôle d'une autre forme d'affaiblissement qui n'est pas liée à l'application de la règle $w?$ mais à celle de $\top$. Les axiomes qu'elle engendre, ont la forme suivante : $\dfrac{}{\vdash \top, \Delta}\top$
Il est difficile de contrôler leur production car le multi-ensemble de formules Δ est totalement arbitraire. La seule contrainte qu'on peut lui imposer, est qu'il contienne uniquement des sous-formules du but à prouver ; en effet, nous cherchons à construire des preuves sans coupures qui respectent donc la propriété de la sous-formule. Mais le nombre de ces formules n'est pas limité donc on risque souvent d'engendrer une infinité d'axiomes $\top$.
[Tammet, 1993] propose de dissocier au sein des axiomes $\top$, l'introduction de la constante $\top$ de l'affaiblissement constitué par l'introduction du multi-ensemble de formules arbitraires Δ. Il a donc remplacé la règle correspondante par deux nouvelles règles. Ainsi les affaiblissements correspondants vont pouvoir être descendus dans les preuves comme les affaiblissements classiques pour mieux être contrôlés. Par rapport à ces derniers, ils nécessitent toutefois un traitement particulier ce qui va donner lieu à la distinction introduite par [Tammet, 1993] entre *séquents affaiblissables* et *séquents non affaiblissables*.
Nous allons intégrer cette idée en proposant une modification du système d'inférence dont la justification

est que tous les affaiblissements, quelle que soit leur source ($w?$ ou $\top$), y sont mobiles.
Dans le système CLL$\downarrow$ obtenu, nous allons pouvoir maintenant descendre *tous les affaiblissements* au maximum. [Tammet, 1993] s'en était arrêté là mais nous verrons que ce n'est pas suffisant pour contrôler leur production dans une construction en avant des preuves.
Nous allons proposer une solution qui repose sur la distinction au sein d'une preuve entre *formules de base*, partiellement issues d'axiomes, et *formules d'affaiblissement*, totalement issues d'affaiblissements. Ces dernières posent problème. C'est pourquoi on essaie de les introduire seulement quand elles se combinent avec une formule de base. Ceci nous amène à une dernière modification du système d'inférence que nous allons spécialiser dans la démonstration d'un séquent particulier $\vdash \Delta$. Les affaiblissements ainsi que les contractions y sont complètement intégrés aux inférences logiques. Mais cela nécessite la définition à partir de $\vdash \Delta$, d'un ensemble particulier de formules où l'on puisera les formules d'affaiblissements à introduire dans les preuves. C'est pourquoi, nous l'appellerons la *base d'affaiblissement* associée à $\vdash \Delta$. Dans ce nouveau système CLL$\downarrow_\Delta$, on déplacera les inférences logiques pour obtenir les preuves normales.

1 Séquents affaiblissables : le système d'inférence CLL$\downarrow$

1.1 Dissociation des axiomes de type $\top$

Nous allons remplacer la règle $\top$ par deux nouvelles :

- la règle $\top'$ qui a comme fonction d'engendrer la constante $\top$ $\quad \dfrac{}{\vdash \top}\,\top'$

- la règle $w_\top$ qui produit une forme spécifique d'affaiblissement $\quad \dfrac{\vdash \Delta}{\vdash F, \Delta}\,w_\top$

Contrairement à $w?$, la règle $w_\top$ n'est soumise à aucune restriction quant à la forme de F. Par contre, elle ne peut pas, pour être admissible dans CLL, être utilisée à partir de n'importe quel séquent $\vdash \Delta$. Si l'on fixe comme condition que la constante $\top$ doit être présente dans le séquent Δ, on obtient bien une règle admissible mais cette condition est trop restrictive : elle va empêcher de descendre les inférences $w_\top$ aussi loin qu'on le voudrait. En effet, elles seront bloquées par toute inférence où la constante $\top$ est active.

Ceci va nous amener à étudier dans quelle mesure la propriété pour un séquent d'être affaiblissable qui, au départ, est liée à la présence de $\top$, peut se transmettre une fois que cette constante est entrée en combinaison avec d'autres formules. Nous traduirons le fait que l'on puisse appliquer la règle $w_\top$ à un séquent, c'est-à-dire qu'il est affaiblissable, en le marquant par un booléen que nous appellerons son *indicateur d'affaiblissement*. Ainsi ce séquent sera représenté de la façon suivante : $\overset{a}{\vdash}\Delta$ où son indicateur d'affaiblissement a vaudra 1 si le séquent est affaiblissable et 0 s'il ne l'est pas.

1.2 Séquents affaiblissables

Nous allons passer en revue les règles de CLL pour examiner dans quelle mesure elle permettent à la propriété d'être ou non affaiblissable, de se transmettre des prémisses vers la conclusion. Elles vont alors être complétées pour indiquer comment calculer l'indicateur d'affaiblissement attaché à leur conclusion à partir de ceux attachés à leurs prémisses.

1.2.1 Les axiomes

Au niveau des axiomes, seuls ceux de type $\top'$ produisent des séquents affaiblissables d'où les règles :

$$\frac{}{\overset{0}{\vdash} A, A^\perp}\,\mathrm{id} \qquad \frac{}{\overset{0}{\vdash} 1}\,1 \qquad \frac{}{\overset{1}{\vdash} \top}\,\top'$$

1.2.2 Les règles non sensibles au contexte avec un prémisse.

Ces règles d'inférences sont neutres dans la transmission de la propriété. Donc si a est le booléen attaché au prémisse, a est attaché à la conclusion. Ceci concerne les règles $\wp$, $\oplus_1$, $\oplus_2$, ?, c? et $\exists$.
Nous avons choisi de regrouper les règles w? et $w_\top$ en une seule règle car nous nous sommes rendus compte que, dans une preuve, en permutant un affaiblissement classique (c'est-à-dire une inférence de type w? ou $\perp$) avec d'autres inférences, il pouvait se transformer en affaiblissement de type $w_\top$ et vice-versa. Nous aurons désormais une seule règle d'affaiblissement w se présentant ainsi :

$$\frac{\vdash^{a} \Delta}{\vdash^{a} F, \Delta}\, w \qquad \textit{si } a = 0, \textit{ alors } F \textit{ est de la forme } ?F' \textit{ ou } \perp$$

Une inférence de type w sera qualifiée d'*affaiblissement classique* lorsque a vaut 0 et d'*affaiblissement spécifique* lorsque a vaut 1.

1.2.3 Les règles sensibles au contexte avec un prémisse.

a) La règle !.

Expliquons pourquoi elle bloque la transmission de la propriété d'être affaiblissable. Supposons que ce ne soit pas le cas ; nous pourrions avoir alors ce genre de déduction :

$$\dfrac{\dfrac{\vdash^{1} F, ?\Delta}{\vdash^{1} !F, ?\Delta}\,!}{\vdash^{1} !F, G, ?\Delta}\, w_\top$$

Pour que cette déduction soit correcte relativement à CLL, il faut pouvoir remonter l'inférence $w_\top$, le but étant pour une preuve complète de reconstituer les axiomes $\top$. Il faut donc que l'inférence ! soit permutable avec l'inférence $w_\top$. Or, ce n'est pas le cas si G n'a pas ? comme connecteur de tête.
La règle ! se présente donc ainsi :

$$\frac{\vdash^{a} F, ?\Delta}{\vdash^{0} !F, ?\Delta}\,!$$

b) La règle $\forall$.

Elle est neutre malgré la condition sur la variable quantifiée. On résout alors les conflits éventuels par un renommage. Considérons par exemple la preuve :

$$\dfrac{\dfrac{\mathcal{P}\left\{\begin{array}{c}\vdots\\ \vdash^{1} F[y/x], \Delta\end{array}\right.}{\vdash^{1} \forall x F, \Delta}\,\forall}{\vdash^{1} \forall x F, G, \Delta}\, w_\top$$

Pour que la preuve soit correcte relativement à CLL, il faut pouvoir permuter l'inférence $\forall$ avec l'inférence $w_\top$. Or si y est libre dans G, la condition relative à y pour $\forall$ est violée. On préserve la correction de la

preuve en choisissant une variable nouvelle z et en renommant y en z dans la preuve $\mathcal{P}$. La permutation est alors possible et l'on obtient :

$$\dfrac{\dfrac{\mathcal{P}[z/y]\left\{\begin{array}{c}\vdots\\ \vdash^{1} F[y/x], \Delta\end{array}\right.}{\vdash^{1} F[z/x], G, \Delta}\, w_{\top}}{\vdash^{1} \forall x F, G, \Delta}\, \forall$$

La règle se présente donc ainsi :

$$\dfrac{\vdash^{a} F[y/x], \Delta}{\vdash^{a} \forall x F, \Delta}\, \forall \qquad y \textit{ non libre dans } \forall x F, \Delta$$

1.2.4 Les règles avec deux prémisses.

La règle $\otimes$ transmet la propriété d'être affaiblissable même si elle ne vient que d'un seul prémisse car elle opère une juxtaposition des contextes initiaux. Pour la règle &, c'est le contraire : vu qu'elle opère une superposition des contextes initiaux, c'est la propriété de ne pas être affaiblissable qu'elle transmet, même ne venant que d'un seul prémisse. D'où la forme de ces deux règles :

$$\dfrac{\vdash^{a_1} F_1, \Delta_1 \quad \vdash^{a_2} F_2, \Delta_2}{\vdash^{a_1 \vee a_2} F_1 \otimes F_2, \Delta_1, \Delta_2}\, \otimes \qquad \dfrac{\vdash^{a_1} F_1, \Delta \quad \vdash^{a_2} F_2, \Delta}{\vdash^{a_1 \wedge a_2} F_1 \& F_2, \Delta}\, \&$$

1.3 Le système CLL↓.

Nous disposons maintenant d'un système d'inférence où tout d'affaiblissement de quelque forme que ce soit, est mobile et va pouvoir être descendu au maximum dans les preuves. Il est défini par la figure 4.1 et noté CLL↓. On retrouve ainsi la proposition de [Tammet, 1993] sous une forme un peu différente dans la mesure où nous avons introduit une notion nouvelle de marquage d'un séquent par un booléen pour indiquer s'il est affaiblissable ou pas.
Nous allons montrer que CLL↓ est équivalent à CLL ce qui exige une étude préalable de la permutabilité d' inférences en son sein.

1.3.1 Permutabilité d'inférences dans CLL↓

Cette notion doit être légèrement modifiée par rapport à sa définition initiale 2.2.2 dans CLL car il faut prendre en compte le fait que les séquents sont affectés d'un indice signalant s'ils sont affaiblissables ou pas. Pour cela, on complète la définition 2.2.2 de la façon suivante :

les indices d'affaiblissement attachés aux hypothèses et à la conclusion dans le résultat de la permutation doivent être les mêmes que dans l'objet de la permutation.

Munis de cette définition, il est alors facile de reprendre le tableau du théorème 2.2.1 et de l'adapter à CLL↓. Nous obtenons ainsi le tableau 4.1. D'après celui-ci, nous allons pouvoir monter facilement les affaiblissements dans les preuves de CLL↓. Les seuls obstacles sont les inférences de type ! mais nous verrons qu'elles en sont vraiment seulement dans le cas où les affaiblissements devant être montés introduisent la constante $\perp$.

FIG. 4.1 - *Le système d'inférence de CLL↓*

groupe identité

$$\frac{}{\vdash^{0} A, A^{\perp}}\ \mathrm{id}$$

groupe logique

négation

$$F^{\perp\perp} = F$$

$$\begin{array}{rl} (F \otimes G)^{\perp} = F^{\perp} \wp G^{\perp} & (F \wp G)^{\perp} = F^{\perp} \otimes G^{\perp} \\ 1^{\perp} = \perp & \perp^{\perp} = 1 \\ (F \& G)^{\perp} = F^{\perp} \oplus G^{\perp} & (F \oplus G)^{\perp} = F^{\perp} \& G^{\perp} \\ \top^{\perp} = 0 & 0^{\perp} = \top \\ (!F)^{\perp} = ?F^{\perp} & (?F)^{\perp} = !F^{\perp} \\ (\forall x F)^{\perp} = \exists x F^{\perp} & (\exists x F)^{\perp} = \forall x F^{\perp} \end{array}$$

opérateurs multiplicatifs

$$\frac{\vdash^{a_1} F_1, \Delta_1 \quad \vdash^{a_2} F_2, \Delta_2}{\vdash^{a_1 \vee a_2} F_1 \otimes F_2, \Delta_1, \Delta_2}\ \otimes \qquad \frac{}{\vdash^{0} 1}\ 1 \qquad \frac{\vdash^{a} F_1, F_2, \Delta}{\vdash^{a} F_1 \wp F_2, \Delta}\ \wp$$

opérateurs additifs

$$\frac{\vdash^{a_1} F_1, \Delta \quad \vdash^{a_2} F_2, \Delta}{\vdash^{a_1 \wedge a_2} F_1 \& F_2, \Delta}\ \& \qquad \frac{}{\vdash^{1} \top}\ \top' \qquad \frac{\vdash^{a} F_1, \Delta}{\vdash^{a} F_1 \oplus F_2, \Delta}\ \oplus_1 \qquad \frac{\vdash^{a} F_2, \Delta}{\vdash^{a} F_1 \oplus F_2, \Delta}\ \oplus_2$$

opérateurs exponentiels

$$\frac{\vdash^{a} F, ?\Delta}{\vdash^{0} !F, ?\Delta}\ ! \qquad \frac{\vdash^{a} F, \Delta}{\vdash^{a} ?F, \Delta}\ ? \qquad \frac{\vdash^{a} ?F, ?F, \Delta}{\vdash^{a} ?F, \Delta}\ \mathrm{c}?$$

quantificateurs

$$\frac{\vdash^{a} F[y/x], \Delta}{\vdash^{a} \forall x F, \Delta}\ \forall \qquad \frac{\vdash^{a} F[t/x], \Delta}{\vdash^{a} \exists x F, \Delta}\ \exists$$

avec y non libre dans la conclusion

affaiblissement

$$\frac{\vdash^{a} \Delta}{\vdash^{a} F, \Delta}\ w \qquad \textit{avec, si } a = 0,\ F \textit{ de la forme } ?F' \textit{ ou } \perp$$

TAB. 4.1 - *Permutabilité d'inférences dans les preuves de CLL↓*

$t_2 \backslash t_1$	$\otimes$	$\wp$	&	$\oplus$	?	c?	!	$\forall$	$\exists$	w
$\otimes$							np			
$\wp$	np						np			
&	np	–	–	np	np	–	np	–	np	np
$\oplus$							np			
?										
c?	np									
!	×	×	×	×	np		×	×	×	
$\forall$							np		np	
$\exists$							np			
w							np			

Donc cela ne nous empêchera pas d'établir la correction de CLL↓ par rapport à CLL qui repose essentiellement sur la reconstitution des axiomes $\top$ dont les deux aspects ont été dissociés dans CLL↓. La complétude est facile à montrer.

Théorème 4.1.1 *Tout séquent $\vdash \Delta$ est prouvable dans CLL si et seulement si il existe un booléen a tel que le séquent $\overset{a}{\vdash} \Delta$ est prouvable dans CLL↓.*

Preuve 4.1.1 *Commençons par démontrer la correction de CLL↓ relativement à CLL. Soit une preuve $\mathcal{P}$ de CLL↓ de conclusion $\overset{a}{\vdash} \Delta$. Montons successivement et au maximum dans $\mathcal{P}$ tous les affaiblissements spécifiques. D'après le tableau 4.1, ils peuvent être montés jusqu'aux axiomes $\top'$. Après ce mouvement, ces affaiblissements vont tous se situer dans des configurations de la forme :*

$$\dfrac{\dfrac{\dfrac{}{\overset{1}{\vdash} 1}\,\top'}{\overset{1}{\vdash} 1, F_1}\,w}{\vdots}$$
$$\dfrac{\vdots}{\overset{1}{\vdash} 1, F_1, \cdots, F_n}\,w$$

Nous allons remplacer ces configurations par des axiomes de type $\top$ de CLL qui auront la forme :

$$\dfrac{}{\vdash 1, F_1, \cdots, F_n}\,\top$$

Toutes les autres inférences ont une traduction dans CLL si bien que nous obtenons une preuve de CLL qui a comme conclusion $\vdash \Delta$.
Montrons maintenant la complétude de CLL↓ relativement à CLL. Considérons une preuve $\mathcal{P}$ de CLL. Nous allons remplacer chaque inférence de $\mathcal{P}$ par une inférence de CLL↓ en parcourant la preuve de haut en bas. Simultanément, nous allons inférer les indicateurs d'affaiblissement des conclusions intermédiaires. Les seules inférences qui n'ont pas d'équivalent dans CLL↓ sont les axiomes $\top$ mais comme on vient de le voir, ils peuvent être traduits dans CLL↓ par un axiome $\top'$ suivi d'affaiblissements. En fin de compte, on obtient une preuve de CLL↓ qui a la même conclusion que $\mathcal{P}$. □

2 Formules de base et d'affaiblissement

Les affaiblissements étant la principale source d'indéterminisme dans la construction descendante des preuves, nous allons commencer par traiter ce problème.

Nous avons vu qu'il fallait descendre ces inférences au maximum dans les preuves. Le système CLL↓ qui a été conçu pour cela, facilite ce mouvement. Ainsi, on peut amener chaque affaiblissement juste au-dessus de l'inférence où la formule qu 'il introduit, devient active (sauf si cette inférence est de type $\wp$ car il y a alors deux formules actives). [Tammet, 1993] a pris en compte cette descente des affaiblissements dans le système qu'il propose. Mais cela ne suffit pas à assurer un contrôle de ceux-ci dans la construction descendante d'une preuve. Considérons par exemple les deux déductions suivantes de CLL↓ :

$$\dfrac{\dfrac{\overset{0}{\vdash} \Delta}{\overset{0}{\vdash} ?F, \Delta}\,w}{\overset{0}{\vdash} ?F \oplus G, \Delta}\,\oplus_1 \qquad\qquad \dfrac{\dfrac{\overset{0}{\vdash} G,\ \Delta}{\overset{0}{\vdash} ?F,\ G,\ \Delta}\,w}{\overset{0}{\vdash} ?F \wp G, \Delta}\,\wp$$

Dans les deux déductions, l'affaiblissement est descendu au maximum. Mais dans celle de gauche, cela n'améliore en rien le contrôle sur celle-ci à partir de son hypothèse $\overset{0}{\vdash} \Delta$: une telle déduction peut être engendrée à partir de n'importe quelle conclusion intermédiaire déjà produite.
Ce n'est pas le cas de la déduction de droite qui ne peut pas être produite à partir de n'importe quelle conclusion intermédiaire déjà établie comme hypothèse : celle-ci doit contenir la formule G.
Si l'on analyse les fondements d'une telle différence, on s'aperçoit qu'à gauche, la formule $?F \oplus G$ est totalement issue de l'affaiblissement tandis qu'à droite, la formule $?F \wp G$ ne l'est que partiellement.
Nous allons généraliser cette distinction en rangeant les formules au sein d'une preuve en deux catégories : *les formules de base* et *les formules d'affaiblissement.*

2.1 Définition

Définition 4.2.1 *Dans une preuve de CLL↓, une formule est dite de base* (B-formule) *si elle contient au moins un littéral (une formule atomique ou sa négation) introduit par un axiome. Dans le cas contraire, elle est dite d'affaiblissement* (A-formule).

Illustrons cette définition sur l'exemple de la preuve $\mathcal{P}$ suivante :

Exemple 4.2.1

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\overset{0}{\vdash} a,\ a^\perp}\,id}{\overset{0}{\vdash} a,\ a^\perp,\ ?b(x)}\,w}{\overset{0}{\vdash} a \oplus c(x),\ a^\perp,\ ?b(x)}\,\oplus_1}{\overset{0}{\vdash} a \oplus c(x),\ c(x)^\perp \oplus a^\perp,\ ?b(x)}\,\oplus_2 \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\overset{0}{\vdash} c(x),\ c(x)^\perp}\,id}{\overset{0}{\vdash} c(x),\ c(x)^\perp,\ ?d(x)}\,w}{\overset{0}{\vdash} c(x),\ c(x)^\perp \oplus a^\perp,\ ?d(x)}\,\oplus_1}{\overset{0}{\vdash} a \oplus c(x),\ c(x)^\perp \oplus a^\perp,\ ?d(x)}\,\oplus_2}{\overset{0}{\vdash} a \oplus c(x),\ c(x)^\perp \oplus a^\perp,\ ?b(x)\&?d(x)}\,\&}{\overset{0}{\vdash} (a \oplus c(x))\wp(c(x)^\perp \oplus a^\perp),\ ?b(x)\&?d(x)}\,\wp}{\overset{0}{\vdash} \exists x((a \oplus c(x))\wp(c(x)^\perp \oplus a^\perp)),\ ?b(x)\&?d(x)}\,\exists}{\overset{0}{\vdash} \exists x((a \oplus c(x))\wp(c(x)^\perp \oplus a^\perp)),\ \forall x(?b(x)\&?d(x))}\,\forall}{\overset{0}{\vdash} [\exists x((a \oplus c(x))\wp(c(x)^\perp \oplus a^\perp))]\wp[\forall x(?b(x)\&?d(x))]}\,\wp$$

Dans la preuve $\mathcal{P}$, la formule $\forall x(?b(x)\&?d(x))$ est une A-formule alors que $[\exists x((a \oplus c(x))\wp(c(x)^\perp \oplus a^\perp))]\wp[\forall x(?b(x)\&?d(x))]$ *est une B-formule car elle contient par exemple la*

formule atomique a qui est introduite par un axiome (même si elle est aussi introduite d'une autre façon dans l'inférence $\oplus_2$ de gauche).

2.2 Le contrôle des inférences produisant les A-formules.

Dans une construction des preuves descendante, ce sont donc non seulement les affaiblissements mais l'ensemble des inférences qui participent à la production des A-formules qu'il est difficile de contrôler. Dans l'exemple 4.3.1, pour la formule $\forall x(?b(x)\&?d(x))$, il s'agit de deux affaiblissements suivis d'une inférence de type & puis d'une autre de type $\forall$. Cette suite de quatre inférences peut être déclenchée à partir de n'importe quelle conclusion intermédiaire déjà établie. Le contrôle est donc nul.
Il devient réel à partir du moment où la A-formule $\forall x(?b(x)\&?d(x))$ entre en combinaison avec une B-formule $\exists x((a \oplus c(x))\wp(c(x)^\perp \oplus a^\perp))$ dans la dernière inférence de type $\wp$.

Ainsi, un moyen de contrôler la production de A-formules est de la retarder jusqu'au moment où celles-ci deviennent actives dans des inférences où elles se combinent avec des B-formules pour former des disjonctions multiplicatives ou des conjonctions.

La traduction d'un tel principe dans le processus de normalisation des preuves de CLL$\downarrow$ va nous amener à descendre au maximum dans une preuve toutes les inférences qui contribuent à produire les A-formules. L'objectif est pour une A-formule donnée, de rassembler toutes les inférences qui participent à sa construction en un bloc situé juste avant une inférence où elle entre en combinaison avec une B-formule.
Appliquons ce principe à la preuve $\mathcal{P}$ qui nous a servi d'illustration jusqu'à maintenant. Il faut descendre les 4 inférences qui contribuent à la production de la formule $\forall x(?b(x)\&?d(x))$ juste avant qu'elle se combine avec la B-formule $\exists x((a \oplus c(x))\wp(c(x)^\perp \oplus a^\perp))$ dans la dernière inférence $\wp$. C'est possible et nous obtenons la preuve $\mathcal{P}'$ suivante :

$$
\begin{array}{c}
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash^0 a,\ a^\perp}\,id}{\vdash^0 a \oplus c(x),\ a^\perp}\,\oplus_1}{\vdash^0 a \oplus c(x),\ c(x)^\perp \oplus a^\perp}\,\oplus_2}{\vdash^0 (a \oplus c(x))\wp(c(x)^\perp \oplus a^\perp)}\,\wp}{\vdash^0 \exists x((a \oplus c(x))\wp(c(x)^\perp \oplus a^\perp))}\,\exists}{\vdash^0 \exists x((a \oplus c(x))\wp(c(x)^\perp \oplus a^\perp)),\ ?b(x)}\,w \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash^0 c(x),\ c(x)^\perp}\,id}{\vdash^0 c(x),\ c(x)^\perp \oplus a^\perp}\,\oplus_1}{\vdash^0 a \oplus c(x),\ c(x)^\perp \oplus a^\perp}\,\oplus_2}{\vdash^0 (a \oplus c(x))\wp(c(x)^\perp \oplus a^\perp)}\,\wp}{\vdash^0 \exists x((a \oplus c(x))\wp(c(x)^\perp \oplus a^\perp))}\,\exists}{\vdash^0 \exists x((a \oplus c(x))\wp(c(x)^\perp \oplus a^\perp)),\ ?d(x)}\,w}{\vdash^0 \exists x((a \oplus c(x))\wp(c(x)^\perp \oplus a^\perp)),\ b(x)\&?d(x)}\,\&}{\vdash^0 \exists x((a \oplus c(x))\wp(c(x)^\perp \oplus a^\perp)),\ \forall x(?b(x)\&?d(x))}\,\forall}{\vdash^0 [\exists x((a \oplus c(x))\wp(c(x)^\perp \oplus a^\perp))]\wp[\forall x(?b(x)\&?d(x))]}\,\wp
\end{array}
$$

Ceci nous suggère tout naturellement de simplifier le système d'inférence en étendant la règle d'affaiblissement w lorsqu'elle s'applique aux séquents non affaiblissables : jusqu'à maintenant, elle ne permet d'introduire que des formules de la forme $?F$ ou $\perp$; pourquoi ne pas lui permettre d'introduire n'importe quelle A-formule ? Les autres règles seraient alors réservées à la production des B-formules. En descendant au maximum ces affaiblissements étendus, nous pourrions faire en sorte que toute A-formule ne soit introduite que lorsque l'on en a besoin pour produire une B-formule. L'indéterminisme dans la construction descendante des preuves serait ainsi sérieusement réduit.

Dans un tel système, la preuve $\mathcal{P}'$ pourrait devenir :

$$
\begin{array}{c}
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash^0 a,\ a^\perp}\,id}{\vdash^0 a \oplus c(x),\ a^\perp}\,\oplus_1}{\vdash^0 a \oplus c(x),\ c(x)^\perp \oplus a^\perp}\,\oplus_2}{\vdash^0 (a \oplus c(x))\wp(c(x)^\perp \oplus a^\perp)}\,\wp}{\vdash^0 \exists x((a \oplus c(x))\wp(c(x)^\perp \oplus a^\perp))}\,\exists}{\vdash^0 \exists x((a \oplus c(x))\wp(c(x)^\perp \oplus a^\perp)),\ \forall x(?b(x)\&?d(x))}\,w}{\vdash^0 [\exists x((a \oplus c(x))\wp(c(x)^\perp \oplus a^\perp))]\wp[\forall x(?b(x)\&?d(x))]}\,\wp
\end{array}
$$

Une telle solution est séduisante mais pour qu'elle puisse se concrétiser, il faut répondre positivement aux deux questions suivantes :

- *Dans une preuve de CLL↓, est-il toujours possible de descendre les inférences contribuant à la production d'une A-formule de façon à les rassembler juste avant une inférence où cette A-formule se combine à une B-formule ?*
- *A partir d'un séquent donné, est-il possible de délimiter a priori un ensemble de formules d'où sera issue toute A-formule partie prenante d'une preuve quelconque de ce séquent ?*

ce que nous allons faire dans les sections suivantes.

3 Descente des inférences produisant les A-formules

Commençons par examiner les difficultés rencontrées dans ce mouvement.

3.1 Problème avec ⊗

L'exemple suivant montre qu'il n'est pas toujours possible de répondre positivement à la première question. Considérons la preuve de CLL↓ :

$$
\begin{array}{c}
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash^0 a,\ a^\perp}\,id}{\vdash^0 a,\ a^\perp,\ ?c}\,w \quad \dfrac{\dfrac{}{\vdash^0 b,\ b^\perp}\,id}{\vdash^0 b,\ b^\perp,\ ?d}\,w}{\vdash^0 a,\ a^\perp,\ b,\ b^\perp,\ ?c\otimes ?d}\,\otimes}{\vdash^0 a,\ a^\perp,\ b,\ a^\perp \oplus b^\perp,\ ?c\otimes ?d}\,\oplus_2 \quad \dfrac{\dfrac{\dfrac{}{\vdash^0 a,\ a^\perp}\,id}{\vdash^0 a,\ a^\perp,\ ?c}\,w \quad \dfrac{\dfrac{}{\vdash^0 b,\ b^\perp}\,id}{\vdash^0 b,\ b^\perp,\ ?d}\,w}{\vdash^0 a,\ a^\perp,\ b,\ b^\perp,\ ?c\otimes ?d}\,\otimes}{\vdash^0 a,\ a^\perp \oplus b^\perp,\ b,\ b^\perp,\ ?c\otimes ?d}\,\oplus_1}{\vdash^0 a^\perp \& b^\perp,\ a^\perp \oplus b^\perp,\ a,\ b,\ ?c\otimes ?d}\,\&}{\vdash^0 (a^\perp \& b^\perp)\wp(?c\otimes ?d),\ a^\perp \oplus b^\perp,\ a,\ b}\,\wp
\end{array}
$$

Dans une construction de cette preuve descendante, on ne peut contrôler l'introduction de la A-formule $?c \otimes ?d$ que lorsqu'elle devient active dans la dernière inférence $\wp$ aux côtés de la B-formule $a^\perp \& b^\perp$. Il

faudrait donc descendre les deux inférences $\otimes$ qui l'engendrent juste avant celle-ci. Malheureusement, elles seront bloquées dans leur mouvement par l'inférence &.

> *Ceci nous amène à restreindre notre ambition initiale : nous laisserons de côté les A-formules qui contiennent le connecteur $\otimes$.*

Les inférences de type & ne posent par contre aucun problème, étant donné la facilité avec laquelle elles peuvent être descendues.

3.2 Problème avec !

Considérons la preuve suivante :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash^{0} a,\ a^{\perp}}\,id}{\vdash^{0} a\wp a^{\perp}}\,\wp}{\vdash^{0} ?(a\wp a^{\perp})}\,?}{\vdash^{0} ?(a\wp a^{\perp}),\ ?b}\,w}{\vdash^{0} ?(a\wp a^{\perp}),\ !?b}\,!}{\vdash^{0} (?(a\wp a^{\perp}))\oplus b,\ !?b}\,\oplus_1}{\vdash^{0} ((?(a\wp a^{\perp}))\oplus b)\wp(!?b)}\,\wp$$

Il est impossible de rassembler les inférences qui contribuent à la production de la A-formule $!?b$ juste avant qu'elle ne devienne active au côté d'une B-formule dans la dernière inférence sans violer la condition relative au contexte de l'inférence !.

> *Donc nous laisserons aussi de côté les A-formules qui contiennent l'opérateur !.*

3.3 Descente des inférences

Mise à part les deux exceptions que nous venons de mentionner, il est possible de rassembler aussi bas que possible dans une preuve, les inférences qui contribuent à la production d'une A-formule entrant en combinaison avec une B-formule, ce qu'exprime le théorème suivant :

Théorème 4.3.1 *Soit F une formule de CLL ne contenant pas les opérateurs $\otimes$ et !. Dans CLL$\downarrow$, pour toute preuve $\mathcal{P}$ d'un séquent $\vdash^{a} F, \Delta$ où F est une A-formule, il existe une preuve qui a les propriétés suivantes :*

- elle a même conclusion que $\mathcal{P}$ et conserve pour celle-ci, la même partition entre A-formules et B-formules ;

-les inférences introduisant les sous-formules de F sont à la fin de la preuve (elles ne sont suivies par aucune autre).

Preuve 4.3.1 *Voir en annexe C* □

Ce théorème nous permet, avec certaines restrictions, de descendre dans une preuve les inférences contribuant à produire une A-formule pour les rassembler juste avant une inférence où cette formule se combine avec une B-formule. L'exemple traité dans la sous-section 3.2 de ce chapitre illustre cette possibilité.
L'étape suivante consiste à remplacer le bloc de ces inférences par une seule qui constitue un affaiblissement dans un sens étendu. La règle w dans sa version classique permet d'ajouter à tout séquent non affaiblissable une formule de la forme $\perp$ ou $?F$. L'étendre, voudrait dire qu'elle permette d'ajouter à tout séquent non affaiblissable, une A-formule. Mais nous avons vu que la notion de A-formule n'a de sens que

dans une preuve déjà constituée. Néanmoins ne peut-on pas déterminer a priori la forme syntaxique des A-formules utilisées pour prouver un séquent donné? C'est ce que nous allons examiner.

4 Base d'affaiblissement associée à un séquent.

Fixons un séquent $\vdash \Delta$ comme but à prouver. Les A-formules qui apparaissent dans une preuve sans coupures de $\vdash \Delta$ sont nécessairement des sous-formules des formules de $\vdash \Delta$. Les A-formules initiales sont celles qui sont engendrées par les inférences de type w. Si ces affaiblissements sont spécifiques, les formules engendrées sont quelconques. Mais s'ils sont classiques, elles sont de la forme $\perp$ ou $?F$. Ensuite, à partir de ces formules initiales, on peut construire d'autres A-formules par induction en appliquant les règles d'inférence logiques mais en excluant $\otimes$ et ! vu les problèmes qu'elles posent.
Nous pouvons ainsi délimiter un ensemble dont sont issues toutes les formules d'affaiblissement engendrées par des affaiblissements classiques : *la base d'affaiblissement associée à* $\vdash \Delta$ que nous noterons B_Δ. Celle-ci peut être définie rigoureusement ainsi :

Définition 4.4.1 *La base d'affaiblissement associée à un séquent $\vdash \Delta$ de CLL est l'ensemble B_Δ de sous-formules de formules de $\vdash \Delta$ construit inductivement ainsi :*

- il comprend déjà toutes celles de la forme $\perp$ ou $?F$;

- si deux formules F_1 et F_2 appartiennent à B_Δ, alors toute sous-formule de la forme $F_1 \wp F_2$ ou $F_1 \& F_2$ est membre de B_Δ ;

- si une formule F appartient à B_Δ, alors toute sous-formule de la forme $F \oplus G$, $G \oplus F$, $\exists x F$ ou $\forall x F$ est membre de B_Δ ;

Remarque 4.4.1 *La définition précédente pourrait être affinée car elle repose sur la notion de sous-formule qui permet n'importe quelle substitution à partir d'une formule quantifiée existentiellement alors qu'il est possible d'en prévoir un nombre fini parmi lesquelles se trouveront nécessairement celles qui sont utilisées dans les preuves de $\vdash \Delta$. Nous reviendrons sur cette question dans le prochain chapitre quand nous aborderons la construction descendante de preuves.*

On peut espérer réduire la base d'affaiblissement d'une autre façon : si deux formules F_1 et F_2 sont membres de cette base et si F_1 est une sous-formule de F_2, est-il vraiment nécessaire de garder F_1 dans la base? L'exemple suivant montre que la présence de & interdit une telle simplification.
Considérons la preuve :

$$
\begin{array}{c}
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash^0 A,\ A^\perp}\,id}{\vdash^0 ?A,\ A^\perp}\,? \qquad \dfrac{\dfrac{\dfrac{}{\vdash^0 B,\ B^\perp}\,id}{\vdash^0 B\wp B^\perp}\,\wp}{\vdash^0 B\wp B^\perp,\ ?A}\,w}{\vdash^0 A^\perp\&(B\wp B^\perp),\ ?A}\,\&}{\vdash^0 ?(A^\perp\&(B\wp B^\perp)),\ ?A}\,?}{\vdash^0 ?(A^\perp\&(B\wp B^\perp)),\ ?A,\ ?C^\perp}\,w}{\vdash^0 ?(A^\perp\&(B\wp B^\perp)),\ ?A,\ !?C^\perp}\,!}{\vdash^0 ?(A^\perp\&(B\wp B^\perp)),\ ?A\oplus C,\ !?C^\perp}\,\oplus_1
\end{array}
$$

Si l'on prend pour $\vdash \Delta$, la conclusion de cette preuve, la base d'affaiblissement associée B_Δ sera, d'après la définition qui vient d'être donnée, constituée des formules suivantes : $?(A^\perp \& (B \wp B^\perp))$, $?A$, $?A \oplus C$, $?C^\perp$. Bien que $?A$ soit une sous-formule de $?A \oplus C$, on ne peut pas supprimer la première de la base car le séquent $\vdash \Delta$ n'est pas prouvable sans un affaiblissement qui introduise $?A$, affaiblissement qui ne peut pas être suivi immédiatement par une inférence de type $\oplus_1$ introduisant $?A \oplus C$.
Cet exemple illustre le pouvoir des inférences de type & de transformer des formules d'affaiblissement qui ne sont pas actives en formules de base par superposition.

5 CLL$\downarrow_\Delta$: un système spécialisé pour prouver $\vdash \Delta$

A ce point, on peut étendre la règle d'affaiblissement w pour lui permettre d'ajouter aux séquents non affaiblissables, non seulement des formules de la forme $\perp$ ou $?F$ mais aussi n'importe quelle formule issue de la base d'affaiblissement B_Δ. Elle se présentera maintenant ainsi :

$$\frac{\vdash^a \Delta'}{\vdash^a F, \Delta'}\, w \qquad \textit{où, si } a = 0,\ F \textit{ est un élément de } B_\Delta$$

De cette façon, nous spécialisons aussi le système d'inférence dans la construction de preuves du seul séquent $\vdash \Delta$ puisque la règle w est maintenant liée à B_Δ. Ce nouveau cadre va nous permettre de descendre dans les preuves, les affaiblissements élargis et de limiter au maximum l'application des règles logiques à des séquents où au moins une formule active est une formule de base. Nous allons monter aussi les contractions au maximum.
Selon une technique que nous avions déjà utilisée pour CLL$\uparrow$, nous allons fusionner les affaiblissements avec les inférences qui suivent immédiatement et les contractions avec celles qui précèdent immédiatement. Nous obtiendrons un système qui n'aura plus ni règle de contraction, ni règle d'affaiblissement explicites mais où celles-ci auront été intégrées dans les règles logiques qui auront été redéfinies.

5.1 Deux opérations particulières sur les multi-ensembles de formules

La redéfinition des règles $\otimes$ et & nécessite l'introduction de deux opérations sur les multi-ensembles de formules : $+$ et $\times$. La première $+$ va permettre dans l'application de la règle $\otimes$ de juxtaposer les contextes initiaux tout en contractant les formules de la forme $?F$ qui vont se trouvées ainsi dupliquées. Elle va permettre aussi dans l'application de la règle ? de contracter la formule principale obtenue avec une autre présente éventuellement dans le contexte. D'où la définition :

Définition 4.5.1 *Soit Δ_1 et Δ_2 deux multi-ensembles de formules de logique linéaire. La somme $\Delta_1 + \Delta_2$ est obtenue en prenant toute formule de Δ_1 ou Δ_2 et en calculant son ordre de multiplicité de la façon suivante : si cette formule est de la forme ?F, il sera égal à 1 sinon on additionne la valeur qu'il prend dans chacun des deux multi-ensembles de départ.*

Par exemple, nous aurons : $\{a, a, ?b, ?b\} + \{a, \perp, ?b\} = \{a, a, a, \perp, ?b\}$. Si un des multi-ensembles se réduit à un singleton, nous le confondrons par abus de langage avec son seul élément. Ainsi, nous pourrons écrire : $?a + \{b, ?a\} = \{?a, b\}$.
La seconde opération, $\times$, va permettre dans l'application de la règle & de superposer les contextes initiaux en les affaiblissant au préalable pour les faire coïncider. Bien entendu, les possibilités d'affaiblissement vont dépendre du caractère affaiblissable ou non des séquents dont font partie ces contextes. C'est pourquoi à chaque multi-ensemble de formule sur lequel porte l'opération, on associe un booléen indiquant si ce multi-ensemble est intégré dans un séquent affaiblissable ou non.

Définition 4.5.2 *Soit Δ_1 et Δ_2 deux multi-ensembles de formules de CLL et soit a_1 et a_2 deux booléens. Supposons que la condition suivante est respectée :*
si $a_i = 0$ pour i=1 ou i=2, alors toute formule de Δ_{3-i} qui n'appartient pas à B_Δ, est présente dans Δ_i avec un ordre de multiplicité supérieur ou égal.
Alors le produit $(\Delta_1, a_1) \times (\Delta_2, a_2)$ est un multi-ensemble constitué des formules de Δ_1 ou Δ_2 avec le plus grand des ordres de multiplicité qu'elles y ont.

Par exemple, le produit $(\{a,\ b,\ ?c\}\ ,0)\ \times\ (\{a,\ a,\ ?d\},\ 1)$ n'est pas défini et cela provient uniquement du fait que la formule a a un ordre de multiplicité de 1 dans le premier multi-ensemble qui n'est pas affaiblissable alors que cet ordre de multiplicité est de 2 dans le deuxième multi-ensemble.
Par contre, le produit $(\{a,\ a,\ b,\ ?c\}\ ,0)\ \times\ (\{a,\ ?d\},\ 1)$ est défini et vaut $\{a,\ a,\ b,\ ?c,\ ?d\}$.

5.2 La définition de $CLL{\downarrow}_\Delta$

A l'aide des deux opérations qui viennent d'être établies, nous allons pouvoir redéfinir les règles logiques de CLL$\downarrow$ pour y intégrer la règle $c?$ et w étendue.
Selon le nombre d'affaiblissements qui seront pris en compte dans une règle logique, celle-ci va se transformer en 1, 2, 3 ou même 4 nouvelles règles (voir par exemple $\otimes$). Nous obtenons alors le système $CLL{\downarrow}_\Delta$, décrit par la figure 4.2.
Dans ce système, il ne reste maintenant plus que deux règles difficiles à contrôler dans une construction descendante de preuves de $\vdash \Delta$: $\otimes_{w12}$ et $!_w$; ce sont les seules qui ne font intervenir aucune formule active. Le progrès par rapport à CLL$\downarrow$ où la règle d'affaiblissement w pouvait être appliquée librement, est donc considérable. On constate aussi qu'on obtient un meilleur contrôle des affaiblissements que dans le système proposé par [Tammet, 1993]: dans ce système subsistent un nombre plus important de règles s'appliquant à des prémisses sans formule active. C'est normal car $CLL{\downarrow}_\Delta$ est le résultat d'une descente maximum des inférences produisant des A-formules alors que le système proposé par Tammet est le résultat de la descente des seuls affaiblissements.
Pour que la démonstration soit complète, il faut établir l'équivalence entre $CLL{\downarrow}_\Delta$ et CLL$\downarrow$. Commençons par la correction.

Théorème 4.5.1 *Tout séquent $\overset{a}{\vdash}\Delta'$ prouvable dans $CLL{\downarrow}_\Delta$ est prouvable dans CLL$\downarrow$.*

Preuve 4.5.1 *Nous pouvons procéder en deux étapes. Nous pouvons montrer tout d'abord que la règle w étendue est admissible dans CLL$\downarrow$ par induction sur la structure de la formule introduite par cette règle. Ensuite il est facile de montrer que toute règle logique de $CLL{\downarrow}_\Delta$ est admissible dans CLL$\downarrow$ en la remplaçant par une règle logique de CLL$\downarrow$ accompagnée éventuellement d'affaiblissements et de contractions.*□

La complétude du nouveau système relativement à CLL$\downarrow$ est bien entendu restreinte aux preuves du séquent $\vdash \Delta$ en fonction duquel il a été construit. D'où le théorème:

Théorème 4.5.2 *Pour toute conclusion intermédiaire $\overset{a'}{\vdash}\Delta'$ d'une preuve de $\overset{a}{\vdash}\Delta$ dans CLL$\downarrow$, il existe une preuve d'un séquent $\overset{a'}{\vdash}\Gamma$ dans $CLL{\downarrow}_\Delta$ telle que Δ' est obtenu en complétant Γ par des formules qui, si a'=0, appartiennent à B_Δ. Nous dirons que le séquent $\overset{a'}{\vdash}\Gamma$ subsume $\overset{a'}{\vdash}\Delta'$.*

Preuve 4.5.2 *Soit une preuve quelconque $\mathcal{P}$ de $\overset{a}{\vdash}\Delta$. Soit $\overset{a'}{\vdash}\Delta'$ une conclusion intermédiaire quelconque de $\mathcal{P}$ et $\mathcal{P}'$ la preuve extraite de $\mathcal{P}$ qui a comme conclusion $\overset{a'}{\vdash}\Delta'$. Nous allons démontrer par induction sur la structure de $\mathcal{P}'$ qu'il existe un séquent $\overset{a'}{\vdash}\Gamma$ qui est prouvable dans $CLL{\downarrow}_\Delta$ et qui subsume $\overset{a'}{\vdash}\Delta'$. Soit I la dernière inférence de $\mathcal{P}'$. Nous faire une analyse de cas selon le type de I.*

1. I est un axiome.
   Etant donné que les axiomes des deux systèmes sont les mêmes, le résultat est immédiat.

2. I est de type $\otimes$.
   I a alors la forme :

$$\frac{\overset{a_1}{\vdash}F_1, \Delta'_1 \quad \overset{a_2}{\vdash}F_2, \Delta'_2}{\overset{a_1 \vee a_2}{\vdash}F_1 \otimes F_2, \Delta'_1, \Delta'_2}\ \otimes$$

   Par hypothèse d'induction, il existe deux séquents $\overset{a_1}{\vdash}\Gamma_1$ et $\overset{a_2}{\vdash}\Gamma_2$ prouvables dans $CLL{\downarrow}_\Delta$ et subsumant respectivement les séquents $\overset{a_1}{\vdash}F_1, \Delta'_1$ et $\overset{a_2}{\vdash}F_2, \Delta'_2$.

FIG. 4.2 - *Le système d'inférence de $CLL{\downarrow}_\Delta$*
Pour toute règle ci-dessous, une composante d'une formule principale non présente dans un prémisse, est nécessairement un élément de B_Δ.

groupe identité

$$\frac{}{\vdash^{0} A, A^{\perp}}\ \mathrm{id}$$

groupe logique

négation

$$F^{\perp\perp} = F$$

$$(F \otimes G)^{\perp} = F^{\perp} \wp G^{\perp} \qquad (F \wp G)^{\perp} = F^{\perp} \otimes G^{\perp}$$

$$1^{\perp} = \perp \qquad \perp^{\perp} = 1$$

$$(F \& G)^{\perp} = F^{\perp} \oplus G^{\perp} \qquad (F \oplus G)^{\perp} = F^{\perp} \& G^{\perp}$$

$$\top^{\perp} = 0 \qquad 0^{\perp} = \top$$

$$(!F)^{\perp} = ?F^{\perp} \qquad (?F)^{\perp} = !F^{\perp}$$

$$(\forall x F)^{\perp} = \exists x F^{\perp} \qquad (\exists x F)^{\perp} = \forall x F^{\perp}$$

opérateurs multiplicatifs

$$\frac{\vdash^{a_1} F_1, \Delta_1 \quad \vdash^{a_2} F_2, \Delta_2}{\vdash^{a_1 \vee a_2} F_1 \otimes F_2, \Delta_1 + \Delta_2}\ \otimes \qquad \frac{\vdash^{0} \Delta_1 \quad \vdash^{a_2} F_2, \Delta_2}{\vdash^{a_2} F_1 \otimes F_2, \Delta_1 + \Delta_2}\ \otimes_{w1} \qquad \frac{\vdash^{a_1} F_1, \Delta_1 \quad \vdash^{0} \Delta_2}{\vdash^{a_1} F_1 \otimes F_2, \Delta_1 + \Delta_2}\ \otimes_{w2} \qquad \frac{\vdash^{0} \Delta_1 \quad \vdash^{0} \Delta_2}{\vdash^{0} F_1 \otimes F_2, \Delta_1 + \Delta_2}\ \otimes_{w12}$$

$$\frac{}{\vdash^{0} 1}\ 1 \qquad \frac{\vdash^{a} F_1, F_2, \Delta}{\vdash^{a} F_1 \wp F_2, \Delta}\ \wp \qquad \frac{\vdash^{a} F_2, \Delta}{\vdash^{a} F_1 \wp F_2, \Delta}\ \wp_{w1} \qquad \frac{\vdash^{a} F_1, \Delta}{\vdash^{a} F_1 \wp F_2, \Delta}\ \wp_{w2}$$

opérateurs additifs

$$\frac{\vdash^{a_1} F_1, \Delta_1 \quad \vdash^{a_2} F_2, \Delta_2}{\vdash^{a_1 \wedge a_2} F_1 \& F_2, (\Delta_1, a_1) \times (\Delta_2, a_2)}\ \& \qquad \frac{\vdash^{0} \Delta_1 \quad \vdash^{a_2} F_2, \Delta_2}{\vdash^{0} F_1 \& F_2, (\Delta_1, 0) \times (\Delta_2, a_2)}\ \&_{w1} \qquad \frac{\vdash^{a_1} F_1, \Delta_1 \quad \vdash^{0} \Delta_2}{\vdash^{0} F_1 \& F_2, (\Delta_1, a_1) \times (\Delta_2, 0)}\ \&_{w2}$$

$$\frac{}{\vdash^{1} \top}\ \top' \qquad \frac{\vdash^{a} F_1, \Delta}{\vdash^{a} F_1 \oplus F_2, \Delta}\ \oplus_1 \qquad \frac{\vdash^{a} F_2, \Delta}{\vdash^{a} F_1 \oplus F_2, \Delta}\ \oplus_2$$

opérateurs exponentiels

$$\frac{\vdash^{a} F, ?\Delta}{\vdash^{0} !F, ?\Delta}\ ! \qquad \frac{\vdash^{a} ?\Delta}{\vdash^{0} !F, ?\Delta}\ !_w \qquad \frac{\vdash^{a} F, \Delta}{\vdash^{a} ?F + \Delta}\ ?$$

quantificateurs

$$\frac{\vdash^{a} F[y/x], \Delta}{\vdash^{a} \forall x F, \Delta}\ \forall \qquad \frac{\vdash^{a} F[t/x], \Delta}{\vdash^{a} \exists x F, \Delta}\ \exists$$

avec y non libre dans la conclusion

Selon la présence ou non des formules F_1 et F_2 dans Γ_1 et Γ_2 respectivement, nous avons 4 situations possibles.

(a) $\Gamma_1 = F_1, \Gamma_{11}$ **et** $\Delta'_2 = F_2, \Gamma_{22}$.
Dans $CLL{\downarrow}_\Delta$, nous pouvons construire l'inférence suivante :

$$\frac{\vdash^{a_1} F_1, \Gamma_{11} \quad \vdash^{a_2} F_2, \Gamma_{22}}{\vdash^{a_1 \vee a_2} F_1 \otimes F_2, \Gamma_{11} + \Gamma_{22}} \otimes$$

Il est ensuite aisé de prouver que $\vdash^{a_1 \vee a_2} F_1 \otimes F_2, \Gamma_{11} + \Gamma_{22}$ subsume $\vdash^{a_1 \vee a_2} F_1 \otimes F_2, \Delta_1, \Delta_2$ en s'appuyant sur l'hypothèse d'induction et sur le fait que le booléen $a_1 \vee a_2$ est supérieur ou égal à a_1 et à a_2.

(b) $\Gamma_1 = F_1, \Gamma_{11}$ **et** $F_2 \notin \Gamma_2$.
Si $a_2 = 1$, $\vdash^{a_2} \Gamma_2$ subsume $\vdash^{a_1 \vee a_2} F_1 \otimes F_2, \Delta'_1, \Delta'_2$,.
Si $a_2 = 0$, par hypothèse d'induction, F_2 appartient à B_Δ. Donc, dans $CLL{\downarrow}_\Delta$, nous pouvons construire l'inférence suivante :

$$\frac{\vdash^{a_1} F_1, \Gamma_{11} \quad \vdash^{a_2} \Gamma_2}{\vdash^{a_1 \vee a_2} F_1 \otimes F_2, \Gamma_{11} + \Gamma_2} \otimes_{w2}$$

Comme dans le cas précédent, il est ensuite facile de montrer que $\vdash^{a_1 \vee a_2} F_1 \otimes F_2, \Gamma_{11} + \Gamma_2$ subsume $\vdash^{a_1 \vee a_2} F_1 \otimes F_2, \Delta'_1, \Delta'_2$.
Les deux autres situations se traitent de manière analogue en utilisant les règles $\otimes_{w1}$ et $\otimes_{w12}$ de $CLL{\downarrow}_\Delta$.

3. I est de type $\wp$.
I a alors la forme :

$$\frac{\vdash^{a'} F_1, F_2, \Delta'_1}{\vdash^{a'} F_1 \wp F_2, \Delta'_1} \wp$$

Par hypothèse d'induction, nous pouvons trouver un séquent $\vdash^{a'} \Gamma$ prouvable dans $CLL{\downarrow}_\Delta$ et subsumant le séquent $\vdash^{a'} F_1, F_2, \Delta'_1$.
Selon la présence ou non des formules F_1 et F_2 dans Γ, nous nous trouvons devant 4 situations possibles.

(a) $\Gamma = F_1, F_2, \Gamma_1$.
Dans $CLL{\downarrow}_\Delta$, nous pouvons construire l'inférence suivante :

$$\frac{\vdash^{a'} F_1, F_2, \Gamma_1}{\vdash^{a'} F_1 \wp F_2, \Gamma_1} \wp$$

En utilisant l'hypothèse d'induction, nous pouvons déduire que $\vdash^{a'} F_1 \wp F_2, \Gamma_1$ subsume $\vdash^{a'} F_1 \wp F_2, \Gamma_1$.

(b) $\Gamma = F_1, \Gamma_1$ **et** $F_2 \notin \Gamma_1$.
Si a" = 0, par hypothèse d'induction, F_2 appartient nécessairement à B_Δ. Par conséquent, dans $CLL{\downarrow}_\Delta$, nous pouvons construire l'inférence suivante :

$$\frac{\vdash^{a'} F_1, \Gamma_1}{\vdash^{a'} F_1 \wp F_2, \Gamma_1} \wp_{w2}$$

En utilisant l'hypothèse d'induction, il est ensuite facile de montrer que $\vdash^{a'} F_1 \wp F_2, \Gamma_1$ *subsume* $\vdash^{a'} F_1 \wp F_2, \Delta'_1$.
Le cas symétrique où F_1 *n'appartient pas à* Γ *et où* F_2 *en est élément, se traite de façon analogue en utilisant cette fois la règle* $\wp_{w1}$.

(c) $F_1 \notin \Gamma$ et $F_2 \notin \Gamma$.
Si $a' = 0$, *par hypothèse d'induction,* F_1 *et* F_2 *appartiennent nécessairement à* B_Δ *donc* $F_1 \wp F_2$ *aussi. Par conséquent,* $\vdash^{a'} \Gamma$ *subsume* $\vdash^{a'} F_1 \wp F_2, \Delta'_1$. *Si* $a' = 1$, *c'est immédiat.*

4. I est de type &.
I a alors la forme :

$$\frac{\vdash^{a_1} F_1, \Delta'_1 \quad \vdash^{a_2} F_2, \Delta'_1}{\vdash^{a_1 \wedge a_2} F_1 \& F_2, \Delta'_1} \&$$

Par hypothèse d'induction, il existe deux séquents $\vdash^{a_1} \Gamma_1$ *et* $\vdash^{a_2} \Gamma_2$ *prouvables dans* $CLL{\downarrow}_\Delta$ *et subsumant respectivement les séquents* $\vdash^{a_1} F_1, \Delta'_1$ *et* $\vdash^{a_2} F_2, \Delta'_1$.
Selon la présence ou non des formules F_1 *et* F_2 *dans* Γ_1 *et* Γ_2 *respectivement, nous avons 4 situations possibles.*

(a) $\Gamma_1 = F_1, \Gamma_{11}$ et $\Gamma_2 = F_2, \Gamma_{12}$.
Dans $CLL{\downarrow}_\Delta$, *nous pouvons construire l'inférence suivante :*

$$\frac{\vdash^{a_1} F_1, \Gamma_{11} \quad \vdash^{a_2} F_2, \Gamma_{12}}{\vdash^{a_1 \wedge a_2} F_1 \& F_2, (\Gamma_{11}, a_1) \times (\Gamma_{12}, a_2)} \&$$

Il reste ensuite à montrer que $\vdash^{a_1 \wedge a_2} F_1 \& F_2, (\Gamma_{11}, a_1) \times (\Gamma_{12}, a_2)$ *subsume* $\vdash^{a_1 \wedge a_2} F_1 \& F_2, \Delta'_1$.
Cela se fait aisément en utilisant les hypothèses d'induction et les propriétés du produit $\times$ *de multi-ensembles.*

(b) $F_1 \notin \Gamma_1$ et $F_1 \not\in \Gamma_1$.
Supposons que $a_1 \wedge a_2 = a_1$. *Par hypothèse d'induction,* $\vdash^{a_1} \Gamma_1$ *subsume* $\vdash^{a_1} F_1, \Delta'_1$ *et comme* $F_1 \not\in \Gamma_1$, $\vdash^{a_1} \Gamma_1$ *subsume* $\vdash^{a_1} \Delta'_1$ *et* $F_1 \in B_\Delta$. *On montre de la même façon que* $F_2 \in B_\Delta$. *Par conséquent,* $F_1 \& F_2$ *appartient à* B_Δ *aussi. Finalement on peut conclure que* $\vdash^{a_1} \Gamma_1$ *subsume* $\vdash^{a_1} F_1 \& F_2, \Delta'_1$ *c'est-à-dire* $\vdash^{a_1 \wedge a_2} F_1 \& F_2, \Delta'_1$ *car* $a_1 \wedge a_2 = a_1$.

Les deux autres cas se traitent de la même façon que le premier en utilisant les règles $\&_{w1}$ *et* $\&_{w2}$ *de* $CLL{\downarrow}_\Delta$.

5. I est de type c?
I se présente alors ainsi :

$$\frac{\vdash^{a'} ?F, ?F, \Delta'_1}{\vdash^{a'} ?F, \Delta'_1} c?$$

Par hypothèse d'induction, il existe un séquent $\vdash^{a'} \Gamma$ *qui est prouvable dans* $CLL{\downarrow}_\Delta$ *et qui subsume* $\vdash^{a'} ?F, ?F, \Delta'_1$.
Il est facile de montrer que tout séquent prouvable dans $CLL{\downarrow}_\Delta$ *ne contient pas deux formules identiques avec ? comme connecteur de tête. Donc* $\vdash^{a'} \Gamma$ *contient au maximum un exemplaire de* $?F$ *ce qui montre qu'il subsume* $\vdash^{a'} ?F, \Delta'_1$.

Nous laisserons le soin au lecteur de traiter les autres cas qui ne posent pas de problème.□

Les affaiblissements et les contractions ayant été intégrés dans les inférences logiques, il ne reste maintenant plus qu'à ordonner ces dernières dans les preuves de CLL$\downarrow_\Delta$ pour obtenir des preuves normales.

6 Preuves normales dans CLL$\downarrow_\Delta$

6.1 Permutabilité d'inférences et subsomption

La normalisation des preuves dans CLL$\downarrow_\Delta$ exige au préalable que nous transposions les résultats sur la permutabilité d'inférences dans CLL$\downarrow$ (voir à ce sujet le tableau 4.1) dans CLL$\downarrow_\Delta$.
Cette tâche ne présente pas de difficultés mais elle est rendue fastidieuse par la forme complexe des règles $\otimes$ et & dans CLL$\downarrow_\Delta$. C'est pourquoi nous ne la présenterons pas ici. On peut dire seulement que la mobilité des inférences est quelque peu diminuée du fait de l'intégration des contractions et des affaiblissements dans les règles logiques. D'autre part, il est utile d'étendre légèrement la définition de la permutabilité de deux inférences en y intégrant la notion de subsomption introduite par [Tammet, 1993].
Considérons par exemple la déduction :

$$\dfrac{\dfrac{\vdash^{a} A,\ ?B}{\vdash^{a} A,\ ?B \oplus C}\oplus_1 \qquad \vdash^{b} D,\ ?B}{\vdash^{a \vee b} A \otimes D,\ ?B \oplus C,\ ?B}\otimes$$

Si l'on cherche à permuter l'inférence $\oplus_1$ avec $\otimes$, on devra obtenir la déduction :

$$\dfrac{\dfrac{\vdash^{a} A,\ ?B \quad \vdash^{b} D,\ ?B}{\vdash^{a \vee b} A \otimes D,\ ?B}\otimes}{\vdash^{a \vee b} A \otimes D,\ ?B \oplus C}\oplus_1$$

La conclusion du résultat de la permutation n'est plus identique mais subsume seulement celle de l'objet de la permutation. Mais nous dirons quand même que les inférences sont permutables.
Malgré ces différences, la classification entre inférences faciles à monter et inférences faciles à descendre est la transposition de celle obtenue dans CLL$\downarrow$ avec une exception : les types $\wp_{w1}$ et $\wp_{w2}$ se rangent dans $T\downarrow$. D'où :

$$\boxed{\begin{array}{c} T\uparrow = \{\otimes,\ \otimes_{w1},\ \otimes_{w2}\ \wp,\ \oplus,\ \exists,\ ?\} \\ T\downarrow = \{\wp_{w1},\ \wp_{w2},\ \&,\ \&_{w1},\ \&_{w2},\ \forall\} \end{array}}$$

6.2 Preuves normales

Les preuves normales de CLL$\uparrow$ se définissaient par un lien particulier entre chaque conclusion intermédiaire et l'inférence la produisant. Cela correspondait au fait que nous nous situions dans une perspective de construction ascendante des preuves.
Maintenant que nous nous situons dans une perspective opposée, il n'est pas étonnant que nous définissions une preuve normale en précisant *le lien entre chacune de ses conclusions intermédiaires et l'inférence dont elle est un prémisse.*

Définition 4.6.1 *Une preuve de CLL$\downarrow_\Delta$ est dite normale si toutes ses conclusions intermédiaires sont normales.*
Une conclusion intermédiaire normale est caractérisée ainsi :

Si *elle contient la partie active d'un prémisse d'une inférence d'un type appartenant à* $T\uparrow$

et si dans cette partie active, lorsque l'inférence n'est pas de type $\wp$,
il y a au moins une formule qui n'est pas un littéral négatif et qui n'a pas? comme connecteur de tête
alors *elle est le prémisse d'une inférence de ce type*
sinon si *elle est prémisse d'une inférence d'un type appartenant à $T\downarrow$*
alors*la formule principale de cette inférence,*
si elle est active dans une inférence qui n'est pas de type $\wp$,
l'est dans celle qui suit immédiatement.

Remarque 4.6.1 *On pourrait pousser plus loin la normalisation des preuves en montant au maximum les inférences de type $\otimes_{w12}$ mais ce mouvement n'est pas simple et cela alourdirait beaucoup la définition ci-dessus.*

Illustrons cette définition par un exemple.

Exemple 4.6.1 *Considérons la preuve de $CLL\downarrow_{\Delta}$ suivante :*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash^0 a(x),\ a(x)^\perp}\,id}{\vdash^0 a(x),\ ?(a(x)^\perp)}\,?}{\vdash^0 a(x),\ (?(a(x)^\perp))\wp?b(x)}\,\wp_{w2} \quad \dfrac{}{\vdash^0 1}\,1}{\vdash^0 a(x)\&1,\ (?(a(x)^\perp))\wp?b(x)}\,\& \quad \dfrac{}{\vdash^0 1}\,1}{\vdash^0 a(x)\&1,\ ((?(a(x)^\perp))\wp?b(x))\otimes?(b(x)^\perp),\ 1}\,\otimes_{w2}}{\vdash^0 a(x)\&1,\ \exists x(((?(a(x)^\perp))\wp?b(x))\otimes?(b(x)^\perp)),\ 1}\,\exists}{\vdash^0 \forall x(a(x)\&1),\ \exists x(((?(a(x)^\perp))\wp?b(x))\otimes?(b(x)^\perp)),\ 1}\,\forall$$

Cette preuve n'est pas normale car la conclusion intermédiaire $\vdash^0 a(x),\ (?(a(x)^\perp))\wp?b(x)$ n'est pas normale à double titre :

- $(?(a(x)^\perp))\wp?b(x)$ constitue la partie active d'une inférence de type $\otimes_{w2}$ alors que l'inférence qui suit est de type & qui n'appartient pas à $T\uparrow$;

- la formule principale $a(x)\&1$ de l'inférence de type & qui suit immédiatement n'est pas active dans celle d'après mais seulement dans la dernière de la preuve.

En montant les inférences $\otimes_{w2}$ et $\exists$ et en insérant une nouvelle inférence $\otimes_{w12}$, nous obtenons la preuve normale suivante :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash^0 a(x),\ a(x)^\perp}\,id}{\vdash^0 a(x),\ ?(a(x)^\perp)}\,?}{\vdash^0 a(x),\ (?(a(x)^\perp))\wp?b(x)}\,\wp_{w2} \quad \dfrac{}{\vdash^0 1}\,1}{\vdash^0 a(x),\ ((?(a(x)^\perp))\wp?b(x))\otimes?(b(x)^\perp),\ 1}\,\otimes_{w2}}{\vdash^0 (a(x),\ \exists x(((?(a(x)^\perp))\wp?b(x))\otimes?(b(x)^\perp)),\ 1}\,\exists \quad \dfrac{\dfrac{}{\vdash^0 1}\,1 \quad \dfrac{}{\vdash^0 1}\,1}{\vdash^0 \exists x(((?(a(x)^\perp))\wp?b(x))\otimes?(b(x)^\perp)),\ 1,\ 1}\,\otimes_{w12}}{\vdash^0 a(x)\&1,\ \exists x(((?(a(x)^\perp))\wp?b(x))\otimes?(b(x)^\perp)),\ 1}\,\&}{\vdash^0 \forall x(a(x)\&1),\ \exists x(((?(a(x)^\perp))\wp?b(x))\otimes?(b(x)^\perp)),\ 1}\,\forall$$

6.3 Théorème de normalisation

La généralisation du processus de normalisation présenté dans l'exemple ci-dessus se traduit par le théorème suivant.

Théorème 4.6.1 *Tout séquent prouvable dans $CLL{\downarrow}_\Delta$ admet une preuve normale dans $CLL{\downarrow}_\Delta$.*

Preuve 4.6.1 *Soit une preuve $\mathcal{P}$ quelconque de $CLL{\downarrow}_\Delta$. Nous allons montrer par induction sur la structure de $\mathcal{P}$ qu'il existe dans $CLL{\downarrow}_\Delta$, une preuve normale qui a même conclusion que $\mathcal{P}$.*
Soit I la dernière inférence de $\mathcal{P}$. Par hypothèse d'induction, nous pouvons remplacer toutes les preuves des prémisses de I extraites de $\mathcal{P}$ par des preuves normales. Nous obtenons une nouvelle preuve $\mathcal{P}'$ qui n'est pas nécessairement normale : l'introduction de I peut faire perdre à certaines conclusions intermédiaires le caractère normal qu'elles avaient auparavant.
Distinguons deux cas selon le type de I.

1. le type de I appartient à $T\uparrow$.
   Cela peut entraîner certaines conclusions intermédiaires à ne plus être normales pour deux raisons :

   - *elles contiennent la partie active d'un prémisse de I qui a au moins une formule qui n'est pas un littéral négatif et qui n'a pas ? comme connecteur de tête et sont prémisses d'une inférence d'un type hors de $T\uparrow$;*
   - *elles sont prémisses d'une inférence d'un type appartenant à $T\downarrow$ et la formule principale de cette inférence n'est pas active dans celle qui suit immédiatement, mais dans l'inférence I.*

   Dans les branches où apparaissent de telles anomalies, il suffit de monter I jusqu'aux inférences introduisant une de ses formules actives pour les faire disparaître. Ce mouvement est possible à cause des propriétés de permutabilité des inférences dont le type appartient à $T\uparrow$ (il nécessite parfois, comme on l'a vu dans l'exemple précédent, l'insertion de nouvelles inférences lors du franchissement d'inférences de type &).

2. le type de I n'appartient pas à $T\uparrow$.
   Si I n'a pas de formules actives (elle est alors de type $\otimes_{w12}$ ou $!_w$), la preuve $\mathcal{P}'$ est normale.
   Sinon, la conclusion intermédiaire qui est prémisse d'une inférence I' d'un type appartenant à $T\downarrow$ ayant une formule principale qui n'est pas active immédiatement mais seulement dans l'inférence I.
   Pour corriger l'anomalie, il va falloir descendre I' jusqu'à I mais cela ne suffit pas car cette descente peut avoir pour effet de détruire le caractère normal d'un prémisse d'une inférence qui se situait juste avant I' et qui était aussi d'un type appartenant à $T\downarrow$.
   Pour éviter cet inconvénient, nous descendons tout un bloc d'inférences de type appartenant à $T\downarrow$.

□

Le théorème de normalisation peut être complété par un autre qui montre que, lorqu'une conclusion intermédiaire contient des formules qui vont être ensuite actives dans des inférences d'un type appartenant à $T\uparrow$, l'ordre dans lequel ces inférences sont effectuées, n'a pas d'importance.

Théorème 4.6.2 *Soit $\overset{a'}{\vdash}\Delta_a,\ \Delta'$ une conclusion intermédiaire d'une preuve normale $\mathcal{P}$ de $\overset{a}{\vdash}\Delta$.*
Supposons que dans la preuve $\mathcal{P}$, cette conclusion intermédiaire est suivie d'une autre qui est le prémisse d'une inférence I d'un type appartenant à $T\uparrow$ et qui a Δ_a pour partie active.
Supposons aussi que, si I n'est pas de type $\wp$, il y a au moins une formule de Δ_a qui n'est pas un littéral négatif et qui n'est pas de la forme $?F$.
Alors il existe une preuve normale de $\overset{a}{\vdash}\Delta$ où $\overset{a'}{\vdash}\Delta_a,\ \Delta'$ est le prémisse d'une inférence avec Δ_a comme partie active.

Preuve 4.6.2 *Il suffit de monter dans la preuve de départ l'inférence dont Δ_a est une partie active juste après l'inférence qui produit $\overset{a'}{\vdash}\Delta_a,\ \Delta'$.*□

Conclusion

Il est intéressant de comparer les résultats obtenus ici avec ceux de [Tammet, 1993]. Il faut pour cela, dépasser une différence de présentation qui provient du fait que T. Tammet utilise le formalisme de la résolution alors que nous restons dans le cadre du calcul des séquents.
Cela nous permet d'utiliser *directement* les propriétés liées au formalisme du calcul des séquents, notamment celles reposant sur la permutabilité d'inférences, sans être obligé de passer par la médiation d'une transposition.
Sur le fond, nous avons, comme T. Tammet, bâti notre méthode pour normaliser les preuves, sur la permutabilité d'inférences. Mais le fait de l'ériger en système, nous a permis non seulement de retrouver les principaux résultats obtenus par T. Tammet (séquents affaiblissables, descente des affaiblissements, mouvement des inférences logiques) mais aussi d'aller plus loin.
Notamment, nous avons pu résoudre de manière satisfaisante le problème crucial du contrôle des affaiblissements dans une construction descendante des preuves : partant de la distinction au sein d'une preuve entre formules de base et formules d'affaiblissement, nous avons montré qu'il ne suffisait pas de descendre au maximum dans une preuve les affaiblissements pour mieux les contrôler mais qu'il fallait englober dans ce mouvement toutes les inférences contribuant à produire les formules d'affaiblissement. De là, l'extension de la règle d'affaiblissement, la notion de base d'affaiblissement et la spécialisation du système d'inférence pour l'adapter à la construction descendante de preuves d'un séquent particulier.
Après ces quelques remarques, la comparaison entre les méthodes utilisant le formalisme du calcul des séquents et celles basées sur la résolution est cependant loin d'être épuisée. Il faut notamment prendre en compte les travaux de [Mints, 1993] qui ont permis de simplifier le système de résolution de Tammet tout en l'étendant au premier ordre.

Chapitre 5

Stratégies de recherche de preuves

Introduction

Dans les chapitre 3 et 4, nous avons utilisé la permutabilité d'inférences pour normaliser au maximum les preuves dans CLL. L' objectif était de définir une classe de preuves normales qui soit à la fois complète relativement au fragment considéré et la plus restreinte possible. Cela doit permettre de réduire au maximum l'indéterminisme dans la construction des preuves, ce qui est notamment important pour la *démonstration automatique en logique linéaire.*
Ainsi, la montée maximum de certaines inférences au cours du processus de normalisation, va donner lieu aux principes duaux de *composition immédiate* dans la construction de preuves en avant et de *décomposition en chaîne* dans la construction de preuves en arrière.
Symétriquement, la descente maximum d'autres inférences va se traduire par les principes duaux de *composition en chaîne* dans la construction de preuves en arrière et de *décomposition immédiate* dans la construction de preuves en avant [Galmiche and Perrier, 1994a].

- Dans ce chapitre, nous commencerons par présenter ces principes.

- Ensuite, nous allons étudier en détail les stratégies de recherche de preuves en arrière [Andreoli, 1992]. La construction de preuves de bas en haut s'organise autour de l'édification d'un arbre de preuve incomplet que nous appellerons *l'arbre de recherche.*
  L'indéterminisme dans la construction se traduit par le phénomène de *retour-arrière dans la construction de l'arbre de recherche.* Après avoir mis en évidence ses facteurs, nous montrerons que les principes de *décomposition immédiate et en chaîne* permettent de réduire certains d'entre eux. Pour d'autres, nous ferons appel à des techniques qui ne sont plus basées sur la permutabilité d'inférences : utilisation d'une forme de *neutralité logique globale des séquents prouvables* et *report de certain choix* pour les règles $\exists$ [Shankar, 1992; Lincoln and Shankar, 1994] et $\otimes$ [Hodas and Miller, 1991, 1994].

- Après, nous aborderons les stratégies de recherche de preuves en avant [Tammet, 1993; Mints, 1993] qui constituent l'aspect le plus original de ce chapitre. La construction de preuves de haut en bas s'organise autour de l'édification d'un ensemble de séquents sensés représenter à un moment donné les conclusions intermédiaires déjà établies, les plus proches du but. Nous appellerons cet ensemble *la base de recherche.*
  L'indéterminisme ne se traduira pas par un retour-arrière mais par une *extension de la base de recherche* : quand à partir de l'une de ses conclusions intermédiaires, nous savons qu'il n'est possible d'en inférer qu'une, nous remplacerons dans l'ensemble l'ancienne par la nouvelle et la taille de la base ne variera pas ; dans le cas contraire, la nouvelle conclusion viendra s'ajouter à l'ancienne, ce qui augmentera la taille de la base. Ce qui fait qu'à terme, un certain nombre de ces conclusions intermédiaires va se révéler inutile.
  Après avoir analysé les différentes sources d'indéterminisme, nous montrerons comment l'application des *principes de composition immédiate et en chaîne* vont permettre de réduire certaines d'entre

elles.
Comme pour la construction ascendante des preuves, les substitutions liées aux quantification du premier ordre, seront engendrées au niveau des axiomes par le biais de *l'unification*. Mais le changement de sens de construction des preuves, va donner au processus un tour original : les substitutions ne vont pas être effectuées au moment de leur génération mais laissées en attente jusqu'à l'introduction des quantificateurs dans les clauses.
Enfin, pour limiter la taille de la base de recherche, nous aurons recours à deux techniques : la suppression des conclusions intermédiaires *subsumées* par d'autres et de celles qui peuvent être analysées comme *inutiles* d'après un simple examen de leur composition comparée à celle du but à prouver. Il s'agit là de techniques déjà utilisées par [Tammet, 1993] et plus généralement dans le cadre de la résolution [Voronkov, 1992].

1 Composition et de décomposition immédiates et en chaîne

Ces principes sont la traduction dans le processus de construction de preuves de ceux adoptés pour leur normalisation.

1.1 Composition immédiate et de décomposition en chaîne.

Ces principes découlent tous deux du fait que dans la normalisation des preuves, certaines inférences ont été montées au maximum si bien que leurs formules actives sont introduites par une inférence qui se situe juste avant.

Le principe de composition immédiate appliqué à un type d'inférence t, va consister dans la construction descendante d'une preuve à composer une formule à l'aide d'une inférence de type t, dès que ses composants apparaissent dans les conclusions intermédiaires.

Il se justifie par le théorème 4.6.2 vu dans le chapitre précédent.

Le principe de décomposition en chaîne appliqué à un type d'inférence t,va consister, lui, dans la construction ascendante d'une preuve, à poursuivre la décomposition d'une formule par celle de ses composants, dès qu'elle a commencé dans une inférence de type t.

Sa justification se trouve dans le théorème de normalisation 3.4.1, relié à la définition 3.4.1 des preuves normales.
[Andreoli, 1992] avait déjà mis en évidence ce dernier principe sous le nom de "focusing" de même que [Hodas and Miller, 1991, 1994] sous le nom de "backtracking".
Pour bien comprendre que les deux principes de composition immédiate et de décomposition en chaîne ne sont que les deux faces complémentaires d'un même phénomène, prenons un exemple.

Exemple 5.1.1 *Considérons une preuve de CLL normale comprenant une inférence $\otimes$ dont la formule principale $F_1 \otimes F_2$ est active dans une inférence $\oplus$. Que la normalisation ait été effectuée en vue d'une construction des preuves dans un sens ou dans l'autre, l'inférence $\oplus$ doit être située immédiatement après l'inférence $\otimes$. Elles forment donc la configuration suivante :*

$$\dfrac{\dfrac{\vdash F_1,\ \Delta_1 \quad \vdash F_2,\ \Delta_2}{\vdash F_1 \otimes F_2,\ \Delta_1,\ \Delta_2}\otimes}{\vdash (F_1 \otimes F_2) \oplus G,\ \Delta_1,\ \Delta_2}\oplus_1$$

Imaginons maintenant que nous soyons en train de construire la preuve de haut en bas. Dès que la formule $F_1 \otimes F_2$ apparaît dans une conclusion intermédiaire, nous pouvons la remplacer par $(F_1 \otimes F_2) \oplus G$ en vertu du principe de composition immédiate appliqué à $\oplus_1$.
Plaçons-nous maintenant dans une construction de la preuve de bas en haut. Eh bien ! Juste après avoir décomposé la formule $(F_1 \otimes F_2) \oplus G$, nous allons continuer en décomposant $F_1 \otimes F_2$ par application à $\oplus_1$ du principe de décomposition en chaîne.

Le lien entre forme d'une preuve normale et stratégie de construction apparaît clairement à travers cet exemple.
Pour savoir à quel type d'inférence, chaque principe s'applique, il suffit donc de reprendre les résultats des chapitre 3 et 4 relatifs à la forme des preuves normales dans CLL$\uparrow$ et dans CLL$\downarrow_\Delta$.

Le principe de décomposition en chaîne s'appliquera à $\otimes'$, $\oplus$, $?'$, $\exists$.
Le principe de composition immédiate s'appliquera à $\otimes$, $\otimes_{w1}$, $\otimes_{w2}$, $\wp$, $\oplus$, $?$, $\exists$.

1.2 Composition en chaîne et de décomposition immédiate

Ils découlent tous deux du fait que dans les preuves normales, certaines inférences sont descendues au maximum si bien que leur formule principale est active dans l'inférence qui suit immédiatement.

Le principe de composition en chaîne appliqué à un type d'inférence t, signifie que, dans la construction descendante d'une preuve, lorsque nous avons commencé à composer une formule F à l'aide d'une inférence de type t, nous poursuivons en composant la formule dont F est une sous-formule immédiate.

Il trouve sa justification dans le théorème de normalisation 4.6.1 relié à la définition 4.6.1 des preuves normales.

Le principe de décomposition immédiate appliqué à un type d'inférence t, consiste, dans la construction ascendante d'une preuve, à décomposer une formule à l'aide d'une inférence de type t, dès qu'elle apparaît dans une conclusion intermédiaire.

Il est justifié par le théorème 3.4.2.
[Andreoli, 1992] avait déjà mis en évidence ce principe sous le nom de *réversibilité* mais celui-ci n'apparaissait pas clairement chez lui comme symétrique du *"focusing"*. L'étude systématique que nous avons menée à partir de la permutabilité d'inférences, a permis de mettre en avant le fondement commun à ces deux principes.
Pour montrer la dualité entre le principe de composition en chaîne et celui de décomposition immédiate, prenons un exemple.

Exemple 5.1.2 *Considérons une preuve normale comportant une inférence de type & dont la formule principale $F_1 \& F_2$ est active dans une inférence de type $\oplus_1$. Quel que soit le sens de construction envisagé, l'inférence $\oplus_1$ doit suivre immédiatement l'inférence &. Celles-ci forment alors la configuration :*

$$\dfrac{\dfrac{\vdash F_1,\ \Delta \quad \vdash F_2,\ \Delta}{\vdash F_1 \& F_2,\ \Delta}\ \&}{\vdash (F_1 \& F_2) \oplus G,\ \Delta}\ \oplus_1$$

Si nous cherchons à construire cette preuve de haut en bas, après avoir composé la formule $F_1 \& F_2$, nous allons continuer en composant $(F_1 \& F_2) \oplus G$ par application à & du principe de composition en chaîne. Maintenant si nous essayons de construire la preuve dans l'autre sens, dès que la formule $F_1 \& F_2$ apparaît dans une conclusion intermédiaire, nous la décomposons en vertu du principe de décomposition immédiate appliqué à &.

La définition et les propriétés des preuves normales dans CLL$\downarrow_\Delta$ (cf chapitre 4) et dans CLL$\uparrow$ (cf chapitre 3) nous permettent de déterminer à quels types d'inférence s'appliquent ces principes.

Le principe de composition en chaîne s'appliquera à $\wp_{w1}$, $\wp_{w2}$, $\&$, $\&_{w1}$, $\&_{w2}$ *et* $\forall$.
Le principe de décomposition immédiate s'appliquera à $\wp$, $\&$, $!$, $\forall$ *et* $\perp$.

Nous allons maintenant mettre en œuvre ces principes pour élaborer des stratégies de recherche de preuves dans un sens et dans l'autre.

2 Stratégies de construction de preuves en chaînage arrière

Nous allons commencer par décrire le processus de construction de preuves de bas en haut dans CLL à travers une procédure générale, c'est-à-dire permettant d'engendrer toute preuve de CLL↑, et complète, c'est -à-dire qui réussit si le séquent auquel elle s'applique est prouvable.
Nous analyserons ensuite les sources d'indéterminisme dans la mise en œuvre de cette procédure et nous verrons dans quelle mesure des stratégies fondées sur les principes de décomposition immédiate et en chaîne, en engendrant des preuves normales, permettent de les réduire.
Nous verrons enfin qu'après application de ces principes, il subsiste des facteurs importants d'indéterminisme et nous ferons alors appel à des techniques qui ne sont plus fondées sur la permutabilité d'inférences, pour les réduire.

2.1 Une procédure complète de recherche de preuves

2.1.1 Description de la procédure

CLL étant indécidable [Lincoln *et al.*, 1990], cette procédure ne peut être qu'une procédure de semi-décision. Considérons un séquent $\vdash \Delta$ à prouver dans CLL. Puisque nous cherchons à construire une preuve de $\vdash \Delta$ de bas en haut, nous nous plaçons d'emblée dans le cadre approprié de CLL↑.
Le processus de construction s'organise autour d'une déduction de conclusion $\vdash \Delta$ et que nous noterons $\mathcal{D}_\Delta$ qui va traduire à tout moment l'état courant de la recherche et que nous appellerons *l'arbre de recherche.*

- Au départ, $\mathcal{D}_\Delta$ se réduit à l'hypothèse $\vdash \Delta$ que nous qualifierons de *but principal.*
- Chaque étape du processus va consister à choisir une hypothèse $\vdash \Delta'$ de $\mathcal{D}_\Delta$ qui constitue un *but* non satisfait de l'arbre de recherche, et à faire coïncider $\vdash \Delta'$ avec la conclusion d'une inférence I de CLL↑.
  - Si nous y arrivons, nous étendons l'arbre de recherche en ajoutant les prémisses de I comme fils de $\vdash \Delta'$. Nous disons que nous avons effectué *une expansion de $\mathcal{D}_\Delta$ à partir du but $\vdash \Delta'$ à l'aide de l'inférence I.* Si I est un axiome, le but $\vdash \Delta'$ est alors satisfait.
  - Si une telle expansion échoue, nous effectuons un retour-arrière jusqu'au premier but qui soit un ancêtre de $\vdash \Delta'$ et à partir duquel toutes les possibilités d'expansion n'ont pas encore été explorées.
- La recherche réussit lorsqu'il ne reste plus dans l'arbre de recherche, de but à satisfaire. $\mathcal{D}_\Delta$ est alors une preuve de $\vdash \Delta$.
- La recherche échoue lorsque $\mathcal{D}_\Delta$ se réduit à $\vdash \Delta$ et qu'il ne reste plus de possibilités d'expansions à partir du but principal. Celui-ci n'est donc pas prouvable.

2.1.2 Complétude de la procédure

La procédure sera complète si elle réussit chaque fois que le séquent $\vdash \Delta$ est prouvable. Or, dans CLL↑, la seule règle responsable de la non-terminaison de la recherche est la règle?' car c'est la seule pour laquelle la taille du prémisse est supérieure à celle de la conclusion. Donc, utilisée pour étendre l'arbre de recherche, elle va provoquer une augmentation de la taille des buts alors que les autres règles entraîne une diminution de cette taille. Une conséquence de ceci est par exemple que MALL est décidable car ce fragment n'utilise pas la règle?.
Pour assurer la complétude de la procédure de recherche, il faut donc gérer astucieusement l'application de la règle?'. Une façon simple de le faire, proposée par [Tammet, 1993], est de limiter le nombre d'inférences de type?' qui se suivent dans l'arbre de recherche. Le processus va alors se trouver divisé en *phases*, caractérisées par un nombre maximum d'inférences de type?' autorisées à se suivre dans $\mathcal{D}_\Delta$.
Nous commençons par effectuer la recherche sans utiliser la règle?'. Si la recherche échoue sans qu'on ait éprouvé le besoin d'utiliser la règle?' (nous dirons que la contrainte n'a pas joué), alors le séquent

n'est pas prouvable. Sinon, nous reprenons la recherche dans une nouvelle phase où nous nous permettons d'utiliser la règle?' une seule fois de suite dans $\mathcal{D}_\Delta$. Et ainsi de suite. La complétude est alors assurée par le fait que dans chaque phase, la procédure termine car *le nombre de valeurs possibles de $\mathcal{D}_\Delta$ y est fini.* L'inconvénient est que dans chaque phase, la recherche est reprise à zéro. Une manière d'éviter cela est pour chaque phase, de conserver en mémoire une photographie de l'arbre de recherche au moment où pour la première fois, la contrainte joue. Dans la phase suivante, la recherche sera reprise à partir de ce moment.
La procédure que nous venons de décrire peut être formalisée : c'est ce que nous avons fait dans la figure 5.1. Dans la procédure telle qu'elle est formalisée, la boucle interne termine toujours, ce qui assure la

FIG. 5.1 - *Procédure complète de construction d'une preuve de CLL↑ de bas en haut*

```
procédure prouver↑(⊢ Δ)
        initialiser D_Δ à ⊢ Δ ;
        initialiser le nombre limite n d'inférences de type?' autorisées à se suivre dans D_Δ, à 0 ;
        initialiser à vrai le booléen contrainte qui indique si la limite n a joué dans la phase précédente ;
        tantque contrainte = vrai
        faire contrainte := faux ;
            initialiser à faux le booléen finphase qui indique qu'une phase de recherche est terminée
            tantque D_Δ possède des buts non satisfaits et que finphase = faux
            faire choisir un but ⊢ Δ' ;
                si il est possible d'effectuer une nouvelle expansion à partir de ⊢ Δ'
                alors choisir une nouvelle inférence I de conclusion ⊢ Δ' ;
                    si l'expansion de D_Δ à partir de ⊢ Δ' et à l'aide de I est autorisée
                    alors effectuer cette expansion
                    sinon contrainte := vrai
                    finsi
                sinon si ⊢ Δ' = ⊢ Δ
                    alors finphase := vrai
                    sinon supprimer le but ⊢ Δ' de D_Δ
                    finsi
                finsi
            fintantque
            si finphase = faux
            alors retourner(prouvé)
            sinon n := n + 1
            finsi
        fintantque
        retourner(non prouvable)
```

complétude. La boucle externe, par contre, n'est pas assurée de terminer. D'autre part, il est clair que cette procédure est un cadre général permettant de construire n'importe quelle preuve de CLL↑. Il s'agit maintenant de la munir de stratégies visant à réduire l'indéterminisme pour qu'elle ne produise plus que des preuves normales.

2.2 Réduction de l'indéterminisme

Dans la procédure que nous venons de décrire, nous pouvons recenser trois moments où des choix sont effectués et où nous avons indéterminisme : *le choix du but* à étendre puis le premier étant fait, le choix au sein de ce but, de *la partie principale* de l'inférence qui va être utilisée et enfin *le choix de l'inférence* elle-même.
Analysons ces trois facteurs d'indéterminisme successivement et voyons en quoi l'application des principes de décomposition immédiate et en chaîne permet de les réduire.

2.2.1 Le choix du but

Il n'influe pas sur le résultat et ne provoquera ainsi aucun retour-arrière dans le processus. C'est pourquoi, l'indéterminisme correspondant peut être qualifié de "don't care".
Cela ne veut pas dire que nous devons être indifférents à ce choix. Au contraire, il peut être décisif en termes d'efficacité dans la recherche.
La règle que nous observons en la matière est de *choisir le but pour lequel on peut prévoir de trancher le plus rapidement possible pour savoir s'il est prouvable ou non.*
Cette prévision peut prendre des formes les plus diverses. Elle peut notamment s'appuyer sur le fait que certains buts se trouvent dans des fragments particuliers de CLL propres à une normalisation poussée des preuves et donc une construction efficace de celles-ci.

2.2.2 Le choix de la partie principale

Une fois que le but à partir duquel va se faire l'expansion est fixé, le choix à l'intérieur de celui-ci, des formules qui vont constituer la partie principale de inférence qui va permettre de faire l'expansion revient à choisir un ordre dans lequel les règles d'inférence vont être appliquées pour prouver le but. C'est pourquoi il relève d'un indéterminisme partiellement "don't know" et partiellement "don't care".
Et c'est ici qu'interviennent naturellement les principes de décompositions immédiate et en chaîne. Voyons comment ils s'appliquent.

- Si le but est de la forme $\vdash !F, ?\Delta'$, le théorème 3.4.1 de normalisation dans CLL$\uparrow$ nous permet de choisir à coup sûr !F comme formule principale.

- Si le but contient une formule ayant comme connecteur de tête, $\wp$, &, $\forall$ ou $\perp$, *le principe de décomposition immédiate* nous permet de choisir celle-ci comme formule principale.

- Si le but n'est pas d'une des deux formes indiquées juste avant et s'il contient d'autres formules que des littéraux, il est facile de tester tout d'abord s'il peut être produit par un axiome ou pas. En cas de réponse négative, la partie principale à choisir se réduit à une formule et relève alors d'un indéterminisme "don't know".
  Mais si nous choisissons une formule principale qui a comme connecteur de tête $\otimes$, $\oplus$ ou $\exists$, ses composantes, si elles ne sont pas des littéraux négatifs et si elles n'ont pas? comme connecteur de tête, seront les formules principales des sous-buts qui vont venir remplacer le but après expansion. C'est une application du *principe de décomposition en chaîne.*
  Si la formule principale choisie est de la forme ?F, elle sera introduite par une inférence de type?' et si F n'est pas un littéral, elle se retrouvera formule principale du sous-but qui va venir se substituer au but actuel, toujours par application du *principe de décomposition en chaîne.*

- Si le but ne contient que des littéraux, soit il n'est pas prouvable, soit il est la conclusion d'un axiome.

2.2.3 Le choix de l'inférence

A partir du moment où la partie principale est fixée, cela détermine dans la plupart des cas l'inférence à utiliser. Si elle contient plus d'une formule, l'inférence est un axiome. Sinon, le connecteur de tête va nous indiquer la règle à utiliser sauf dans un cas : s'il s'agit de $\oplus$, il y a deux règles possibles : $\oplus_1$ et $\oplus_2$.
La règle d'inférence une fois déterminée, cela ne veut pas dire pour autant que l'inférence elle-même est complètement fixée. C'est vrai en général sauf pour deux exceptions, $\otimes'$ et $\exists$.
Pour $\exists$, il faut encore choisir le terme qui va venir se substituer à la variable quantifiée. Il est facile de comprendre que celui-ci ne va pas être choisi complètement au hasard et que les termes présents dans le contexte vont nous permettre de limiter le choix. Nous étudierons ce problème dans la section 2.5.
Pour $\otimes'$, il faut partager le contexte présent dans le but en deux parties qui vont devenir les deux contextes des deux sous-buts qui vont venir remplacer le but actuel. Il y a là une cause importante d'indéterminisme. Ainsi, si le contexte contient n formules, il y a 2^n façons de le partager. Il est donc impératif de le réduire.

Nous verrons dans la section 2.5 qu'une méthode pour le faire, consiste à différer le choix jusqu'à ce que nous arrivions aux axiomes. C'est la même méthode qui sera utilisée pour $\exists$.

Voyons comment une stratégie s'appuyant sur les principes de décomposition immédiate et en chaîne peut être mise en œuvre à travers un exemple.

2.3 Un exemple

Soit à prouver dans CLL le séquent :

$$\vdash ?(?(!a \otimes \top)\&?(!b \otimes \top))\wp(?(a^{\perp} \oplus b^{\perp})\wp 0)$$

C'est la traduction de la formule de la logique intuitionniste suivant les règles de correspondance données par [Girard, 1987] :

$$(\neg a \vee \neg b) \Rightarrow \neg(a \wedge b)$$

L'application des principes de décompositions immédiate et en chaîne quand elle est possible, aboutit à ce que le processus de recherche va se trouver décomposé en une succession de phases déterministes (notées **D**) où ces principes sont appliqués et indéterministes (notées **I**) où ils ne peuvent pas l'être.

D. Le but étant constitué d'une seule formule ayant $\wp$ comme connecteur de tête, la première expansion est complètement déterminée. Ensuite, elle se poursuit par l'application du principe de décomposition immédiate à $\wp$. Le but courant devient alors :

$$\vdash ?(?(!a \otimes \top)\&?(!b \otimes \top)),\ ?(a^{\perp} \oplus b^{\perp}),\ \mathbf{0}$$

Pour alléger l'écriture, appelons F la formule $?(!a \otimes \top)\&?(!b \otimes \top)$ et G la formule $a^{\perp} \oplus b^{\perp}$. Le but courant peut alors s'écrire : $\vdash ?F,\ ?G,\ \mathbf{0}$.

I. Nous entrons dans une phase indéterministe où rien ne nous permet de trancher entre ?F et ?G comme formule principale de la prochaine inférence. Un mauvais choix nous imposera de faire ensuite un retour-arrière. Ici, seul le choix de ?F nous conduira au succès. En utilisant la règle ?', le but courant est alors remplacé par :

$$\vdash ?(!a \otimes \top)\&?(!b \otimes \top),\ ?F,\ ?G,\ \mathbf{0}$$

D. Par application du principe de composition en chaîne à ?' (ou ce qui revient au même, de composition immédiate à &), nous obtenons deux sous-buts :

$$\vdash ?(!a \otimes \top),\ ?F,\ ?G,\ \mathbf{0} \quad \text{et} \quad \vdash ?(!b \otimes \top),\ ?F,\ ?G,\ \mathbf{0}$$

La construction de la preuve se divise maintenant en deux.

1. Satisfaction du sous-but $\vdash ?(!a \otimes \top),\ ?F,\ ?G,\ \mathbf{0}$

   I. Il y a le choix entre 3 formules principales pour la prochaine inférence : celui de ?F ou ?G ne fait qu'ajouter des pas inutiles dans la recherche (si nous n'avions pas pris soin de limiter l'utilisation de la règle ?', il risquerait même d'amener celle-ci à se prolonger indéfiniment). Celui de $?(!a \otimes \top)$ est celui qui nous mène le plus rapidement au succès.
   Le but courant se transforme alors en :

   $$\vdash !a \otimes \top,\ ?(!a \otimes \top),\ ?F,\ ?G,\ \mathbf{0}$$

   D. L'application du principe de décomposition en chaîne à ?' nous permet de transformer ce but en deux sous-buts :

   $$\vdash !a,\ ?(!a \otimes \top),\ ?F,\ ?G \quad \text{et} \quad \vdash \top,\ ?(!a \otimes \top),\ ?F,\ ?G,\ \mathbf{0}$$

   Le second est la conclusion d'un axiome de type $\top$. Le premier peut encore se réduire grâce au théorème de normalisation 3.4.1 en :

   $$\vdash a,\ ?(!a \otimes \top),\ ?F,\ ?G$$

I. Pour la formule principale de la prochaine inférence, il y a trois possibilités. Celle qui mène le plus vite au succès est ?G, c'est-à- dire $?(a^{\perp} \oplus b^{\perp})$. Par application de la règle ?', nous obtenons alors le but :

$$\vdash\ a^{\perp} \oplus b^{\perp},\ a,\ ?(!a \otimes \top),\ ?F,\ ?G$$

D Nous appliquons le principe de décomposition en chaîne à ?'. Le choix entre les règles $\oplus_1$ et $\oplus_2$ est facile à faire grâce à des considérations de neutralité logique (nous y reviendrons en détail un peu plus loin). Nous appliquons la règle $\oplus_1$ et nous obtenons alors le but courant :

$$\vdash\ a^{\perp},\ a,\ ?(!a \otimes \top),\ ?F,\ ?G$$

Celui-ci est alors la conclusion d'un axiome de type *id'*.

2. Satisfaction du sous-but $\vdash\ ?(!b \otimes \top),\ ?F,\ ?G,\ \mathbf{0}$
Nous procédons d'une façon analogue à la satisfaction de l'autre sous-but. Les rôles des atomes a et b sont seulement ici inversés.

La preuve complète ainsi construite se présente ainsi :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash a^{\perp},\ a, ?(!a \otimes \top),\ ?F,\ ?G}\,\mathrm{id'}}{\vdash a^{\perp} \oplus b^{\perp}, a, ?(!a \otimes \top), ?F, ?G}\,\oplus_1}{\vdash\ a,\ ?(!a \otimes \top),\ ?F,\ ?G}\,?'}{\vdash\ !a,\ ?(!a \otimes \top),\ ?F,\ ?G}\,!\quad \dfrac{}{\vdash \top, ?(!a \otimes \top), ?F, ?G, \mathbf{0}}\,\top}{\vdash\ !a \otimes \top,\ ?(!a \otimes \top),\ ?F,\ ?G,\ \mathbf{0}}\,\otimes'}{\vdash\ ?(!a \otimes \top),\ ?F,\ ?G,\ \mathbf{0}}\,?'\quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash\ b^{\perp},\ b,\ ?(!b \otimes \top),\ ?F,\ ?G}\,\mathrm{id'}}{\vdash a^{\perp} \oplus b^{\perp}, b, ?(!b \otimes \top), ?F, ?G}\,\oplus_2}{\vdash\ b,\ ?(!b \otimes \top),\ ?F,\ ?G}\,?'}{\vdash\ !b,\ ?(!b \otimes \top),\ ?F,\ ?G}\,!\quad \dfrac{}{\vdash \top, ?(!b \otimes \top), ?F, ?G, \mathbf{0}}\,\top}{\vdash\ !b \otimes \top,\ ?(!b \otimes \top),\ ?F,\ ?G,\ \mathbf{0}}\,\otimes'}{\vdash\ ?(!b \otimes \top),\ ?F,\ ?G,\ \mathbf{0}}\,?'}{\vdash\ ?(!a \otimes \top) \& ?(!b \otimes \top),\ ?F,\ ?G,\ \mathbf{0}}\,\&}{\vdash\ ?F,\ ?G,\ \mathbf{0}}\,?'}{\vdash ?F,\ ?G \wp \mathbf{0}}\,\wp}{\vdash ?F \wp (?G \wp \mathbf{0})}\,\wp$$

Cet exemple laisse deviner une des faiblesses de la stratégie utilisée : toute formule de la forme $?F$ qui apparaît dans un but suite à une décomposition, y subsiste jusqu'aux axiomes. Donc au fur et à mesure que l'arbre de recherche se construit, le nombre de formules de la forme $?F$ présentes dans les buts, risque de grossir augmentant par là l'indéterminisme concernant le choix de la formule principale.
Nous allons voir maintenant d'autres techniques qui ne sont pas fondées sur la permutabilité d'inférences, qui vont nous aider à réduire l'indéterminisme d'une façon différente.

2.4 Neutralité logique globale des séquents prouvables

Nous allons montrer comment il est possible d'utiliser un filtre qui permette de détecter une certaine classe de séquents non prouvables et qui évite ainsi d'entreprendre une recherche de preuve inutile pour cette classe de séquents.
Si on s' intéresse à l'origine dans les preuves des littéraux (formules atomiques ou leurs négations), autres que les constantes logiques, on constate qu'elle est de deux ordres :

- certains sont formules principales d'axiomes de type *id'* mais dans ce cas là, ils sont introduits en même temps que leur négation qui doit donc être présente en temps que sous-formule dans la conclusion de la preuve ;

- d'autres apparaissent lors de l'introduction de formules de la forme ?F et $F \oplus G$ dans des inférences de type *id'* et $\oplus$.

On peut prévoir que certains de ces littéraux se rangeront toujours dans la première catégorie, quelle

que soit la preuve de ce séquent. Ce sont ceux qui ne sont pas dans le champ d'un opérateur? ou $\oplus$. Nous les appellerons les *ressources effectives* du séquent tandis que les autres constitueront ses *ressources potentielles.*
Les ressources effectives étant dans une preuve introduite par un axiome de type *id'*, elles doivent y trouver leur négation d'où la propriété suivante :

Dans un séquent prouvable dans CLL, toute ressource effective trouve sa négation.

Montrons sur un exemple comment cette propriété peut être utilisée comme filtre pour écarter certains séquents non prouvables.

Exemple 5.2.1 *Considérons le séquent :* $\vdash\ a^{\perp} \otimes b,\ b \oplus a,\ a,\ ?(a^{\perp}\&b^{\perp}) \oplus c,\ a\&c$
Ses ressources effectives sont les littéraux $a^{\perp}$ *et* b *constituant la première formule, l'atome isolé* a *et les atomes* a *et* c *formant la dernière formule. Tous ces littéraux ont leur négation dans le séquent sauf* c*. Donc le séquent n'est pas prouvable.*

On peut préciser où doit être localisée la négation d'une ressource effective. Ainsi par exemple, le séquent $\vdash\ a \otimes a^{\perp},\ \Delta$ où Δ ne contient pas l'atome a, n'est pas prouvable car a et sa négation $a^{\perp}$ sont dans deux composantes différentes d'une conjonction donc ils se trouveraient, au cours de la preuve du séquent, inévitablement répartis dans deux séquents différents en contradiction avec le théorème ci-dessus.
Il faut voir toutefois que, plus on pousse loin le raffinement, plus la mise en œuvre du filtre est coûteuse en temps. Il y a donc un équilibre à trouver entre temps que le filtre fait gagner et temps que son utilisation nécessite.
Il y a toutefois un fragment de LL où il peut être amélioré de façon efficace : MLL. Tous les littéraux d'un séquent de MLL sont des ressources effectives d'où la propriété suivante :

Dans tout séquent prouvable de MLL, on peut partitionner les ressources en paires de littéraux qui sont négation l'un de l'autre.

2.5 Ajournement du choix dans l'application des règles $\exists$ et $\otimes$

2.5.1 Cas de la règle $\exists$

Considérons tout d'abord la règle $\exists$. Rappelons sa forme :

$$\frac{\vdash\ F[t/x],\ \Delta'}{\vdash\ \exists x F,\ \Delta'}\ \exists$$

Au moment où elle est appliquée pour réaliser une expansion de l'arbre de recherche, nous ne disposons d'aucune information qui puisse nous guider dans le choix du terme t qu'il va falloir substituer à x. Les contraintes sur la forme que doit prendre le terme t, s'exprimeront seulement lorsque nous allons obtenir des buts de la forme $\vdash\ a(t_1),\ a(t_2)^{\perp},\ ?\Delta'$. Pour que de tels buts soient la conclusion d'axiomes *id'*, il va falloir que les termes t_1 et t_2 soient identiques. Si par exemple t_1 est égal à un terme clos f(c,g(c)) et si t_2 s'exprime en fonction du terme t intervenant dans l'inférence $\exists$, de la façon suivante : $t_2 = f(c,t)$, cela entraîne comme contrainte que t doit être égal à g(c).
Ainsi, il faut différer la substitution du terme t à la variable x jusqu'à ce que nous arrivions aux axiomes où elle se fera au travers d'une unification.
[Shankar, 1992] a proposé un mécanisme de *skolémisation dynamique* pour la logique intuitionniste qui a été ensuite adapté à la logique linéaire [Lincoln and Shankar, 1994]. Ce mécanisme permet à la fois de reporter les substitutions liées aux inférences $\exists$ et de garantir la correction des inférences $\forall$.
Nous avons intégré ce mécanisme lors de l'implantation de nos stratégies mais nous n'avons pas retenu une dernière proposition de [Lincoln and Shankar, 1994] qui vise à rendre permutables les inférences de type $\exists$ avec celles de type $\forall$ car elle restreint l'application de la permutabilité des autres types d'inférence.

2.5.2 Cas de la règle $\otimes'$

Venons-en maintenant au problème du partage du contexte final dans l'application de la règle $\otimes'$ pour constituer les deux contextes initiaux. Rappelons cette règle.

$$\frac{\vdash F_1,\ \Delta_1,\ ?\Delta' \quad \vdash F_2,\ \Delta_2,\ ?\Delta'}{\vdash F_1 \otimes F_2,\ \Delta_1,\ \Delta_2,\ ?\Delta'}\ \otimes'$$

Le problème est de diviser le contexte final Δ_1, Δ_2 en deux contextes initiaux Δ_1 et Δ_2. Nous allons reprendre une proposition de [Hodas and Miller, 1991, 1994] qui repose sur la même idée que la méthode utilisée pour réduire l'indéterminisme relatif à la règle $\exists$: différer le choix jusqu'à ce qu'on ait l'information suffisante pour le faire.
On cherche à prouver la formule F_1 avec l'ensemble du contexte final mais en sachant que l'on pourra y puiser la seule part nécessaire pour le faire. Ce qui reste du contexte après la preuve de F_1, constituera le contexte dans lequel F_2 va devoir être démontrée mais cette fois, il devra être consommé complètement. Cette méthode réduit considérablement les retours-arrière dans la recherche mais ne les supprime pas totalement.

Exemple 5.2.2 *Soit à prouver le séquent :* $\vdash\ (a \oplus b) \otimes (a \oplus c),\ a^{\perp},\ b^{\perp}$.
Nous allons d'abord chercher à prouver $a \oplus b$ *en puisant dans le contexte* $a^{\perp}$, $b^{\perp}$ *ce qui est nécessaire pour le faire.*
Une possibilité est d'utiliser la formule $a^{\perp}$ *car le séquent* $\vdash\ a \oplus b,\ a^{\perp}$ *est prouvable.*
Il faut ensuite utiliser le reste du contexte, c'est-à-dire la formule $b^{\perp}$ *pour prouver* $a \oplus c$. *Ce n'est pas possible ce qui entraîne un retour-arrière pour effectuer une autre preuve de* $a \oplus b$. *C'est possible en utilisant cette fois la formule* $b^{\perp}$. *La partie restante du contexte est maintenant la formule* $a^{\perp}$ *qui permet de démontrer* $a \oplus c$. *Le séquent initial est donc prouvable.*

2.6 Conclusion

Ce qu'il faut retenir de cette étude de la construction ascendante de preuves dans CLL, ce n'est pas avant tout les résultats obtenus en termes de stratégies. Pour l'essentiel, ils reprennent ceux obtenus par [Andreoli, 1992] et intègrent certaines techniques particulières proposées par [Lincoln and Shankar, 1994] pour gérer les variables quantifiées et par [Hodas and Miller, 1991, 1994] pour gérer le partage de contexte dans l'application de la règle $\otimes$.
Notre travail met en lumière *les liens étroits entre les notions de permutabilité, mouvement d'inférences, normalisation de preuves et stratégies de recherche de preuves.*
A travers la démarche suivie, se dessine *une méthode générale pour élaborer des stratégies de recherche de preuves.* Nous l'avons appliquée à CLL mais dans un cadre aussi large, nous ne pouvions pas espérer déboucher sur des stratégies très efficaces. Il faut pour cela, travailler dans un fragment logique beaucoup plus restreint. Le choix du fragment sera fonction de l'application que l'on veut en faire. C'est ce que nous verrons dans le chapitre 6 avec l'élaboration d'un calcul de processus.

3 Stratégies de construction de preuves en chaînage avant

Comme dans la section précédente, nous commencerons par expliquer en quoi consiste la construction d'une preuve de haut en bas, à travers la présentation d'une procédure générale et complète.
Puis nous analyserons les causes d'indéterminisme dans la mise en œuvre de cette procédure et nous verrons comment l'application des principes de composition immédiate et en chaîne peut agir sur certaines d'entre elles pour engendrer des preuves normales. Pour les autres, nous verrons comment y faire face à l'aide de techniques qu'on retrouve dans la mise en œuvre de la résolution.

3.1 Une procédure complète de recherche de preuve

3.1.1 Description de la procédure

Soit un séquent $\vdash \Delta$ à prouver dans CLL. Comme nous cherchons à en construire une preuve de haut en bas, nous nous plaçons dans le cadre approprié de CLL$\downarrow_\Delta$. Il s'agira donc de montrer que $\overset{a}{\vdash}\Delta$ est prouvable dans CLL$\downarrow_\Delta$ pour un certain indicateur d'affaiblissement a.
A la différence de la construction ascendante de preuves, la construction ne va pas s'organiser autour de l'arbre de preuve mais autour d'un ensemble de séquents qui seront sensés représenter à un moment donné, les conclusions intermédiaires de la preuve à construire qui sont déjà établies et qui se situent le plus en avant (c'est-à-dire le plus bas possible dans l'arbre de preuve). Nous appellerons cet ensemble la *base de recherche* et nous la noterons $\mathcal{E}_\Delta$. Ses éléments seront qualifiés de *clauses* par référence à la résolution à laquelle la méthode s'apparente.

- Au départ, $\mathcal{E}_\Delta$ est vide.
- Chaque étape du processus de recherche consiste à effectuer une inférence à partir de prémisses qui sont membres de $\mathcal{E}_\Delta$. Au départ, la base étant vide, ce sont nécessairement des axiomes qui sont utilisés comme inférences.
  La conclusion produite vient remplacer naturellement dans $\mathcal{E}_\Delta$ les prémisses dont elle est issue. Mais l'indéterminisme dans la recherche se manifeste ici de la façon suivante : si, à partir d'un prémisse donné, il y a indétermination quant à la conclusion à produire, ce prémisse est conservé dans la base de recherche. Ainsi, *l'indéterminisme se traduira ici non par un phénomène de retour-arrière mais par une augmentation de la taille de la base de recherche.*
- La recherche réussit si, dans une étape du processus, la conclusion nouvelle inférée à partir de $\mathcal{E}_\Delta$ est le but $\overset{a}{\vdash}\Delta$ à prouver.

3.1.2 Complétude de la procédure

Il est clair que si les inférences ne sont pas guidées, le processus risque le plus souvent de ne pas terminer. Heureusement, nous recherchons des preuves sans coupures c'est-à-dire qui respectent *la propriété de la sous-formule. Nous n'effectuerons donc que des inférences qui introduisent des sous-formules du but.* Ce principe, s'il restreint considérablement l'espace de recherche, n'assure pas pour autant la complétude de la procédure car certaines sous-formules peuvent se trouver dupliquées. Une façon d'y parvenir, est de limiter la taille des clauses utilisées pour en inférer de nouvelles. Le processus de recherche va se voir découpé en *phases* correspondant chacune à une taille limite des clauses utilisées.
Dans une première phase, nous n'utiliserons dans $\mathcal{E}_\Delta$ que les clauses qui comprennent au maximum une formule. Si dans cette phase, nous n'inférons ni le but, ni de clause contenant plus d'une formule, nous pouvons conclure que le séquent $\vdash \Delta$ n'est pas prouvable. Sinon, si le but n'a pas été inféré, nous entamons une nouvelle phase où le nombre de formules autorisé dans un prémisse d'une inférence est maintenant de 2 et ainsi de suite.
Si nous nous situons dans un cadre propositionnel, la complétude de la procédure découle du fait qu'il n'y a qu'un nombre fini de séquents formés de sous-formules du but et de taille limitée. Si nous passons au premier ordre, ce n'est pas le cas car à partir d'une formule de la forme $\exists x F$, il peut être engendré une infinité de sous-formules de la forme $F[t_1/x], \cdots, F[t_n/x], \cdots$. Mais nous verrons que nous pouvons toujours nous ramener à un nombre fini de termes $t_1, \cdots, t_p$ pour effectuer les substitutions correspondantes. Donc dans ce cas aussi, chaque phase est assurée de terminer.
Nous pouvons formaliser la procédure que nous venons de décrire. C'est ce que nous faisons dans la figure 5.2. Nous allons maintenant analyser en détail les sources d'indéterminisme dans la mise en œuvre de cette procédure et voir dans quelle mesure l'application des principes de compositions immédiate et en chaîne permet de les réduire.

3.2 Réduction de l'indéterminisme

Examinons les choix que nous devons effectuer à chaque étape du processus de recherche. Ils se résument à trois :

FIG. 5.2 - *Procédure complète de preuve en avant dans $CLL{\downarrow}_\Delta$*

```
procédure prouver↓(⊢ Δ)
        construire la base d'affaiblissement B_Δ associée à ⊢ Δ ;
        initialiser la base de recherche ℰ_Δ aux clauses inférées par des axiomes
        et contenant uniquement des sous-formules du but ;
        initialiser la taille limite l autorisée pour les prémisses des inférences à 1 ;
        tantque il existe dans ℰ_Δ des clauses de taille supérieure ou égale à l
        faire tantque il est possible de faire de nouvelles inférences
                à partir de clauses de ℰ_Δ de taille ne dépassant pas l
            faire   choisir une ou deux clauses de ℰ_Δ de taille ne dépassant pas l ;
                    choisir une nouvelle inférence I ayant pour prémisses les clauses choisies ;
                    si la conclusion produite est de la forme ⊢^a Δ
                    alors retourner(prouvé)
                    sinonsi la conclusion produite ∉ ℰ_Δ
                        alors l'insérer dans ℰ_Δ
                        finsi
                    finsi
            fintantque
            l := l + 1
        fintantque
        retourner(non prouvable)
```

- celui d'une clause qui constitue un prémisse de la prochaine inférence ;

- celui des formules qui, au sein de cette clause, vont constituer la partie active de l'inférence ; du fait d'une différenciation de toutes les sous-formules du but par un marquage, ce choix détermine totalement le type de l'inférence et sauf pour $\otimes$ et &, l'inférence elle-même ;

- dans le cas où l'inférence produit une conjonction, le choix de l'autre prémisse.

Etudions maintenant en détail l'indéterminisme relatif à chacun de ceux-ci.

3.2.1 Choix d'une clause comme prémisse de la prochaine inférence

L'indéterminisme lié à ce choix est "don't care" en ce sens qu'il n'a pas d'influence sur l'introduction de clauses inutiles à la preuve dans la base de recherche. Par contre, il peut affecter l'efficacité de la recherche en termes d'espace et de temps. Pour répondre à ce souci, on peut se fixer une priorité : *choisir d'abord les clauses à partir desquelles il est possible d'en inférer de nouvelles de façon déterministe.* Ainsi, on n'augmente pas la taille de la base de recherche.

3.2.2 Choix de la partie active de cette clause

Il revient à effectuer un choix de l'ordre dans lequel les inférences où les formules de la clause sont actives, vont être effectuées. L'indéterminisme lié à ce choix est donc partiellement "don't care", partiellement "don't know". Et c'est pour le réduire qu'ont été élaborés les principes de composition immédiate et en chaîne. Voyons de quelle façon.

Supposons que l'on ait choisi une clause de $\mathcal{E}_\Delta$ comme prémisse de la prochaine inférence et dans cette clause, les formules qui en constitueront la partie active. Selon le type de l'inférence, elles vont être au nombre de 0, 1 ou 2. Si cette partie active n'est pas vide, elle va nous aider, grâce à la propriété de la sous-formule, à prévoir la formule principale à produire et l'inférence qui va le faire mais elle ne détermine pas complètement cette prévision.

Considérons par exemple le but à prouver : $\vdash ?b \oplus c,\ ?b \otimes c^\perp$. Supposons que nous ayons choisi comme prémisse de la prochaine inférence, une clause de la forme $\vdash^{a'} ?b,\ \Gamma$ et comme partie active la formule $?b$. La propriété de la sous-formule nous permet seulement de dire que la formule principale qui sera

produite, sera soit $?b \oplus c$, soit $?b \otimes c^{\perp}$. Nous pourrons donc effectuer soit une inférence de type $\oplus_1$, soit une inférence de type $\otimes$.
Pour limiter ce phénomène, nous avons choisi de *marquer dans le but toutes les sous-formules qui ne sont pas des littéraux, afin de les différencier. De cette façon, une formule d'une clause de la base qui n'est pas un littéral, ne pourra plus s'identifier avec plus d'une sous-formule du but.*
Ce marquage étant maintenant sous-entendu pour la suite, revenons maintenant au choix de la partie active d'une clause considérée comme prémisse de la prochaine inférence. Nous pouvons alors distinguer plusieurs cas.

- La clause se trouve dans l'une des trois situations suivantes :

  1. elle contient une formule qui n'est pas un littéral, qui n'est pas de la forme $?F$ et qui, dans le but $\vdash \Delta$, est composante d'une sous-formule ayant $\otimes$, $\oplus$ ou $\exists$ comme connecteur de tête ;
  2. elle contient une formule qui n'est pas un littéral et qui, dans le but, est composante d'une sous-formule ayant ? comme connecteur de tête ;
  3. elle contient deux formules qui ne sont pas toutes deux des littéraux ou qui n'ont pas toutes deux ? comme connecteur de tête et qui, dans le but, constituent les deux composantes d'une sous-formule ayant $\wp$ comme connecteur de tête.

  Alors la ou les formules en question peuvent être choisies comme formules actives. Si l'inférence correspondante ne nécessite pas un autre prémisse (c'est-à-dire si elle n'est pas de type $\otimes$), elle est effectuée. Dans la base de recherche, la clause qui a servi de prémisse, est alors remplacée par la nouvelle conclusion. Cela signifie que l'étape est déterministe, ce qui est justifié par le *principe de composition immédiate* appliqué ici.

- Dans les autres cas, il y a un indéterminisme "don't know" concernant le choix de la partie active, mais une fois qu'il est fait et que la partie active sélectionnée n'est pas vide, il détermine complètement celui de l'inférence sauf si elle est de type & auquel cas nous avons besoin d'un autre prémisse. Du fait de l'indéterminisme, la conclusion produite ne viendra pas remplacer dans la base de recherche, le prémisse dont elle est issue mais elle s'ajoutera à lui.
  Toutefois, dans le cas où l'inférence effectuée est de type $\wp_{w1}$, $\wp_{w2}$, & ou $\forall$, la formule principale est fixée comme formule active d'une prochaine inférence si cette inférence n'est pas de type $\wp$. Il s'agit là de l'application du *principe de composition en chaîne.*

3.2.3 Dans le cas où l'inférence produit une conjonction, choix de l'autre prémisse.

L'indéterminisme relatif à ce choix est très limité du fait que l'on connaît à la fois la conjonction à produire et un des prémisses de l'inférence.

Illustrons ces principes à l'aide d'un exemple.

3.3 Un exemple

Reprenons le séquent utilisé pour illustrer la construction d'une preuve en chaînage arrière (voir sous-section 2.3 de ce chapitre). Il avait la forme :

$$\vdash ?(?(!a \otimes \top)\&?(!b \otimes \top))\wp(?(a^{\perp} \oplus b^{\perp})\wp 0)$$

Nous travaillerons bien entendu dans le système d'inférence spécialisé $\mathrm{CLL}{\downarrow_{\Delta}}$ associé à ce séquent $\vdash \Delta$. Entamons la procédure de recherche telle qu'elle est décrite dans la figure 5.2 mais en appliquant une stratégie s'appuyant sur les principes de composition immédiate et en chaîne.

1. Construction de la base d'affaiblissement B_{Δ}.
   $B_{\Delta} = \{?(?(!a \otimes \top)\&?(!b \otimes \top)),\ \ ?(!a \otimes \top),\ \ ?(!b \otimes \top),\ \ ?(!a \otimes \top)\&?(!b \otimes \top),\ \ ?(a^{\perp} \oplus b^{\perp})\}$

2. Initialisation de la base de recherche $\mathcal{E}_\Delta$.
   Les clauses initiales sont celles qui sont produites par des axiomes et dont toutes les formules sont des sous-formules du but. D'où :

$$\mathcal{E}_\Delta = \{\vdash^0 a,\ a^\perp \quad \vdash^0 b,\ b^\perp \quad \vdash^1 \top\}$$

3. Recherche proprement dite.
   Elle va se traduire par une suite de phases déterministes (notées **D**) où de nouvelles clauses vont se substituer à d'anciennes, et de phases indéterministes (notées **I**) où les nouvelles clauses vont venir s'ajouter aux anciennes.

D. Nous commençons par effectuer tous les pas déterministes qu'il est possible de faire à partir de la base de recherche initiale.
Dans les axiomes d'identité, seuls les littéraux négatifs peuvent être actifs dans des inférences de type $\oplus$. D'où un nouvel état de la base de recherche :

$$\mathcal{E}_\Delta = \{\vdash^0 a,\ a^\perp \oplus b^\perp \quad \vdash^0 b,\ a^\perp \oplus b^\perp \quad \vdash^1 \top\}$$

L'exécution du pas déterministe qui vient d'être effectué ne découle pas d'un principe exposé auparavant mais de l'analyse particulière du but à prouver. Une telle analyse est toutefois facile à intégrer dans un système automatique.
Ensuite par application du principe de composition immédiate à ?, nous obtenons :

$$\mathcal{E}_\Delta = \{\vdash^0 a,\ ?(a^\perp \oplus b^\perp) \quad \vdash^0 b,\ ?(a^\perp \oplus b^\perp) \quad \vdash^1 \top\}$$

On pourrait montrer facilement que lorsqu'on est sûr qu'une inférence de type ! ne sera pas effectuée après une autre de type &, on peut la réaliser dès que possible. En utilisant cette propriété, nous obtenons :

$$\mathcal{E}_\Delta = \{\vdash^0 !a,\ ?(a^\perp \oplus b^\perp) \quad \vdash^0 !b,\ ?(a^\perp \oplus b^\perp) \quad \vdash^1 \top\}$$

Ensuite le principe de composition immédiate appliqué successivement à $\otimes$ et ? pour les deux premières clauses, nous amène à :

$$\mathcal{E}_\Delta = \{\vdash^1 ?(!a \otimes \top),\ ?(a^\perp \oplus b^\perp) \quad \vdash^1 ?(!b \otimes \top),\ ?(a^\perp \oplus b^\perp) \quad \vdash^1 \top\}$$

I. Il s'agit de choisir un prémisse de la prochaine inférence. La dernière clause est exclue car il n'est pas possible de réaliser à partir de $\mathcal{E}_\Delta$ dans l'état, une inférence où la constante $\top$ soit active. Les deux premières clauses étant similaires, le choix entre les deux n'a guère d'importance. Choisissons par exemple la première. Elle a deux formules actives potentielles : $?(!a \otimes \top)$ et $?(a^\perp \oplus b^\perp)$.
On ne peut pas le savoir a priori mais les deux mènent au but. Sélectionnons par exemple la première. Cela détermine le type de l'inférence à effectuer : &. La formule principale sera : $?(!a \otimes \top)\&?(!b \otimes \top)$. Il ne reste plus qu'à trouver l'autre prémisse. Or, seule la deuxième clause convient ; cela tient à la forme particulière des règles $\&_{w1}$ et $\&_{w2}$ qui exigent que l'un des prémisses ne soit pas affaiblissable.
L'inférence nous permet alors de produire la nouvelle clause suivante :

$$\vdash^1 ?(!a \otimes \top)\&?(!b \otimes \top),\ ?(a^\perp \oplus b^\perp)$$

Elle est ajoutée à $\mathcal{E}_\Delta$ qui compte maintenant 4 éléments.

D. Il s'agit maintenant d'examiner s'il n'est pas possible d'effectuer de transformation déterministe à partir de la nouvelle clause qui vient d'être insérée dans la base de recherche.
Il est possible d'appliquer le principe de composition immédiate à ?. Le nouvel état de la base est alors le suivant :

$$\begin{aligned}\mathcal{E}_\Delta = \{&\vdash^1 ?(!a \otimes \top),\ ?(a^\perp \oplus b^\perp) \quad \vdash^1 ?(!b \otimes \top),\ ?(a^\perp \oplus b^\perp) \quad \vdash^1 \top \\ &\vdash^1 ?(?(!a \otimes \top)\&?(!b \otimes \top)),\ ?(a^\perp \oplus b^\perp)\}\end{aligned}$$

I. Il y a plusieurs possibilités pour choisir un prémisse de la nouvelle inférence mais la meilleure consiste à prendre la dernière clause produite. Alors, la seule formule active possible est $?(a^{\perp} \oplus b^{\perp})$ et l'inférence est alors de type $\wp_{w2}$. La clause est alors remplacée par :

$$\vdash^{1} ?(?(!a \otimes \top)\&?(!b \otimes \top)),\ (?(a^{\perp} \oplus b^{\perp}))\wp \mathbf{0}$$

D Par application à $\wp_{w2}$ du principe de composition en chaîne, nous pouvons inférer le but $\vdash \Delta$ à l'aide de la règle $\wp$ appliquée à la nouvelle clause.

3.4 Gestion des substitutions au premier ordre.

Comme dans le processus ascendant de construction de preuves, il vaut mieux reporter les substitutions relatives à l'application de la règle $\exists$ au niveau des axiomes où elles seront effectuées par le biais de l'unification.
Cela implique qu'on puisse différencier les variables quantifiées existentiellement des autres. Nous représenterons chacune d'elles par une *variable de Herbrand* singulière (notée $\mathbf{x_1}, \cdots, \mathbf{x_n}$).
Pour indiquer aussi que chaque variable quantifiée universellement représente un identificateur nouveau, nous la représenterons par une *constante de Herbrand* singulière (notée $\mathbf{h_1}, \cdots, \mathbf{h_n}$).
Le processus de recherche d'une preuve se déroule alors de la façon suivante :

- Les clauses constituant la base de recherche initiale sont d'une part les conclusions des axiomes $\top'$ et 1 si les constantes correspondantes sont des sous-formules du but. D'autre part, elles sont de la forme $\vdash^{0} A, B^{\perp}$ avec A et $B^{\perp}$ sous-formules du but où les variables quantifiées existentiellement et universellement ont été remplacées respectivement par des variables et constantes de Herbrand. Mais pour qu'elles puissent être considérées comme le produit d'axiomes de type *id*, il faut que les termes A et B soient unifiables par une substitution portant uniquement sur les variables de Herbrand. En général, celle-ci n'est pas unique mais celle qui est retenue, est la plus générale. Elle n'est alors *pas effectuée mais seulement attachée à la clause* qui sera notée ainsi : $s \vdash^{0} A, B^{\perp}$ avec s représentant la substitution portant sur les variables de Herbrand qui unifie A et B et qui soit la plus générale possible.
  A partir de là, nous allons manipuler des clauses couplées à des substitutions (les conclusions des axiomes de type $\top'$ ou 1 sont couplés à la substitution vide).

- Chaque pas dans le processus de recherche étant constitué par une inférence, il va falloir indiquer comment la substitution attachée à la conclusion est définie à partir de celles attachées aux prémisses.

  1. l'inférence comporte un prémisse et n'est pas de type $\forall$ ou $\exists$.
     La substitution associée à la conclusion est celle attachée au prémisse.

  2. l'inférence est de type $\forall$.
     La règle peut alors s'écrire ainsi :

     $$\frac{s \vdash^{a} F[\mathbf{h_k}/x], \Delta}{s \vdash^{a} \forall x, \Delta}\ \forall$$

     avec $\mathbf{h_k}$ *non libre dans la conclusion instanciée par* s

  3. L'inférence est de type $\exists$.
     La règle peut alors s'écrire ainsi :

     $$\frac{s \vdash^{a} F[\mathbf{x_k}/x], \Delta}{s' \vdash^{a} \exists x, \Delta}\ \exists$$

     où s' *est obtenue en supprimant l'instanciation de* $\mathbf{x_k}$ *de la substitution* s

4. L'inférence produit une conjonction.
   Si s_1 et s_2 sont les substitutions attachées aux prémisses, celle qui est couplée à la conclusion est la substitution s qui est à la fois moins générale que s_1 et s_2 mais qui est la plus générale possible. Cette substitution n'existe pas toujours car s_1 et s_2 doivent être compatibles mais si elle existe, elle est unique et nous la noterons $s_1 \cup s_2$.

Il est facile de montrer que le système d'inférence ainsi obtenu est équivalent à $\mathrm{CLL}{\downarrow}_{\Delta}$.
Illustrons cette manière particulière de gérer les substitutions par un exemple.

Exemple 5.3.1 *Soit à prouver le séquent :*

$$\vdash \exists y((\forall z\ atom(z,y))\wp\ atom(a,y)),\ \exists x(atom(x,f(a))^{\perp} \otimes \exists z\ atom(z,f(x))^{\perp})$$

- *Commençons par remplacer les variables quantifiées par des variables et constantes de Herbrand. Le séquent devient :*

  $$\vdash \exists \mathbf{x_1}((\forall \mathbf{h_1}\ atom(\mathbf{h_1},\mathbf{x_1}))\wp\ atom(a,\mathbf{x_1})),\ \exists \mathbf{x_2}(atom(\mathbf{x_2},f(a))^{\perp} \otimes \exists \mathbf{x_3}\ atom(\mathbf{x_3},f(\mathbf{x_2}))^{\perp})$$

- *La base d'affaiblissement B_{Δ} est vide.*

- *Initialisons la base de recherche $\mathcal{E}_{\Delta}$. Les clauses initiales sont les produits d'axiomes de type id. On peut en obtenir 4 :*

  $$\{\mathbf{x_1} \to f(a),\ \mathbf{x_2} \to \mathbf{h_1}\} \overset{0}{\vdash} atom(\mathbf{h_1},\mathbf{x_1}),\ atom(\mathbf{x_2},f(a))^{\perp}$$
  $$\{\mathbf{x_1} \to f(\mathbf{x_2}),\ \mathbf{x_3} \to \mathbf{h_1}\} \overset{0}{\vdash} atom(\mathbf{h_1},\mathbf{x_1}),\ atom(\mathbf{x_3},f(\mathbf{x_2}))^{\perp}$$
  $$\{\mathbf{x_1} \to f(a),\ \mathbf{x_2} \to a\} \overset{0}{\vdash} atom(a,\mathbf{x_1}),\ atom(\mathbf{x_2},f(a))^{\perp}$$
  $$\{\mathbf{x_1} \to f(\mathbf{x_2}),\ \mathbf{x_3} \to a\} \overset{0}{\vdash} atom(a,\mathbf{x_1}),\ atom(\mathbf{x_3},f(\mathbf{x_2}))^{\perp}$$

  Nous les appellerons respectivement Cl_1, Cl_2, Cl_3 et Cl_4.

- *La recherche proprement dite peut commencer sous forme d'une alternance de phases déterministes (**D.**) et indéterministes (**I.**).*

  D. *On constate tout d'abord que la règle $\forall$ n'est pas applicable à partir des clauses Cl_1 et Cl_2 car la condition relative à la variable quantifiée serait violée. Donc les formules atomiques ne peuvent pas être actives dans des inférences issues des 4 clauses : les formules actives seront nécessairement les littéraux négatifs.*
  Par application de la règle $\exists$, on peut donc remplacer les clauses Cl_2 et Cl_4 par les clauses Cl'_2 et Cl'_4 suivantes :

  $$\{\mathbf{x_1} \to f(\mathbf{x_2})\} \overset{0}{\vdash} atom(\mathbf{h_1},\mathbf{x_1}),\ \exists z\ atom(z,f(\mathbf{x_2}))^{\perp}$$
  $$\{\mathbf{x_1} \to f(\mathbf{x_2})\} \overset{0}{\vdash} atom(a,\mathbf{x_1}),\ \exists z\ atom(z,f(\mathbf{x_2}))^{\perp}$$

  I. *Par application de la règle $\forall$ à la clause Cl'_2, nous pouvons inférer une clause $Cl"_2$ qui sera ajoutée à la base et qui a la forme :*

  $$\{\mathbf{x_1} \to f(\mathbf{x_2})\} \overset{0}{\vdash} \forall z\ atom(z,\mathbf{x_1}),\ \exists z\ atom(z,f(\mathbf{x_2}))^{\perp}$$

  On peut ensuite essayer de produire la conjonction. On constate que ce n'est pas possible à partir de la clause Cl_1 d'un côté et de Cl'_2, $Cl"_2$ ou Cl'_4 de l'autre car $\mathbf{x_1}$ ne peut pas être en même temps instanciée par $f(a)$ et $f(\mathbf{h_1})$.
  Il reste alors deux possibilités :

  - *A partir de Cl_3 et Cl'_2, nous obtenons la clause Cl_5 :*
    $$\{\mathbf{x_1} \to f(a),\ \mathbf{x_2} \to a\} \overset{0}{\vdash} atom(\mathbf{h_1},\mathbf{x_1}),\ atom(a,\mathbf{x_1}),\ atom(\mathbf{x_2},f(a))^{\perp} \otimes \exists z\ atom(z,f(\mathbf{x_2}))^{\perp}$$

  - *A partir de Cl_3 et Cl'_2, nous obtenons la clause Cl_6 :*
    $$\{\mathbf{x_1} \to f(a),\ \mathbf{x_2} \to a\} \overset{0}{\vdash} \forall z\ atom(z,\mathbf{x_1}),\ atom(a,\mathbf{x_1}),\ atom(\mathbf{x_2},f(a))^{\perp} \otimes \exists z\ atom(z,f(\mathbf{x_2}))^{\perp}$$

  D. *L'application du principe de composition immédiate à $\wp$ et $\exists$ à partir de la clause Cl_6 nous permet d'engendrer le but ce qui achève la preuve avec succès.*

3.5 Epuration de la base de recherche

Nous avons vu que l'indéterminisme dans la recherche augmentait avec la taille de la base de recherche. Il est donc utile de disposer de techniques qui permettent de l'épurer des clauses superflues et inutiles. Nous reprenons ici les techniques introduites par [Tammet, 1993] en les complétant. Ces techniques s'inscrivent dans un cadre plus général lié à la résolution [Voronkov, 1992].

3.5.1 Suppression des clauses subsumées

Par superflues, nous entendons les clauses qui sont subsumées par d'autres de la base. Cette notion introduite par [Tammet, 1993] est déjà apparue auparavant mais dans des circonstances particulières. Nous allons en donner maintenant une définition générale.

Définition 5.3.1 *Une clause* $\overset{a_1}{\vdash}\Delta_1$ *subsume une clause* $\overset{a_2}{\vdash}\Delta_2$ *si* $a_1 \geq a_2$ *et si* Δ_2 *est obtenue en ajoutant à* Δ_1 *des formules qui, si* $a_1 = 0$, *sont extraites de* B_Δ.

Par exemple la clause $\overset{0}{\vdash}$ a, b ne subsume pas la clause $\overset{1}{\vdash}$ a, b, $?a$ car l'indicateur d'affaiblissement de la première est strictement inférieur à celui de la seconde. Par contre, elle subsume $\overset{0}{\vdash}$ a, b, $?a$.

3.5.2 Suppression des clauses inutiles

Nous entendons par clauses inutiles des clauses dont on peut déterminer a priori qu'elles n'entreront jamais dans aucune preuve du but. L'indéterminisme inhérent au processus de recherche fait qu'inévitablement, de telles clauses sont produites.
Une première classe de clauses inutiles peut être détectées par une analyse des conditions qui font que deux sous-formules du but peuvent constituer deux formules différentes d'une clause entrant dans une de ses preuves. Prenons par exemple le but suivant : $\vdash$ $(?(a \oplus b)) \otimes a^\perp$, $b^\perp$, $a^\perp$. Supposons que dans la base de recherche, nous ayons trouvé la clause suivante : $\overset{0}{\vdash}$ $(?(a \oplus b)) \otimes a^\perp$, $?(a \oplus b)$, $b^\perp$ (ce cas est tout à fait réaliste). Eh bien ! Nous pouvons supprimer celle-ci de la base de recherche car elle ne peut pas mener au but. En effet, si l'on considère les deux premières formules de la clause, on constate que $?(a \oplus b)$ se retrouve dans le but comme sous-formule de $(?(a \oplus b)) \otimes a^\perp$. Le seul moyen d'aboutir à ce résultat c'est d'utiliser une contraction, ce qui implique que $(?(a \oplus b)) \otimes a^\perp$ doit dans le but être nécessairement dans le champ d'un connecteur ?. Ce n'est pas le cas donc la clause est inutile.
Par contre, toujours pour le même but, considérons la clause $\overset{0}{\vdash}$ $?(a \oplus b)$, $a \oplus b$, $a^\perp$. Dans le but, $a \oplus b$ est une sous-formule de $?(a \oplus b)$ mais on ne peut pas en déduire que la clause est inutile car une contraction a très bien pu permettre d'en arriver à ce résultat.
On peut généraliser de telles considérations par le théorème suivant :

Théorème 5.3.1 *Soit une preuve de* $\vdash \Delta$ *dans le système* $CLL{\downarrow}_\Delta$ *et* F_1 *et* F_2 *deux formules d'une conclusion intermédiaire.*
Si dans $\vdash \Delta$, F_1 *est une sous-formule de* F_2 *ou si* F_1 *et* F_2 *sont dans deux composantes distinctes d'une conjonction ou d'une disjonction additive, alors dans* $\vdash \Delta$, F_1 *et* F_2 *sont incluses toutes deux dans une même sous-formule qui a ? comme connecteur de tête.*

Preuve 5.3.1 *Il faut considérer une propriété plus générale que celle décrite dans le théorème et qui porte sur les sous-formules de toute conclusion intermédiaire et pas seulement de la conclusion finale. On procède ensuite par induction sur la structure des sous-preuves extraites de la preuve globale.*□

Une autre classe de clauses inutiles peut être détectée en comparant le nombre littéraux de chaque sorte présents dans ces clauses par rapport à celui des littéraux de même sorte présents dans le but. On utilise pour cela le théorème :

Théorème 5.3.2 *Soit une preuve de* $\vdash \Delta$ *dans le système* $CLL{\downarrow}_\Delta$. *Pour tout littéral qui n'a pas d'occurrence dans une sous-formule de* $\vdash \Delta$ *de la forme* $?F$, *le nombre de ses occurrences dans toute conclusion intermédiaire ne doit pas dépasser celui de ses occurrences dans* $\vdash \Delta$.

Preuve 5.3.2 *On procède comme pour le théorème précédent.*□

3.6 Conclusion

Si nous comparons les stratégies obtenues avec celles élaborées par T. Tammet, nous devrions trouver logiquement qu'elles sont plus efficaces dans la mesure où le processus de normalisation qui les sous-tend, a été poussé plus loin (voir à ce sujet le chapitre précédent) mais la comparaison est difficile à effectuer : le choix des exemples servant de tests et de l'implantation influe beaucoup sur les performances mesurées. La gestion des substitutions au premier ordre est originale du fait que T. Tammet s'est cantonné dans un cadre propositionnel et que G. Mints qui a étendu ses travaux au premier ordre [Mints, 1993], la traite d'une autre façon : au lieu d'attacher des substitutions aux clauses, il introduit de nouveaux prédicats qui contiennent la même information sous forme implicite. Il semble que les deux traitements, de ce point de vue, se valent mais là aussi, la comparaison est difficile.

4 Implantation

Les stratégies en avant comme en arrière ont été implantées sous forme d'un *démonstrateur automatique de théorèmes dans CLL*. Ce démonstrateur a été écrit en Quintus-Prolog. Il comporte deux options correspondant aux deux sens de construction des preuves. Il permet de prouver des séquents en chaînage avant ou arrière. Il permet aussi de démontrer des formules de logique intuitionniste en les traduisant en logique linéaire selon les règles formulées par [Girard, 1987].

Deuxième partie

Programmation parallèle en logique linéaire

Chapitre 6

De la construction de preuves au calcul de processus

Introduction

L'étude de la construction de preuves en logique linéaire va nous permettre maintenant d'aborder l'utilisation de celle-ci comme paradigme de calcul en programmation. Dans un tel paradigme, les *programmes* sont représentés par des *formules logiques* et leur *exécution* par la *recherche de preuves.*
Une façon d'utiliser les résultats précédents serait d'*élaborer un langage de programmation logique* à partir d'un fragment de LL. Notre connaissance de la construction de preuves aiderait à la conception d'un *interprète* de ce langage qui ne serait pas autre chose qu'un *démonstrateur automatique de théorèmes* dans le fragment logique choisi. C'est cette option qu'ont choisi [Harland and Pym, 1990; Hodas and Miller, 1991; Andreoli and Pareschi, 1990a; Volpe, 1994].
Nos préoccupations se situent à un niveau plus théorique qui est de *modéliser le parallélisme dans un cadre logique sous forme d'un calcul de processus.* Dans ce cadre, les *formules* sont vues comme des *processus* et la *construction de preuves* comme *réduction de processus.* C'est dans cette perspective que se situent les travaux de [Miller, 1992; Kobayashi and Yonezawa, 1992]. Un tel calcul de processus est intéressant en tant qu'outil d'analyse mais il peut rejoindre aussi le premier type de préoccupations mentionnées en constituant la *base d'un langage de programmation logique.*

1 Une question essentielle : le choix d'un fragment logique approprié

Que ce soit pour y concevoir un langage de programmation logique ou un calcul de processus, LL est un cadre trop vaste. Dans le premier cas, il ne permet pas de développer des stratégies efficaces de recherche de preuves et donc de bâtir un interprète du langage réaliste. Dans le deuxième cas, il est trop vaste par rapport à ce qu'on veut y modéliser. Nous verrons par exemple plus loin que pour représenter un calcul de processus, nous devons écarter le connecteur $\wp$ et la négation qui ne trouvent aucune interprétation dans un tel calcul. Le choix d'un fragment logique adapté est donc décisif.
Or, nous savons maintenant à partir d'un fragment quelconque de LL, y normaliser les preuves en s'appuyant sur la permutabilité d'inférences propre à ce fragment et en ayant en perspective un sens de construction donné. Nous savons ensuite élaborer des stratégies pour construire ces preuves normales.
Si nous nous plaçons dans la perspective de l'élaboration d'un langage de programmation logique, le choix du fragment va être le fruit d'un compromis entre les possibilités d'expression souhaitée pour le langage et l'efficacité de l'interprète associé. Les premières sont déterminées par la syntaxe des formules destinées à traduire les clauses du programme et les buts. Et la seconde est fonction du degré de normalisation des preuves qu'il sera possible d'atteindre dans le fragment logique déterminé par cette syntaxe. Prenons par exemple les travaux de [Hodas and Miller, 1991, 1994]. Leur motivation est de raffiner Prolog et λ-Prolog

[Miller and Nadathur, 1988] à partir de la logique linéaire pour exprimer notamment la distinction dans un programme logique entre ressources consommables et ressources infinies. Cette exigence se traduit dans la syntaxe des formules qui représentent les clauses et celles qui représentent les buts du programme. Se trouve ainsi délimité le fragment de LL dans lequel se situent les preuves qui vont représenter l'exécution des requêtes. Il s'agit d'un fragment d'ILL'.
Nous pouvons montrer qu'en appliquant à ce fragment la méthode élaborée précédemment, nous retrouvons au bout du processus de normalisation de preuves et d'élaboration de stratégies de construction de celles-ci, les résultats de Hodas et Miller : nous retrouvons les *preuves uniformes* et l'interprète associé à leur construction.
Nous pourrions effectuer un travail analogue dans le fragment voisin choisi par [Harland and Pym, 1990] pour y développer une extension de Prolog qui est très proche ou encore dans le fragment de CLL qui a servi de base au langage LO de [Andreoli and Pareschi, 1991b]. La méthode peut aider aussi à délimiter de nouveaux fragments de LL appropriés à la définition de langages de programmation logique et elle peut permettre d'y construire des interprètes associés.
Mais dans cette thèse, nos préoccupations ne sont pas directement celles-là. Nous nous proposons d'étudier dans quelle mesure il est possible d'*interpréter la construction de preuves en logique linéaire comme un calcul de processus parallèles, d'une manière qui soit la plus générale possible.*
Il s'agit encore trouver un "bon" fragment pour y définir un "bon" calcul en termes de possibilité d'interprétation. On doit pouvoir y exprimer simplement les mécanismes de communication ainsi qu'un certain nombre d'opérations algébriques sur les processus (composition parallèle, composition alternative, récursivité, restriction...).
L'élaboration du calcul que nous avons nommé CPL (Concurrent Programming Logic), va se faire en deux temps :

1. Définir la *forme syntaxique des séquents* de LL qui exprimeront dans CPL le changement d'état des processus. De ce point de vue, nous n'entrerons pas dans les détails car nous reviendrons plus longuement sur le langage dans le prochain chapitre. Nous nous attarderons uniquement sur les choix essentiels.

2. Dans le cadre syntaxique fixé, nous allons utiliser ensuite la permutabilité d'inférences pour *normaliser les preuves* de façon à leur donner un sens, celui de réductions de processus, ce qui se traduira par une spécialisation du système déductif de LL : nous obtiendrons ainsi un *système formel* qui va constituer l'armature de notre calcul de processus.

2 Les grands choix syntaxiques à l'origine de CPL

Le point de départ de notre interprétation est de considérer que les formules sont des processus et que les opérations logiques sur celles-ci sont des opérations algébriques sur les processus. C'est cette idée qui est à la base de [Miller, 1992] et des travaux qui ont suivi [Kobayashi and Yonezawa, 1994a; Lincoln and Saraswat, 1992; Volpe, 1994].
Se pose ensuite le problème de l'interprétation des séquents linéaires par rapport à la notion de réduction de processus. [Miller, 1992] a mis en évidence deux possibilités.

2.1 Interprétation conjonctive et interprétation disjonctive

Considérons deux formules de logique linéaire P et P' qui représentent deux processus. "P se réduit en P'" peut se traduire de deux façons en logique linéaire :

- le séquent $P \vdash P'$ est prouvable ; on obtient alors une *traduction conjonctive* du calcul de processus ;

- le séquent $P' \vdash P$ est prouvable ; on obtient alors une *traduction disjonctive* du calcul de processus.

La traduction des opérations sur les processus est déterminée par l'option choisie. Si on choisit l'option conjonctive, la composition parallèle est alors représentée par l'opérateur $\otimes$. Par contre, si on choisit l'option disjonctive, elle se traduira par l'opérateur dual $\wp$.

Nous reviendrons plus en détail dans le chapitre prochain sur la représentation des opérations sur les processus en logique linéaire mais à titre d'exemple, donnons la correspondance établie par [Miller, 1992] entre opérations du Π-calcul et opérations de la logique linéaire.

	$\Pi - calcul$	$LL - option$ $conjonctive$	$LL - option$ $disjonctive$
terminaison	0	1	$\perp$
comp.parallèle	|	$\otimes$	$\wp$
comp.alternative	+	&	$\oplus$
récurrence	!	!	?
restriction	(x)	$\exists x$	$\forall x$

Comme, en logique linéaire, le séquent $\Gamma \vdash \Delta$ est équivalent à $\Delta^{\perp} \vdash \Gamma^{\perp}$, les deux traductions proposées sont parfaitement équivalentes. [Milner, 1992; Kobayashi and Yonezawa, 1994a; Volpe, 1994] ont choisi l'option disjonctive pour être plus proche de la forme habituelle de la programmation logique telle qu'elle apparaît avec Prolog et pour pouvoir définir une équivalence de processus reposant sur la sémantique des phases [Girard, 1987]. Nous choisissons comme [Lincoln and Saraswat, 1992], l'option conjonctive car elle nous semble plus naturelle.

2.2 La représentation de la communication

[Miller, 1992] a recours à deux constantes d'ordre supérieur *send* et *get* pour représenter les émetteurs et les récepteurs. La communication est représentée sous une forme *synchrone* grâce à un axiome propre, ce qui implique qu'on ne se situe plus en logique pure mais dans le cadre d'une *théorie linéaire*. Par exemple, la démontrabilité du séquent P_1, $(P_2\ m) \vdash (send\ c\ m\ P_1)$, $(get\ c\ P_2)$ dans cette théorie, exprime le fait que le processus $(send\ c\ m\ P_1)$ envoie le message m sur le canal c au processus $(get\ c\ P_2)$. Aprés réalisation de la communication, les deux processus se retrouvent respectivement dans l'état P_1 et $(P_2\ m)$.
Nous préférons rester dans un *cadre logique pur* où nous représenterons les messages par des formules particulières et nous utiliserons l'implication linéaire pour traduire la communication. Ainsi le séquent $M,\ M \multimap P \vdash P$ est prouvable dans LL et peut être interprété comme la réception d'un message M par le processus $M \multimap P$ qui devient alors le processus P. Remarquons que dans ce séquent, le message M a une existence propre, indépendante de tout émetteur, ce qui implique qu'on se situe dans un cadre de *communication asynchrone*. En effet, lorsque la communication est synchrone, un message est consommé dès qu'il est produit et n'a pas d'existence autonome.
Nous avons choisi de représenter les *messages* par des *formules atomiques* de façon à pouvoir normaliser au maximum les preuves pour les interpréter en termes de réductions de processus. Nous rejoignons ainsi les choix effectués en matière de représentation de la communication par [Kobayashi and Yonezawa, 1994a]; la seule différence est qu'ils ont pris l'option disjonctive : ainsi, ils ne construiront pas comme nous les récepteurs à partir de l'implication linéaire mais à partir de son dual : l'attente du message M par le processus P sera exprimée par $M^{\perp} \otimes P$ au lieu de $M \multimap P$.

2.3 Démontrabilité des séquents et réduction de processus

Quelle que soit l'option choisie, qu'elle soit conjonctive ou disjonctive, et avec les restrictions que nous venons de faire quant à la représentation de la communication, il n'est pas toujours possible d'interpréter un séquent démontrable comme le bilan d'une réduction de processus. Par exemple, le séquent

$$M_1 \multimap (M_2 \multimap P) \vdash M_2 \multimap (M_1 \multimap P)$$

est prouvable en logique linéaire mais il ne peut pas être interprété comme établissant le bilan de la réduction du processus $M_1 \multimap (M_2 \multimap P)$ dans le processus $M_2 \multimap (M_1 \multimap P)$.
En multipliant les exemples, on constate grossièrement que dans *l'option conjonctive*, les seuls séquents qui peuvent traduire des réductions de processus sont ceux qui sont prouvables à l'aide uniquement de *règles d'inférence gauches*. Dans *l'option disjonctive*, c'est le contraire : ce sont les *règles d'inférence*

droite qui ont un sens par rapport aux réductions de processus. Pour faire coïncider construction de preuves et réduction de processus, [Miller, 1992] a choisi d'axiomatiser les règles d'inférence relatives à $\oplus$ (composition alternative) et ! (récurrence) pour en limiter l'application. Dans la théorie ainsi obtenue, preuves et réductions de processus se confondent parfaitement. Dans un même but, [Kobayashi and Yonezawa, 1994a] évitent l'utilisation d'axiomes propres en se situant dans CLL. Ils ne manipulent ainsi que des séquents sans partie gauche donc qui ne sont prouvables qu'avec des règles d'inférence droite ce qui correspond bien à l'option disjonctive choisie.
De cette façon, les preuves prennent un sens plus limité : il n'est plus possible d'exprimer directement la réduction partielle d'un processus P en un processus P' par une preuve du séquent $P' \vdash P$ car seuls les séquents sans partie gauche sont autorisés. On ne peut représenter que la réduction totale d'un processus P par une preuve du séquent $\vdash P$. Mais cela induit une terminaison particulière d'une telle réduction totale : d'un point de vue dynamique, celle-ci se traduit par la construction de bas en haut d'une preuve donc elle se termine lorsque l'on atteint un axiome. Le sens que prend alors cette terminaison, est déterminé par le type de l'axiome utilisé. Or ici, le seul possible est :

$$\frac{}{\vdash \top, \Delta}\ \top$$

On peut l'interpréter comme le fait que le processus représenté par la constante $\top$ a le pouvoir de tuer tous les processus contenus dans le multi-ensemble Δ qui s'exécutent en parallèle avec lui. Comme [Lincoln and Saraswat, 1992], nous n'avons pas cherché à faire coïncider construction de preuves et réduction de processus : la première notion est plus large. Comme nous le verrons, cela laisse une porte ouverte à un élargissement de la notion de réduction partielle d'un processus vers la notion de *réalisation d'un processus par un autre* qui est liée à la sémantique dénotationnelle de CPL.

2.4 La syntaxe du langage de CPL

Une fois ces différents choix effectués, la syntaxe des processus est fixée. Comme nous nous situons au premier ordre, nous supposons disposer d'un ensemble infini récursif $\mathcal{V}$ de variables et d'un ensemble récursif $\mathcal{D}$ d'opérateurs munis de leur arité. Ces deux ensembles nous permettent ensuite de définir de façon inductive, un ensemble de termes $\mathcal{D}[\mathcal{V}]$ qui va contenir les données qui vont être manipulées, échangées par les processus. Les variables seront représentées par les lettres x, y, z, u, v et les termes par t, r, s. Bien entendu, ces lettres pourront être éventuellement indexées.
Nous disposons aussi d'un ensemble $\mathcal{M}$ de prédicats avec leur arité. A partir de $\mathcal{M}$ et de $\mathcal{D}[\mathcal{V}]$, nous pouvons construire de façon inductive l'ensemble $\mathcal{M}[\mathcal{D}, \mathcal{V}]$ des formules atomiques qui vont constituer les messages du langage. Nous utiliserons les lettres M, N, K (éventuellement indexées) comme méta-variables représentant des messages.
Les processus de CPL sont alors des formules de la logique linéaire définis inductivement par la grammaire suivante :

$$P ::= M \mid P_s$$
$$P_s ::= 1 \mid 0 \mid M \otimes P_s \mid M \multimap P_s \mid P_s \otimes P_s \mid P_s \& P_s \mid \forall x P_s \mid \exists x P_s \mid !P_s$$

Dans cette grammaire, P_s représente ce qu'on appelle un *processus strict*, c'est-à-dire un processus qui n'est pas un message. Dans l'ordre de présentation de la grammaire, il peut avoir la forme des constantes de terminaison et d'interruption, d'un émetteur, d'un récepteur, du résultat d'une composition parallèle ou alternative de deux processus, de l'application d'un opérateur de généralisation, de restriction ou de récurrence. Nous reviendrons largement sur la signification de ces opérateurs dans le prochain chapitre.
Nous noterons $\mathcal{P}[\mathcal{M}, \mathcal{D}, \mathcal{V}]$ l'ensemble des processus ainsi définis. S'il n'y a pas d'ambiguïté, nous le noterons simplement $\mathcal{P}$. Nous utiliserons les lettres P, Q, L (éventuellement indexées) comme méta-variables représentant des processus.
Nous allons avoir souvent à manipuler non pas des processus isolés mais des systèmes de processus d'où la définition suivante :

Définition 6.2.1 *Un* système de processus de CPL *est un multi-ensemble de processus de CPL.*

Nous utiliserons les lettres Δ, Γ (éventuellement indexées) pour représenter des systèmes de processus. $!\Gamma$ représentera un système de processus dont tous les éléments sont préfixés par !.

La syntaxe des séquents qui vont exprimer les bilans des réductions, découle de celle des processus et des choix effectués précédemment. L'absence d'interprétation des opérateurs $\wp$ et $^{\perp}$ nous amène automatiquement à nous situer dans le cadre de *la logique linéaire intuitionniste*. Les séquents auront nécessairement la forme $\Gamma \vdash P$ où Γ est un système de processus et P un processus seul.
Nous allons faire une légère restriction, celle de considérer pour P un processus strict d'où la définition :

Définition 6.2.2 *Un séquent de CPL est un séquent dont la partie gauche est un système de processus de CPL et la partie droite un processus strict de CPL.*

3 Le système formel de CPL

La syntaxe du langage de CPL étant fixée, elle détermine le fragment de LL dans lequel le calcul se situe. Mais pour que les preuves puissent être interprétées en terme de réductions de processus, il faut au préalable les normaliser. Il suffit pour cela de mettre en œuvre la méthode proposée précédemment qui consiste à s'appuyer systématiquement sur la permutabilité d'inférences dans le fragment en question.

3.1 LL : extension conservative du fragment logique de CPL

Il s'agit de savoir si tout séquent de CPL prouvable dans LL possède une preuve dont toutes les conclusions intermédiaires sont des séquents de CPL. Cette propriété est essentielle pour établir la complétude du calcul relativement à la logique linéaire. C'est ce que nous allons établir maintenant.

Théorème 6.3.1 *Tout séquent de CPL prouvable admet une preuve sans coupures où toutes les conclusions intermédiaires sont des séquents de CPL.*

Preuve 6.3.1 *Soit $\Gamma \vdash P$ un séquent prouvable de CPL et $\mathcal{P}$ une preuve de ce séquent où les inférences de type $\multimap_L$ sont montées au maximum.*
Soit $\Gamma' \vdash \Delta$ une conclusion intermédiaire de $\mathcal{P}$ située à une profondeur h . Nous allons montrer par induction sur h que $\Gamma' \vdash \Delta$ est un séquent de CPL.
- Le cas de base est trivial.
- Pour le cas d'induction, nous allons distinguer différentes situations en fonction du type de l'inférence I dont $\Gamma' \vdash \Delta$ est un prémisse.

1. I n'est pas de type $\multimap_L$.
   Par hypothèse d'induction, la conclusion de I est un séquent de CPL. La forme des règles d'inférences d'ILL fait que si la conclusion de I est un séquent de CPL, ses prémisses aussi.

2. I est de type $\multimap_L$.
   Par hypothèse d'induction, la conclusion de I est un séquent de CPL. Donc il y a alors deux formes possibles pour I :

$$\frac{\Gamma_1 \vdash P_1' \quad P, \Gamma_2 \vdash P_2'}{P_1' \multimap P, \Gamma_1, \Gamma_2 \vdash P_2'} \multimap_L \text{ ou } \frac{\Gamma_1 \vdash P_1', P_2' \quad P, \Gamma_2 \vdash}{P_1' \multimap P, \Gamma_1, \Gamma_2 \vdash P_2'} \multimap_L \text{ où } P_1' \text{ est un message ou la constante } 1$$

   La première possibilité est conforme à notre attente mais pas la seconde. Montrons qu'il n'est pas possible de l'avoir. Plaçons dans l'hypothèse où nous l'avons. Distinguons alors deux cas selon l'origine de P_1'.

   (a) P_1' est introduit par un axiome de id ou 1_R
   Comme I est montée au maximum, P_1' est introduit dans l'inférence qui la précède immédiatement. Celle-ci est donc de type id ou 1_R ce qui est contradictoire avec le fait qu'elle a deux formules dans la partie droite de sa conclusion.

(b) P'_1 **est introduit par un axiome de** 0_L
Puisque I est montée au maximum, nous pouvons supposer que, dans la branche de $\mathcal{P}$ qui se termine par $\Gamma_1 \vdash P'_1, P'_2$, entre I et l'axiome produisant P'_1, il y a seulement des inférences de type $\multimap_L$ dont les formules actives ne sont pas dans la partie gauche de l'axiome 0_L introduisant P'_1. Donc la constante 0 est une formule de Γ_1 et nous pouvons remplacer la preuve partielle extraite de $\mathcal{P}$ qui a $P'_1 \multimap P, \Gamma_1, \Gamma_2 \vdash P'_2$ comme conclusion par l'axiome $\dfrac{}{P'_1 \multimap P, \Gamma_1, \Gamma_2 \vdash P'_2}\, 0_L$

□

Du fait de la présence de la constante 0, il n'est possible d'obtenir une propriété plus forte qui serait que toute preuve sans coupures d'un séquent de CPL a toutes ses conclusions intermédiaires qui sont des séquents de CPL. Cela ne rend que plus nécessaire le travail de normalisation des preuves afin de pouvoir les interpréter comme réduction de processus.
Examinons maintenant les règles d'inférences utiles pour produire des preuves sans coupures de séquents de CPL dont toutes les conclusions intermédiaires sont aussi des séquents de CPL. La propriété de la sous-formule précisée par le théorème 2.4.4, nous permet de dire qu'il s'agit de :

$$id,\ 0_L,\ 1_L,\ \multimap_L,\ \otimes_L,\ \&_L,\ !_L,\ w!_L,\ c!_L,\ \forall_L,\ \exists_L,\ 1_R,\ \multimap_R,\ \otimes_R,\ \&_R,\ !_R,\ \forall_R,\ \exists_R$$

Toute preuve produite par les règles ci-dessus n'a pas sa conclusion qui est un séquent de CPL. On ne peut donc pas les choisir comme base du système de réduction de processus de CPL. C'est une autre façon de mettre en évidence la nécessité de normaliser les preuves.
Venons en maintenant à l'étude de la permutabilité d'inférences dans les preuves de séquents de CPL dont toutes les conclusions intermédiaires sont des séquents de CPL (nous parlerons de preuves de CPL).

3.2 Permutabilité d'inférences dans les preuves de CPL

En annexe D, nous montrons comment transposer dans LL les résultats sur la permutabilité d'inférences de CLL. En extrayant du tableau D.1 alors obtenu, la partie qui nous est utile, nous obtenons le tableau 6.1. Celui-ci nous renseigne sur les inférences faciles à montrer dans les preuves de CLL et celles

TAB. *6.1* - Permutabilité d'inférences dans les preuves de CPL

$t_2 \backslash t_1$	1_L	$\multimap_L$	$\otimes_L$	$\&_L$	$!_L$	$w!_L$	$c!_L$	$\forall_L$	$\exists_L$	$\multimap_R$	$\otimes_R$	$\&_R$	$!_R$	$\forall_R$	$\exists_R$
1_L													*np*		
$\multimap_L$													*np*		
$\otimes_L$		*np*									*np*		*np*		
$\&_L$													*np*		
$!_L$															
$w!_L$															
$c!_L$		*np*									*np*				
$\forall_L$													*np*		
$\exists_L$								*np*					*np*		*np*
$\multimap_R$		*np*								×	×	×	×	×	×
$\otimes_R$										×	×	×	×	×	×
$\&_R$	–	*np*	–	*np*	*np*	*np*	–	*np*	–	×	×	×	×	×	×
$!_R$	×	×	×	×	*np*			×	×	×	×	×	×	×	×
$\forall_R$								*np*		×	×	×	×	×	×
$\exists_R$										×	×	×	×	×	×

faciles à descendre.

- Les inférences de type 1_L, $\otimes_L$, $c!_L$, $\exists_L$, $\multimap_R$, $\&_R$, $\forall_R$ pourront être descendues sans problème.

- Les inférences de type $!_L$, $w!_L$, $\otimes_R$ et $\exists_R$ pourront être montées sans problème.
- Les inférences de type $\multimap_L$, $\&_L$, $\forall_L$ pourront seulement être bloquées dans leur montée par une inférence de type $!_R$.

Ces informations vont nous permettre maintenant de normaliser les preuves de CPL.

3.3 Normalisation des preuves de CPL

L'exécution des calculs va consister à construire des preuves en chaînage arrière donc on allons effectuer la normalisation dans cette perspective, c'est-à-dire que nous allons monter les contractions et les affaiblissements au maximum (cf paragraphe 3.1.2 du chapitre 2).
Nous allons procéder aux mouvements d'inférences par étapes en commençant par déplacer les inférences qui produisent émetteurs et les récepteurs.

1. Considérons *les inférences de type* $\multimap_L$, *c'est-à-dire celles qui produisent des récepteurs gauches* (situés dans la partie gauche d'un séquent). Elles ont la forme :

$$\frac{\Gamma_2 \vdash M \quad P, \Gamma_1 \vdash P'}{M \multimap P, \Gamma_1, \Gamma_2 \vdash P'} \multimap_L$$

D'après le tableau 6.1, ces inférences peuvent être montées jusqu'à l'inférence produisant leur formule active M. Alors nous pouvons obtenir deux configurations.

(a) la première où M est produit par un axiome de type 0_L :

$$\frac{\dfrac{}{0, \Gamma_1 \vdash M}\, 0_L \quad P, \Gamma_2 \vdash P'}{M \multimap P, 0, \Gamma_1, \Gamma_2 \vdash P'} \multimap_L$$

Elle peut être remplacée par l'axiome :

$$\frac{}{M \multimap P, 0, \Gamma_1, \Gamma_2 \vdash P'}\, 0_L$$

(b) la seconde où M est produit par un axiome de type id

$$\frac{\dfrac{}{M \vdash M}\, id \quad P, \Gamma \vdash P'}{M \multimap P, M, \Gamma \vdash P'} \multimap_L$$

Alors, nous pouvons fusionner les deux inférences en une seule d'où une nouvelle règle RC_L qui peut être décrite ainsi avec la convention d'écriture proposée pour les récepteurs :

$$\frac{P, \Gamma \vdash P'}{M.P, M, \Gamma \vdash P'}\, RC_L$$

Nous aurons donc maintenant des preuves où toutes les inférences de type $\multimap_L$ ont été remplacées par des inférences de type RC_L.

2. Considérons maintenant *les inférences de type* $\otimes_R$ *qui produisent les émetteurs droits.* Elles ont la forme :

$$\frac{\Gamma_1 \vdash M \quad \Gamma_2 \vdash P'}{\Gamma_1, \Gamma_2 \vdash M \otimes P'} \multimap_R$$

D'après la tableau 6.1, ces inférences peuvent être montées jusqu'à celles qui produisent la formule active M. Elles peuvent alors se trouver dans deux configurations possibles.

(a) la première où M est produit par un axiome de type 0_L ;

$$\dfrac{\dfrac{}{0,\ \Gamma_1 \vdash M}\,0_L \qquad \Gamma_2 \vdash P'}{0,\ \Gamma_1,\ \Gamma_2 \vdash M \otimes P'}\,\otimes_R$$

Elle peut être remplacée par l'axiome :

$$\dfrac{}{0,\ \Gamma_1,\ \Gamma_2 \vdash M \otimes P'}\,0_L$$

(b) la seconde où M est produit par un axiome de type id ;

$$\dfrac{\dfrac{}{M \vdash M}\,id \qquad \Gamma \vdash P'}{M,\ \Gamma \vdash M \otimes P'}\,\otimes_R$$

Nous pouvons alors fusionner les deux inférences en une seule d'où une nouvelle règle SD_R que nous pouvons décrire en utilisant la convention de notation pour les émetteurs :

$$\dfrac{P,\ \Gamma \vdash P'}{M,\ \Gamma \vdash \underline{M}.P'}\,SD_R$$

3. Nous allons *monter* ensuite *les contractions au maximum.* Nous avons déjà effectué cette tâche pour normaliser les preuves dans CLL (se reporter pour cela au chapitre 3 de la première partie). Le fait de manipuler maintenant des séquents avec partie gauche et partie droite ne change pas fondamentalement les choses. Les contractions seront bloquées soit par des inférences de type $\otimes_R$, soit par des inférences produisant une de leurs formules actives. Comme dans CLL, après suppression des inférences de type $w!_L$ et $c!_L$ qui se neutralisent, nous avons deux configurations possibles.

   (a) Les contractions sont bloquées par une inférence de type $\otimes_R$. On peut alors les fusionner avec celle-ci d'où la nouvelle règle PAR_R :

   $$\dfrac{\Gamma_1,\ !\Gamma \vdash P_1' \quad \Gamma_2,\ !\Gamma \vdash P_2'}{\Gamma_1,\ \Gamma_2,\ !\Gamma \vdash P_1' \otimes P_2'}\,PAR_R$$

   où Γ_1 et Γ_2 ne contiennent pas de processus récurrent ;
   Même s'il n'y a pas de contractions qui suivent, on peut aussi remplacer les inférences de type $\otimes_R$ par une inférence de type PAR_R en ajoutant éventuellement des affaiblissements.

   (b) Les contractions sont bloquées par une inférence de type $!_L$ produisant une de leur formule active. On les fusionne avec celle-ci d'où la nouvelle règle REC_L :

   $$\dfrac{!P,\ P,\ \Gamma \vdash P'}{!P,\ \Gamma \vdash P'}\,REC_L$$

   Même quand une inférence de type $!_L$ n'est pas suivie par une contraction utilisant sa formule principale, nous pouvons la remplacer par une inférence de type REC_L en insérant un affaiblissement.

4. Il ne reste plus qu'à *monter les affaiblissements jusqu'aux axiomes.* Mais après le remplacement des inférences de type $\multimap_L$ produisant des récepteurs gauches par des inférences de type RC_L et de celles de type $\otimes_R$ produisant des émetteurs droits par des inférences de type SD_R, les seuls axiomes

subsistant dans les preuves sont de type 1_R ou 0_L.
Par fusion des axiomes de type 1_R avec les affaiblissements qui suivent immédiatement, nous obtenons une nouvelle règle $TERM_R$:

$$\frac{}{!\Gamma \vdash 1} \; TERM_R$$

Nous ne modifierons pas les règles restantes 0_L, $\otimes_L$, $\&_L$, $\forall_L$, $\exists_L$, $\&_R$, $!_R$, $\forall_R$, $\exists_R$. Nous allons nous contenter de les renommer afin de mettre en concordance le vocabulaire avec la symétrie du système déductif obtenu et l'interprétation de celui-ci dans le cadre d'un calcul de processus.
Enfin, même si nous pouvons nous passer des coupures dans les preuves, nous conserverons la règle correspondante en la renommant $COMP$.

Après toutes ces transformations successives, nous obtenons un nouveau système d'inférence que nous appellerons le système déductif de CPL et qui est défini par la figure 6.1.
La correction et la complétude de ce nouveau système déductif relativement à la logique linéaire sont garantis par le théorème suivant.

Théorème 6.3.2 *Un séquent de CPL est prouvable en logique linéaire si et seulement si il est prouvable dans le système déductif de CPL.*

Preuve 6.3.2 *La démonstration de la correction s'effectue en montrant que les nouvelles règles sont admissibles en logique linéaire, ce qui est évident vu qu'elles correspondent à des fusions d'inférences de LL.*
La démonstration de la complétude s'effectue par induction sur la structure des preuves de séquents de CPL en logique linéaire. Elle reprend la démarche qui vient d'être utilisée pour élaborer le système déductif propre à CPL.□

3.4 Permutabilité d'inférences dans le système de CPL

La construction de preuves dans le système d'inférence de CPL est le pilier sur lequel reposent à la fois sa sémantique opérationnelle et sa sémantique dénotationnelle.
C'est pourquoi l'étude de la permutabilité d'inférences dans ce système répond à une double exigence :

- du point de vue de la sémantique opérationnelle, il s'agit d'aider à l'*élaboration de stratégies de réduction de processus.* Notamment, il est une forme de stratégies de réduction qui retiendra notre attention dans le chapitre 9 : les *stratégies synchrones.*
- du point de vue de la sémantique dénotationnelle, il s'agit d'aider à *la démonstration de propriétés et de relations sur les processus.*

Bien entendu, il ne s'agit pas de repartir à zéro : il suffit de reprendre les résultats du tableau 6.1 et de les transposer en tenant compte de la correspondance entre règles de LL et règles de CPL. On obtient alors le tableau 6.2.
De ce tableau, nous pouvons tirer le bilan suivant :

- les inférences de type $TERM_L$, SD_L, PAR_L, RES_L, SD_R, RC_R, ALT_R, GEN_R peuvent être descendues au maximum dans les preuves ;
- les inférences de type REC_L, SD_R, PAR_R, RES_R peuvent être montées au maximum dans les preuves ;
- les inférences de type $COMP$, $TERM_L$, RC_L, ALT_L, GEN_L peuvent être bloquées dans leur montée uniquement par des inférences de type REC_R .

L'élimination des coupures est une propriété essentielle pour une logique. Elle est préservée dans le système déductif de CPL ce qui est étroitement lié à la symétrie de celui-ci : toute règle gauche peut être couplée

Règles de communication :

Emission

$$\frac{M,\ P,\ \Gamma \vdash P'}{\underline{M}.P,\ \Gamma \vdash P'}\ SD_L \qquad \frac{\Gamma \vdash P'}{M,\ \Gamma \vdash \underline{M}.P'}\ SD_R$$

Réception

$$\frac{P,\ \Gamma \vdash P'}{M,\ M.P,\ \Gamma \vdash P'}\ RC_L \qquad \frac{M,\ \Gamma \vdash P'}{\Gamma \vdash M.P'}\ RC_R$$

Règles d'opération :

Terminaison

$$\frac{\Gamma \vdash P'}{1,\ \Gamma \vdash P'}\ TERM_L \qquad \frac{}{!\Gamma \vdash 1}\ TERM_R$$

Interruption

$$\frac{}{0,\ \Gamma \vdash P}\ BRK$$

Composition parallèle

$$\frac{P_1,\ P_2,\ \Gamma \vdash P'}{P_1 \otimes P_2,\ \Gamma \vdash P'}\ PAR_L \qquad \frac{\Gamma_1,\ !\Gamma \vdash P_1' \quad \Gamma_2,\ !\Gamma \vdash P_2'}{\Gamma_1,\ \Gamma_2,\ !\Gamma \vdash P_1' \otimes P_2'}\ PAR_R$$

où Γ_1 et Γ_2 ne contiennent pas de processus récurrent ;

Composition alternative

$$\frac{P_1,\ \Gamma \vdash P'}{P_1 \& P_2,\ \Gamma \vdash P'}\ ALT_L \qquad \frac{\Gamma \vdash P_1' \quad \Gamma \vdash P_2'}{\Gamma \vdash P_1' \& P_2'}\ ALT_R$$

Récurrence

$$\frac{!P,\ P,\ \Gamma \vdash P'}{!P,\ \Gamma \vdash P'}\ REC_L \qquad \frac{!\Gamma \vdash P'}{!\Gamma \vdash !P'}\ REC_R$$

Restriction

$$\frac{P[y/x],\ \Gamma \vdash P'}{\exists x P,\ \Gamma \vdash P'}\ RES_L \qquad \frac{\Gamma \vdash P'[t/x]}{\Gamma \vdash \exists x P'}\ RES_R$$

Généralisation

$$\frac{P[t/x],\ \Gamma \vdash P'}{\forall x P,\ \Gamma \vdash P'}\ GEN_L \qquad \frac{\Gamma \vdash P'[y/x]}{\Gamma \vdash \forall x P'}\ GEN_R$$

Pour les règles RES_L et GEN_R, la variable y ne doit pas être libre dans la conclusion de l'inférence.

Règle de composition :

$$\frac{\Gamma_1,\ !\Gamma \vdash P \quad P,\ \Gamma_2,\ !\Gamma \vdash P'}{\Gamma_1,\ \Gamma_2,\ !\Gamma \vdash P'}\ COMP$$

où Γ_1 et Γ_2 ne contiennent pas de processus récurrent ;

FIG. *6.1* - Le système déductif de CPL

TAB. *6.2* - **Permutabilité d'inférences dans le système déductif de CPL**

$t_2 \backslash t_1$	$COMP$	$TERM_L$	SD_L	RC_L	PAR_L	ALT_L	REC_L	GEN_L	RES_L	SD_R	RC_R	PAR_R	ALT_R	REC_R	GEN_R	RES_R
$COMP$														np		
$TERM_L$														np		
SD_L	np											np		×		
RC_L														np		
PAR_L	np											np		np		
ALT_L														np		
REC_L																
GEN_L														np		
RES_L	np							np						np		np
SD_R										×	×	×	×	×	×	×
RC_R	np									×	×	×	×	×	×	×
PAR_R										×	×	×	×	×	×	×
ALT_R	np	–	–	np	–	np	np	np	–	×	×	×	×	×	×	×
REC_R	np	×	×	×	×	×	np	×	×	×	×	×	×	×	×	×
GEN_R	np							np		×	×	×	×	×	×	×
RES_R										×	×	×	×	×	×	×

avec une règle droite. Seule la règle BRK peut être considérée à la fois comme règle gauche et droite car la constante 0 peut être introduite aussi bien à droite qu'à gauche.

Théorème 6.3.3 *Tout séquent prouvable dans le système déductif de CPL peut être prouvé sans la règle $COMP$.*

Preuve 6.3.3 *On remarque tout d'abord que le séquent est nécessairement un séquent de CPL. Puis on peut utiliser le théorème d'élimination des coupures dans LL pour construire une preuve sans coupures de ce séquent dans LL. Ensuite il suffit de reprendre la démarche utilisée pour prouver la complétude du système de CPL relativement à LL.*□

Conclusion

Par restriction sur la syntaxe d'ILL, nous avons établi le langage de CPL qui va nous permettre de représenter les processus et leurs changements d'états. Dans le fragment logique déterminé par ce langage, nous avons utilisé la méthode précédemment élaborée pour normaliser les preuves afin de pouvoir les interpréter comme réductions de processus. Nous avons ainsi débouché sur un système déductif simplifié : le système formel de CPL. Les bases du calcul de processus étant posées, il s'agit maintenant de passer à son étude proprement dite.

Chapitre 7

CPL : un calcul de processus asynchrone

Introduction

Les calculs de processus tels que CSP [Hoare, 1985] et CCS [Milner, 1980] ont joué un rôle historique dans le développement de la théorie du parallélisme. Utiliser le cadre d'une logique existante pour concevoir un calcul de processus, permet d'*amalgamer la théorie du parallélisme avec la programmation logique* telle que nous la concevons ici. Cette démarche est récente et a pu voir le jour suite aux possibilités offertes par la logique linéaire. Elle s'est concrétisée avec les travaux de [Miller, 1992; Kobayashi and Yonezawa, 1992; Lincoln and Saraswat, 1992]. CPL se situe dans leur prolongement. Mais quelles sont ses caractéristiques en tant que calcul de processus ? Nous allons les étudier dans ce chapitre en les comparant à celles des autres.

La syntaxe du langage de CPL a été définie dans le chapitre précédent. Nous allons en donner maintenant la sémantique en nous attachant tout d'abord à la *représentation de la communication.* Un point commun aux calculs de processus basés sur la logique linéaire qui les différencie des calculs classiques (CSP, CCS, Π-calcul), est que la communication y est primitivement *asynchrone* [Milner, 1983]. Nous verrons que CPL permet une souplesse dans les mécanismes à travers lesquels elle s'exprime. Nous étudierons ensuite les diverses *opérations algébriques* possibles sur les processus. A travers des exemples empruntés aux domaines les plus variés, nous pourrons mesurer le *pouvoir de modélisation* du langage de CPL.

Dans le chapitre précédent, nous avons établi le système formel de CPL. Nous verrons qu'une *réduction partielle* d'un processus P en un processus P' peut être définie comme une *preuve* dans ce système formel du séquent $P \vdash P'$ mais nous verrons aussi que toute preuve ne peut pas être interprétée comme une réduction partielle de processus. Les deux notions se confondent seulement dans un cas particulier : lorsque P' est égal à la constante 1 qui est le processus de terminaison. Toute preuve du séquent $P \vdash 1$ sera considérée comme une réduction totale du processus P, ce qui va constituer le point de départ de la sémantique opérationnelle que nous définirons à travers une *relation de transition.*
A partir de cette vision initiale purement logique, nous commencerons par définir une *relation de transition globale* entre systèmes de processus qui fasse bien la distinction entre les actions de communication et les modifications concomitantes dans la structure des processus qui seront exprimées par une relation auxiliaire de *réduction structurelle.* Nous en viendrons ensuite à transformer la relation de transition en une *relation de transition locale* entre processus. Elle portera alors sur des processus et non plus des systèmes de processus.
La sémantique opérationnelle de CPL étant établie, nous reviendrons au niveau logique pour préciser dans quelles conditions il est possible d'interpréter une preuve du séquent $P \vdash P'$ dans le système formel de CPL comme une réduction du processus P dans le processus P'.

1 Le langage de CPL

1.1 Processus et communication

L'entité de base du langage est le *processus* qui est représenté par une *formule* de logique linéaire. Parmi les processus, nous en distinguons certains, les *messages*, qui sont représentés par des *formules atomiques*. Le fait de considérer un message comme un processus, implique que l'on se situe dans un cadre *asynchrone*. En effet, lorsque la communication est synchrone, un message n'a pas d'existence indépendante car dès qu'il est produit, il est consommé [Milner, 1983].

1.1.1 La représentation de la communication

La communication étant asynchrone, voyons plus précisément comment elle s'effectue.

- Un *émetteur* aura la forme $M \otimes P$ où M représente le message à envoyer et P le processus une fois que le message est envoyé. Comme nous le verrons plus loin, le connecteur $\otimes$ traduit par ailleurs la composition parallèle de deux processus. Les deux interprétations sont compatibles puisqu'un message est considéré comme un processus. Pour se rapprocher de la notation classique des actions dans les algèbres de processus [Milner, 1989], nous écrirons $\underline{M}.P$ pour $M \otimes P$. Cela permet aussi de distinguer le symbole $\otimes$ lorsqu'il est utilisé en tant que constructeur d'un émetteur et en tant qu'opérateur de composition parallèle de deux processus quelconques.

- Un *récepteur* aura la forme $M \multimap P$ où M représente le message attendu et P le processus une fois le message reçu. Pourquoi utiliser l'implication linéaire pour construire les récepteurs ? Pour le comprendre, considérons le séquent $M, M \multimap P \vdash P$. Il est prouvable dans CPL et on peut l'interpréter comme la réception d'un message M par le processus $M \multimap P$ qui se retrouve dans l'état P après réception. Nous simplifierons la notation d'un récepteur ainsi : $M.P$.

- On peut représenter ici des formes variées de communication [Banatre, 1991]. Outre *la communication point à point* où un émetteur envoie un message à un récepteur unique et clairement identifié (voir à ce sujet le modèle *Acteur* [Agha, 1986]), on peut exprimer la *communication générative* [Gelernter, 1985] où plusieurs récepteurs peuvent être en compétition pour recevoir le même message. Considérons le séquent :

$$M, M \multimap P_1, M \multimap P_2 \vdash P_1 \otimes P_2.$$

  Il traduit la réception d'un même message M par deux processus. Ce séquent n'est pas prouvable dans CPL. Cela signifie qu'un message ne peut être lu que par un processus, ce qui est caractéristique de la communication générative. Les séquents

$$M, M \multimap P_1, M \multimap P_2 \vdash P_1 \otimes (M \multimap P_2) \quad \text{et} \quad M, M \multimap P_1, M \multimap P_2 \vdash (M \multimap P_1) \otimes P_2$$

  sont prouvables car ils respectent ce principe.
  Nous pouvons aussi représenter dans CPL la *communication par diffusion* où un message peut être lu par un nombre quelconque de récepteurs sans être détruit. Modifions légèrement le séquent de départ que nous venons de prendre pour avoir :

$$!M, M \multimap P_1, M \multimap P_2 \vdash P_1 \otimes P_2.$$

  Ce séquent est maintenant prouvable et traduit un exemple de communication par diffusion.
  Enfin, nous pouvons représenter la *communication par mémoire partagée* où plusieurs processus peuvent lire et écrire dans une mémoire qui est commune [LINDA, 1990]. Considérons le séquent prouvable

$$M, M \multimap (M \otimes P_1), M \multimap P_2 \vdash P_1 \otimes P_2.$$

  Il exprime le fait que le message M est lu par le premier processus puis transmis au second qui le détruit après lecture. Pris à la lettre, cet exemple peut susciter le scepticisme quant à l'utilité de la première réception du message M car le bilan en est nul. Cela nous amène au point suivant.

- Si nous restions dans un cadre propositionnel, la communication serait pauvre et se réduirait à de la synchronisation de tâches car le *passage de valeurs* ne serait pas possible. Au premier ordre, nous

disposons de $\forall$ qui constitue un *opérateur de généralisation*. Placé devant un récepteur, il va le rendre capable de recevoir non pas un seul message mais n'importe lequel aura le format attendu. Par exemple le processus $\forall x(M(f(x)).P)$ est capable de recevoir n'importe quel message de la forme $M(f(t))$ où t est un terme quelconque. Le séquent prouvable

$$M(f(a)),\ \forall x(M(f(x)).P) \vdash P[a/x]$$

traduit la réception d'un message M(f(a)) par le processus $\forall x(M(f(x)).P$.

- A l'opposé, l'*opérateur de restriction* $\exists$ permet comme son nom l'indique, de restreindre la communication. Il permet notamment de donner un caractère privé à certains canaux de communication. Considérons par exemple le séquent

$$\underline{M(x,b)}.P_1, \forall y(M(x,y).P_2) \vdash P_1 \otimes P_2[b/y].$$

Il est prouvable et il traduit la transmission d'un message $M(x,b)$ où x peut être considéré comme le canal de transmission et b comme le contenu du message. Modifions le séquent de la façon suivante :

$$\exists x(\underline{M(x,b)}.P_1), \forall y(M(x,y).P_2) \vdash P_1 \otimes P_2[b/y].$$

Le séquent n'est plus prouvable, ce qui signifie que la transmission n'est plus possible : le canal x sur lequel l'émetteur envoie son message, est devenu un canal privé qui n'est plus accessible au récepteur.

Venons-en maintenant aux diverses opérations possibles sur les processus.

1.1.2 Représentation des opérations sur les processus

- La constante 1 représente le processus de *terminaison* tandis que la constante 0 représente le processus d'*interruption* d'un calcul. La première n'a aucune influence sur les autres processus qui se réduisent en parallèle tandis que la seconde à le pouvoir de tous les détruire.

- $\otimes$ et & représentent respectivement la *composition parallèle* et la *composition alternative* de deux processus. Ainsi la réduction de $P_1 \otimes P_2$ équivaut à la réduction en parallèle de P_1 et de P_2 avec pour ceux-ci la possibilité de communiquer. Par contre la réduction de $P_1 \& P_2$ équivaut à la réduction de P_1 ou de P_2 mais pas des deux à la fois et le choix ne peut pas être arbitraire.

- ! est l'opérateur de *récurrence*. Placé devant un processus P, il lui donne le pouvoir tout à la fois de se dupliquer autant de fois que nécessaire et de se détruire. Cela se traduit respectivement par la démontrabilité des séquents $!P \vdash P \otimes \cdots \otimes P$ et $!P \vdash 1$. Cet opérateur a un pouvoir plus étendu que l'opérateur correspondant du Π-calcul [Milner, 1989] qui ne permet pas au processus qu'il préfixe, de se détruire. Le plus souvent, l'opérateur de récurrence s'applique à un récepteur et une duplication est alors couplée avec une réception. C'est ce que traduit par exemple le séquent prouvable suivant :

$$!M.P, M \vdash (!M.P) \otimes P.$$

Comme nous le verrons dans l'exemple 1.2.1, c'est lui qui permet d'exprimer des *définitions récursives* dans le langage. A ce sujet, CPL se distingue à la fois de ACL [Kobayashi and Yonezawa, 1992] et de Hcc [Lincoln and Saraswat, 1992]. Pour ce qui est de la représentation de la communication et des opérations algébriques précédentes, CPL reprend la syntaxe d'ACL sous une forme conjonctive au lieu d'une forme disjonctive. Par contre, ACL utilise l'opérateur ! seulement devant des messages pour exprimer la possibilité de les lire un nombre infini de fois. Les définitions récursives sont intégrées directement dans les règles de calcul. Ainsi si un processus P est défini par l'équation $P = f(P)$, dans une réduction, on pourra remplacer toute occurrence de P par sa définition $f(P)$. Hcc utilise d'une façon semblable à ACL l'opérateur ! et pour les définitions récursives a recours à une constante d'ordre supérieure. L'exemple du paragraphe 1.2.4 fait bien apparaître les conséquences de cette différence de choix.

Remarque 7.1.1 *On peut étendre le langage de CPL par l'ajout d'un opérateur $\oplus$ de bifurcation qui permet de réduire deux processus en parallèle dans deux copies identiques du même contexte mais sans possibilité de communication entre les deux. Si nous l'avons omis c'est qu'il intervient peu souvent dans les applications.*

1.2 Exemples de modélisation dans CPL

Nous allons considérer différents exemples pour illustrer la modélisation dans le langage de CPL.

1.2.1 Du tampon à une place au partage d'une imprimante

a) Représentation du tampon à une place

La modélisation d'une mémoire tampon est un exemple classique en programmation parallèle [Milner, 1989]. Considérons donc un tampon à une place qui ne peut stocker qu'une unité d'information et disposant d'un port d'entrée a et d'un port de sortie b.

Si nous représentons le tampon par le processus $a.\underline{b}.1$, nous obtenons un tampon qui ne peut fonctionner qu'une fois car en logique linéaire, une formule non précédée de ! ne peut être utilisée qu'une fois dans une preuve. C'est pourquoi le séquent

$$a,\ a,\ a.\underline{b}.1 \vdash b \otimes b$$

qui traduit un chargement du tampon suivi d'une libération deux fois de suite, n'est pas prouvable.
Pour corriger ce défaut, on peut penser un peu rapidement qu'il suffit de choisir le processus $!\ (a.\underline{b}.1)$. Malheureusement, on obtient de cette façon un tampon d'une capacité infinie. Par exemple, le séquent

$$a,\ a,\ !(a.\underline{b}.1) \vdash (\underline{b}.1 \otimes \underline{b}.1) \otimes !(a.\underline{b}.1)$$

qui traduit le chargement de deux unités d'information dans le tampon sans aucune libération, est prouvable.
Pour résoudre le problème, nous allons utiliser un sémaphore s qui va servir de garde pour le canal d'entrée a. Ainsi, le tampon va être représenté par le processus $a.\underline{b}.\underline{s}.1 \otimes !\ (s.a.\underline{b}.\underline{s}.1) \otimes \underline{a}.b.s.1$. Il se divise en trois parties :

- la première partie $a.\underline{b}.\underline{s}.1$ constitue la partie active du processus qui attend une unité d'information sur le canal a ; celle-ci, une fois reçue, peut ensuite être libérée sur le canal b ; l'émission d'un signal sur le canal s rend le sémaphore passant et la zone critique de nouveau accessible ;
- la deuxième partie $!\ (s.a.\underline{b}.\underline{s}.1)$ constitue la partie récurrente qui sert de réservoir pour des utilisations futures ; elle est activée par la réception d'un signal sur le canal s ;
- la dernière partie $\underline{a}.b.s.1$ qui permet de détruire la partie active, est utilisée lorsque l'on n'a plus besoin du processus.

Nous pouvons parfaire la définition précédente en distinguant le sémaphore s qui est privé, des ports a et b qui sont publics. On utilise pour cela l'opérateur de restriction $\exists$. On obtient le processus :

$$P_1 \equiv \exists s\ (a.\underline{b}.\underline{s}.1 \otimes !\ (s.a.\underline{b}.\underline{s}.1) \otimes \underline{a}.b.s.1)$$

Remarque 7.1.2 *Dans cette définition, nous avons commis un abus de langage. La variable s est aussi utilisée pour représenter un message. Mais il faut comprendre ceci comme une abréviation de $m(s)$ où m est un prédicat dont le nom n'a pas d'importance. C'est pourquoi nous ne le faisons pas apparaître afin d'alléger la notation.*

b) Définitions récursives

Plutôt que de chercher à écrire directement la définition d'un processus, ce qui peut paraître pénible au non initié, il peut être utile de passer par l'intermédiaire de définitions récursives comme le permet la syntaxe de beaucoup de calculs de processus (CCS [Milner, 1980], Π-calcul [Milner, 1991; Milner *et al.*, 1992], ACL [Kobayashi and Yonezawa, 1994a]). Ici, le tampon à une place peut être caractérisé par l'équation suivante : $P = a.\underline{b}.P$. En fait, il faut comprendre le signe "=" comme une équivalence logique

et même nous n'avons pas besoin d'une condition si forte : il nous suffit de trouver un processus P tel que $P \vdash a.\underline{b}.P$. La solution proposée ci-dessus vérifie cette condition. Ce qui est intéressant, c'est qu'il est possible de résoudre ce type d'équation d'une façon générale.
Considérons une équation de la forme $P = f(P)$ où P est la seule inconnue et $f(P)$ un processus qui s'exprime en fonction de P. Considérons le processus

$$\exists s\ (f(\underline{s}.1) \otimes (!\ s.f(\underline{s}.1)) \otimes \overline{f(\underline{s}.1)})$$

où $\overline{f(\underline{s}.1)}$ vérifie : $f(\underline{s}.1),\ \overline{f(\underline{s}.1)} \vdash 1$. On peut montrer que ce processus vérifie : $P \vdash f(P)$.

c) Composition de processus

Revenons maintenant au processus P_1 modélisant le tampon à une place. A l'aide des opérateurs de CPL, il est facile de construire des processus plus complexes à partir de P_1. Donnons-en quelques exemples.

- Si on veut prendre en compte la valeur de l'information conservée dans le tampon, on peut le faire en considérant le processus suivant :

$$P_1' \equiv \exists s((\forall x\ a(x).\underline{b(x)}.\underline{s}.1) \otimes !\ (\forall x\ (s.a(x).\underline{b(x)}.\underline{s}.1) \otimes (\forall x\ \underline{a(x)}.b(x).s.1)).$$

- Par composition parallèle de n processus P_1 et en masquant les n-1 canaux de liaisons entre ceux-ci, nous obtenons un tampon à n places dont la représentation peut être définie ainsi :

$$P_n \equiv \exists c_1 \cdots c_{n-1}\ (P_1[c_1/b] \otimes P_1[c_1/a][c_2/b] \otimes \cdots \otimes P_1[c_{n-1}/a])$$

- Considérons une imprimante simplifiée qui dispose de deux ports : un port d'entrée *fich* par lequel arrive les fichiers à imprimer et un port de sortie *impr* par lequel il envoie les documents imprimés. Une imprimante ne peut traiter qu'un fichier à la fois ce qui correspond tout à fait au principe du tampon à une place. D'où sa représentation dans CPL :

$$P_{imprim} \equiv P_1'[fich/a][impr/b].$$

  Nous pouvons maintenant intégrer ce processus dans un système où interviennent des utilisateurs qui sont en compétition pour l'usage de l'imprimante. Un utilisateur k peut être représenté simplement par le processus :

$$P_k \equiv !\ ((\forall x\ k(x).\underline{fich(x)}.1) \& P_k').$$

  L'utilisateur k peut soit demander d'imprimer le fichier x qu'il reçoit sur le canal k, soit effectuer une tâche qui n'est pas précisée ici et qui est représentée par le processus P_k'.
  Un système formée de deux utilisateurs partageant une imprimante pourrait alors être représenté ainsi :

$$\exists fich\ imp\ (P_1 \otimes P_2 \otimes P_{imprim}).$$

Nous pouvons de cette façon représenter des systèmes encore plus sophistiqués tels le "jobshop" de [Milner, 1989].

1.2.2 Un programme impératif

Nous nous proposons d'écrire un processus P_{fact} qui modélise un programme impératif qui calcule la factorielle f de n'importe quel entier n.

a) Préliminaires

Nous avons choisi pour le calcul l'algorithme suivant :

```
debut
   entrer(n) ;
   f := 1 ;
   tantque n > 0
   faire f := f × n ;
         n := n − 1
   fintantque
   retourner(f)
fin
```

Pour mettre en œuvre cet algorithme, nous devons faire appel à des opérations arithmétiques élémentaires : *zero* et *unite* qui fournissent respectivement les nombre 0 et 1, *decr* qui décrémente de 1 chaque entier et *mult* qui multiplie deux entiers. Nous aurons aussi besoin d'une opération booléenne *egal* qui détermine si deux entiers sont égaux ou pas.
Nous allons supposer que toutes ces opérations sont fournies par un processus parallèle à P_{fact} que nous appellerons P_{arith}, chaque opération étant effectuée par un sous-processus de P_{arith}. P_{arith} se présente donc sous la forme :

$$P_{zero} \otimes P_{unite} \otimes P_{decr} \otimes P_{mult} \otimes P_{egal}$$

Si nous voulons que le processus P_{fact} soit opérationnel quelle que soit l'implantation des entiers et des opérations choisie dans P_{arith}, il ne faut pas préjuger de celle-ci dans l'écriture de P_{fact}. Nous obtiendrons ainsi une qualité essentielle en programmation : *la modularité* de P_{fact}.

b) L'écriture de P_{fact}

P_{fact} doit être capable de recevoir sur un canal d'entrée que nous appellerons *fact in* un message *(fact in : n)* et de retourner après réduction le message *(fact out : n f)* sur un canal de sortie *fact out.*

Remarque 7.1.3 *Dans CPL, il n'y a pas de syntaxe figée pour les messages mais nous prendrons l'habitude de les représenter en deux parties séparées par ":", la première représentant le nom du canal par lequel le message transite, et la seconde le contenu du message. Quand le processus a un canal d'entrée et un canal de sortie privilégiés, nous désignerons ceux-ci par le nom du processus (*fact *ici) suivi de* in *ou de* out *pour préciser s'il s'agit de l'entrée ou de la sortie. Enfin, nous répéterons dans le message de sortie le contenu du message d'entrée correspondant pour éviter des erreurs de destinataire.*

Si nous voulons utiliser P_{fact} pour plusieurs calculs de factorielles, il faut qu'il soit récurrent. Ensuite, nous constatons que l'algorithme utilise les variables locales n et f pour stocker les résultats intermédiaires. Nous les représenterons par deux canaux privés n_c et f_c. La phase d'initialisation peut être représentée ainsi :

$$\begin{array}{l}
P_{fact} = !\ \forall\ \mathrm{n}\ (\mathrm{fact\ in : n}). \\
\qquad \otimes\ \underline{(\mathrm{zero\ in :})}. \\
\qquad\quad \underline{1} \\
\qquad \otimes\ \underline{(\mathrm{unite\ in :})}. \\
\qquad\quad \underline{1} \\
\qquad \otimes\ \exists\ n_c\ f_c\ \ \otimes\ \forall\ \mathrm{u}\ (\mathrm{unite\ out : u}). \\
\qquad\qquad\qquad\qquad \underline{(f_c : \mathrm{u})}. \\
\qquad\qquad\qquad\qquad\ \ \underline{1} \\
\qquad\qquad\qquad \otimes\ \underline{(n_c : \mathrm{n})}. \\
\qquad\qquad\qquad\qquad\ \ \underline{1} \\
\qquad\qquad\qquad \otimes\ \forall\ \mathrm{z}\ (\mathrm{zero\ out : z}). \\
\qquad\qquad\qquad\qquad\ \ P_{fact-int}
\end{array}$$

Remarque 7.1.4 *Pour rendre lisible la définition des processus, nous avons remplacé ci-dessus les parenthèses par des tabulations et nous avons répété l'opérateur ⊗ devant chacun de ses opérandes.*
Nous écrivons aussi une action par ligne et lorsque plusieurs quantificateurs du même type se suivent, nous ne notons que le premier. Désormais, nous utiliserons cette forme de présentation des processus lorsque leur écriture sera un peu longue.

Ceci étant dit, l'écriture ci-dessus signifie que le processus P_{fact} attend de recevoir en entrée la valeur du nombre n dont il va chercher à calculer la factorielle.
Ensuite, il envoie en parallèle deux requêtes au processus P_{arith} sur les canaux *zero in* et *unite in* de façon à récupérer les valeurs de 0 et de 1 telles qu'elles ont pu y être implantées. Il les recevra sur les canaux *zero out* et *unite out* et elles viendront instancier les variables z et u.
Le caractère privé des canaux f_c et n_c est garanti par l'utilisation de l'opérateur de restriction $\exists$. Le lancement du calcul proprement dit s'effectue par l'envoi des valeurs initiales 1 et n sur les deux canaux privés, au processus $P_{fact-int}$ qui va effectuer le calcul proprement dit. $P_{fact-int}$ peut être défini ainsi :

$$
\begin{array}{l}
P_{fact-int} = !\ \forall\ \mathrm{f\ m}\ (f_c : \mathrm{f}). \\
\qquad (n_c : \mathrm{m}). \\
\qquad \underline{(\mathrm{egal\ in} : (\mathrm{m,z}))}. \\
\qquad \&\ (\mathrm{egal\ out} : (\mathrm{m,z})\ \mathrm{true}). \\
\qquad\quad \underline{(\mathrm{fact\ out} : \mathrm{n\ f})}. \\
\qquad\quad \overline{1} \\
\qquad \&\ (\mathrm{egal\ out} : (\mathrm{m,z})\ \mathrm{false}). \\
\qquad\quad \otimes\ \underline{(\mathrm{mult\ in} : (\mathrm{f,m}))}. \\
\qquad\qquad \overline{\forall\ \mathrm{f'}\ (\mathrm{mult\ out} : (\mathrm{f,m})\ \mathrm{f'})}. \\
\qquad\qquad\quad \underline{(f_c : \mathrm{f'})}. \\
\qquad\qquad\quad \overline{1} \\
\qquad\quad \otimes\ \underline{(\mathrm{decr\ in} : \mathrm{m})}. \\
\qquad\qquad \overline{\forall\ \mathrm{m}\ (\mathrm{decr\ out} : \mathrm{m\ m'})}. \\
\qquad\qquad\quad \underline{(n_c : \mathrm{m'})}. \\
\qquad\qquad\quad \overline{1}
\end{array}
$$

L'utilisation de l'opérateur de récurrence s'explique par le fait que nous avons une définition récursive.
L'activation de $P_{fact-int}$ s'effectue par la réception des valeurs de f et m sur les canaux f_c et n_c.
Le calcul commence par un test pour savoir si la valeur de m est nulle ou non (cela correspond au test de la boucle "tant que" dans l'algorithme).
Si elle est nulle, le calcul est terminé et le résultat final est la valeur de f qui est envoyée sur la canal public *fact out*. Sinon, le processus transmet les valeurs de f et de m au processus P_{arith} pour qu'il effectue les calculs $f \times m$ et $m - 1$.
Le processus $P_{fact-int}$ est ensuite relancé lorsqu'il reçoit les résultats de ces calculs sur les canaux *mult out* et *decr out* et que ces résultats sont transmis sur f_c et n_c.

c) L'exécution du programme

Il faut d'abord modéliser l'agent qui va fournir la valeur d'entrée n du programme et récupérer la valeur de sortie f. Cela peut se faire sous forme d'un processus complémentaire ou co-processus Q_{fact} qui aura la forme :

$$\underline{(\mathrm{fact\ in} : \mathrm{n})}.\forall f((\mathrm{fact\ out} : \mathrm{n\ f})).1.$$

Q_{fact} aura pour rôle d'envoyer sur le canal *fact in* la valeur n de l'entier dont on cherche à connaître la factorielle. Puis il attendra le résultat du calcul qu'il recevra par le canal *fact out* à travers la variable f.
L'exécution du programme *fact* sera alors représenté par la réduction totale de $P_{fact} \otimes P_{arith} \otimes Q_{fact}$ c'est-à-dire la preuve du séquent $P_{fact} \otimes P_{arith} \otimes Q_{fact} \vdash 1$.
A ce sujet, il est intéressant de remarquer que la réponse attendue n'est pas seulement de savoir si ce dernier est prouvable mais c'est la preuve elle-même car elle seule nous donnera la valeur de $!n$ que nous cherchons.

A travers l'exemple qui vient d'être présenté, on peut appréhender certaines facettes du langage notamment la possibilité d'y exprimer des définitions récursives et la modularité puisque nous avons réussi à rendre totalement indépendant le calcul de la factorielle de l'implantation de l'arithmétique nécessaire. Mieux, nous disposons avec P_{fact} d'un *processus générique*, c'est-à-dire qu'il peut en engendrer d'autres par simple composition parallèle avec un autre processus. Il faut bien entendu que les canaux de communication correspondent mais le processus qui est adjoint, ne manipulera plus nécessairement des entiers. Il peut par exemple manipuler des matrices. On retrouve ces caractéristiques dans ACL [Kobayashi and Yonezawa, 1992] et Hcc [Lincoln and Saraswat, 1992].

1.2.3 Représentation en programmation orientée objets

Nous avons choisi un exemple classique [Meyer, 1988], celui de la représentation des polygones.

a) Représentation des objets et des routines

Considérons en programmation orientée objets, la classe *polygone* définie par un attribut *sommets* qui est une liste de points constituant les sommets du polygone et par des routines qui sont des procédures permettant de modifier l'état d'un objet de la classe comme *translation* et *rotation* et des fonctions permettant de créer des objets d'autres classes comme la fonction *perimetre*.
Un objet de la classe *polygone* peut être représenté par un message de la forme $(sommets : (id, [P_1, \cdots, P_n]))$ qui décrit l'état de l'objet ; ici la liste de points $[P_1, \cdots, P_n]$ décrit la position du polygone *id*. Si la classe avait eu d'autres attributs, un objet aurait été représenté par une conjonction multiplicative de formules atomiques, chacune représentant un attribut particulier.
L'ensemble des routines attachées à la classe *polygone* est représenté par un processus $P_{routines}$ qui est la composition parallèle de processus récurrents, chacun permettant de réaliser une procédure ou une fonction attachée à la classe de l'objet. Ici, nous aurions donc :

$$P_{routines} \equiv P_{translation} \otimes P_{rotation} \otimes P_{perimetre}$$

Détaillons la définition possible d'un des processus représentant les routines, $P_{translation}$ par exemple.

$$\begin{array}{l} P_{translation} = \exists \text{ add } \otimes ! \forall \text{ v id (translation : (v,id)).} \\ \qquad\qquad \forall \text{ l (sommets :(id,l)).} \\ \qquad\qquad\quad \underline{\text{(add in : (l,v))}}. \\ \qquad\qquad\quad \forall \text{ l' (add out : (l,v) l').} \\ \qquad\qquad\qquad \underline{\text{(sommets :(id,l'))}}. \\ \qquad\qquad\qquad \text{1} \\ \qquad\qquad \otimes \; P_{add} \end{array}$$

La procédure est activée par la réception d'un message sur le canal *translation* qui indique le vecteur v de la translation et le polygone *id* à translater. Comme la procédure a pour fonction de modifier l'attribut *sommets* du polygone *id*, celui-ci est fourni en entrée. L'opération mathématique qui consiste à ajouter le vecteur v à chacun des sommets de la liste l est effectuée par un processus récurrent auxiliaire P_{add}. En sortie de la procédure est retourné le nouvel état de l'attribut *sommets* du polygone *id*.

b) Modélisation de l'héritage

L'*héritage* est un aspect important de la programmation orientée objets [Meyer, 1988]. Voyons comment il peut être capté dans CPL. Supposons que nous voulions définir une classe *rectangle* comme descendante de la classe *polygone*. Un objet de cette classe va hériter des attributs de la classe *polygone* auxquels pourront s'adjoindre des attributs propres : *longueur* et *largeur* par exemple. D'où une représentation possible d'un objet de la classe rectangle : $(sommets : (id, [P_1, P_2, P_3, P_4])) \otimes (longueur : (id, l_1)) \otimes (largeur : (id, l_2))$. La classe *rectangle* hérite des routines de la classe *polygone* auxquelles peuvent s'ajouter des routines propres telles que la fonction *centre* qui détermine la position du centre d'un rectangle. Dans CPL, ceci s'effectue simplement par la composition parallèle d'un processus récurrent P_{centre} avec ceux représentant les routines attachées à la classe *polygone*.

[Andreoli and Pareschi, 1990a] et [Kobayashi and Yonezawa, 1994c] ont exploité cette propension des calculs basés sur la construction de preuves en logique linéaire à modéliser la programmation orientée objets en développant des applications dans cette direction.

1.2.4 La modification de la structure des processus

La motivation principale qui est à l'origine du Π-calcul [Milner *et al.*, 1992], était d'exprimer la possibilité de modifier la structure des processus. Nous allons montrer que CPL le permet aussi en considérant l'exemple de la gestion de la topologie d'un système distribué.
Les processus constituant du système, sont sensés être des processus fonctionnels c'est-à-dire avec un canal d'entrée et un canal de sortie. Un processus superviseur $P_{connect}$ reçoit en entrée un graphe g décrivant la topologie souhaitée des canaux de communication. Un arc du graphe g a comme point de départ le nom d'un processus et comme point d'arrivée le nom d'un autre processus dont on veut que le canal

d'entrée coïncide avec le canal de sortie du premier. $P_{connect}$ va avoir recours à un autre processus P_{graph} susceptible de fournir les opérations classiques sur les graphes : *graphe-vide* qui indique si un graphe est vide ou pas, *arc* qui extrait un arc d'un graphe et *supprim-arc* qui supprime un arc d'un graphe.

a) Ecriture du processus $P_{connect}$

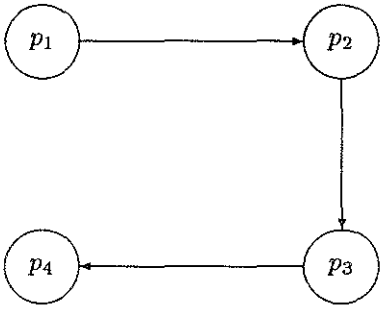
Nous ne préjugerons de la forme d'implantation de g dans la définition de $P_{connect}$ de façon à en conserver la modularité. Nous proposons alors cette définition :

```
P_connect =! ∀ g (connect in : g).
               (graphe-vide in : g).
               & (graphe-vide out : g true).
                 1
               & (graphe-vide out : g false).
                 (arc in : g).
                 ∀ x y (arc out : g (x,y)).
                       ⊗ ∀ u v (x out : u v).
                               (y in : v).
                               1
                       ⊗ (supprim-arc in : (g,(x,y))).
                         ∀ g' (supprim-arc out : (g,(x,y)) g').
                               (connect in : g').
                               1
```

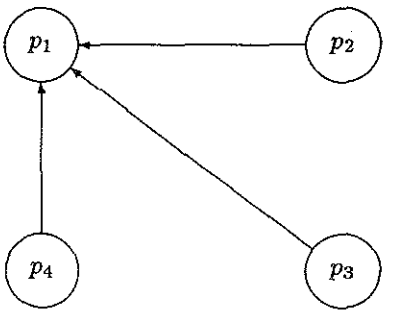
Commentons cette définition.
On teste d'abord si le graphe g reçu en entrée est vide. Si c'est le cas, le calcul est terminé sinon on extrait un arc (x, y) de celui-ci. On crée ensuite un processus qui va faire la jonction entre le canal de sortie du processus x et le canal d'entrée du processus y et en parallèle, on supprime du graphe g, l'arc (x, y). On obtient un graphe g' avec lequel on reprend le calcul.

b) Utilisation de $P_{connect}$

Le processus $P_{connect}$ va être utilisé par la mise en présence avec un co-processus $Q_{connect}$. La réduction partielle du processus $P_{connect} \otimes Q_{connect} \otimes P_{graph}$ va représenter la mise en œuvre de $P_{connect}$. La réduction n'est pas totale car il y a au cours de celle-ci, création de processus qui vont assurer la jonction entre canaux de sortie de certains processus et canaux d'entrée d'autres.
Donnons quelques exemples de co-processus $Q_{connect}$ qui peuvent être utilisés.
Si l'on prend pour $Q_{connect}$ le message *(connect in :* $[(p_1, p_2), (p_2, p_3), (p_3, p_4)]$*)*, nous obtiendrons après réduction de $P_{connect} \otimes Q_{connect} \otimes P_{graph}$, trois nouveaux processus qui permettront de connecter 4 processus p_1, p_2, p_3 et p_4 entre eux pour former un pipe-line.



Si l'on prend maintenant pour $Q_{connect}$ le message *(connect in :* $[(p_2, p_1), (p_3, p_1), (p_4, p_1)]$*)*, nous obtiendrons la configuration suivante .



Une fois de plus, nous trouvons avec cet exemple une illustration de la modularité du langage en ce sens que $P_{connect}$ est utilisable quelle que soit l'implantation choisie pour les graphes et les opérations associées.
Comme dans l'exemple précédent, *modularité* rime avec *généricité*.
En effet composons le processus $P_{connect}$ en parallèle avec le processus P_{comp} défini ainsi :

$$P_{comp} \equiv !\forall f\ g\ (comp\ in :\ f\ g).\underline{(connect\ in\ [(f,g)])}.1$$

Nous obtenons un processus $P_{connect} \otimes P_{comp}$ qui permet de réaliser la composition de deux processus fonctionnels quelconques.

Nous avions vu en définissant la syntaxe de CPL que nous avions dû restreindre la forme des messages de telle sorte qu'il n'est pas possible de transmettre des processus dans la communication. Mais cet exemple montre que nous pouvons d'une certaine façon remplacer la transmission de processus par la transmission de canaux. Ceci n'est pas possible dans ACL du fait de l'usage très limité de l'opérateur ! et cela a motivé son extension à l'ordre supérieur sous la forme de HACL [Kobayashi and Yonezawa, 1994b]. Hcc [Lincoln and Saraswat, 1992], de la même façon, a besoin de l'ordre supérieur pour représenter la mobilité dans la structure des processus.

1.3 Conclusion

Dans cette première section, nous avons pu montrer à travers des exemples, les possibilités d'expression du langage de CPL. Ses points forts sont : la *diversité des mécanismes de communication*, la possibilité d'exprimer des *définitions récursives*, la *modularité* des programmes qui va de pair ici avec la *généricité*. En cela, il est proche de ACL [Kobayashi and Yonezawa, 1994a] dont il a repris la syntaxe mais sous une forme conjonctive. Néanmoins la syntaxe de ACL est un peu plus restreinte en ce sens que les opérations de composition alternative et de généralisation s'appliquent seulement à des récepteurs alors qu'elles s'appliquent à des processus quelconques dans CPL. Cela n'empêche pas toutefois de pouvoir exprimer dans ACL les trois premiers exemples qui viennent d'être décrits. Par contre, l'usage limité de ! dans ACL ne lui permet pas d'exprimer la mobilité dans la structure des processus. Pour y remédier, le calcul sera étendu à l'ordre supérieur [Kobayashi and Yonezawa, 1994b] en intégrant le λ-calcul.
Sur un autre point, CPL présente un choix différent lié à la conception de la terminaison d'un calcul. Lorsque l'on veut représenter dans ACL un processus obtenu par composition parallèle de plusieurs processus, il faut savoir à l'avance lequel terminera le dernier. Sinon, vu que la terminaison du calcul va se faire brutalement par le processus $\top$, le premier qui va terminer interrompra les autres avant qu'ils ne parviennent à leur terme. Dans CPL, nous n'avons pas ce problème car la terminaison du calcul peut s'effectuer soit en douceur par la réduction de tous les processus, soit brutalement comme dans ACL.
Si l'on compare maintenant le langage de CPL avec celui de Hcc [Lincoln and Saraswat, 1992], on constate que ce dernier a un pouvoir d'expression très lié à la programmation par contraintes. La différence essentielle réside dans l'utilisation de l'opérateur !. Dans Hcc, il est employé seulement devant des formules atomiques pour indiquer qu'il s'agit de contraintes. De cette façon, se trouve exprimé le caractère monotone d'un ensemble de contraintes qui ne peut que croître et être interrogé à volonté. Par contre, il ne

permet pas dans Hcc d'exprimer des définitions récursives de processus. On a recours pour cela à une constante d'ordre supérieur.

2 Sémantique opérationnelle

Ayant opté pour une version conjonctive de la traduction du calcul en logique linéaire, nous savons que "le processus P se réduit au processus P'" va être représenté par "le séquent $P \vdash P'$ est prouvable".

2.1 Réduction de processus et preuves de CPL

A travers des exemples, nous allons voir que les deux notions ne se recouvrent pas complètement.

2.1.1 Une preuve qui n'est pas une réduction de processus

Considérons par exemple le tampon à capacité infinie du paragraphe 2.2.1. Il était représenté par le processus $!a.\underline{b}.1$. Montrons que le séquent $!\underline{b}.a.1 \vdash !a.\underline{b}.1$ est prouvable dans CPL.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{!\underline{b}.a.1 \vdash 1}\,TERM_R}{b,\ !\underline{b}.a.1 \vdash \underline{b}.1}\,SD_R}{1,\ b,\ !\underline{b}.a.1 \vdash \underline{b}.1}\,TERM_L}{a,\ a.1,\ b,\ !\underline{b}.a.1 \vdash \underline{b}.1}\,RC_L}{a,\ \underline{b}.a.1,\ !\underline{b}.a.1 \vdash \underline{b}.1}\,SD_L}{a,\ !\underline{b}.a.1 \vdash \underline{b}.1}\,REC_L}{!\underline{b}.a.1 \vdash a.\underline{b}.1}\,RC_R}{!\underline{b}.a.1 \vdash !a.\underline{b}.1}\,REC_R$$

La preuve ci-dessus représente la transformation d'un tampon à capacité infinie initialement plein en un tampon à capacité infinie initialement vide. Evidemment, il est impossible de considérer celle-ci comme une réduction car il semble difficile de vider un tampon infini en une suite d'étapes finies. Le sens que peut avoir une telle preuve apparaîtra lorsque nous aborderons la sémantique dénotationnelle de CPL.

2.1.2 Une preuve qui est une réduction de processus

Montrons maintenant que le séquent $a \otimes !a.\underline{b}.1 \vdash \underline{b}.1 \otimes !a.\underline{b}.1$ est prouvable dans CPL.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\underline{b}.1,\ !a.\underline{b}.1 \vdash \underline{b}.1 \otimes !a.\underline{b}.1}\,ID}{a,\ a.\underline{b}.1,\ !a.\underline{b}.1 \vdash \underline{b}.1 \otimes !a.\underline{b}.1}\,RC_L}{a,\ !a.\underline{b}.1 \vdash \underline{b}.1 \otimes !a.\underline{b}.1}\,REC_L}{a \otimes !a.\underline{b}.1 \vdash \underline{b}.1 \otimes !a.\underline{b}.1}\,REC_L$$

Pour simplifier la preuve, nous avons introduit une nouvelle règle ID qui se présente ainsi :

$$\dfrac{}{P_1, \cdots, P_n \vdash P_1 \otimes \cdots \otimes P_n}\,ID$$

Il est facile de prouver que cette règle est dérivable dans CPL. Maintenant, il est clair que nous pouvons interpréter cette preuve comme une réduction de processus : elle traduit le chargement dans le tampon d'une unité d'information.

2.1.3 Rôle des règles d'inférence gauches

Si l'on observe les règles utilisées dans la dernière preuve, on constate que contrairement à la précédente, les seules règles utilisées sont des règles gauche sauf ID. Nous pourrions en tirer rapidement la conclusion que les preuves de CPL se confondent avec des réductions partielles de processus seulement lorsqu'elles utilisent des règles gauche. Malheureusement, les choses ne sont pas si simples.
Considérons par exemple le séquent $\exists a\ b\ (a \otimes !a.\underline{b}.1) \vdash \exists a\ b\ (\underline{b}.1 \otimes !a.\underline{b}.1)$. Il est prouvable et peut être interprété comme le chargement d'une unité d'information dans le tampon mais par le biais d'un canal privé car tous les ports du tampon sont maintenant masqués. Or toute preuve du séquent utilise deux fois de suite la règle droite RES_R, ce qui contredit l'hypothèse que nous avions pu faire.
La frontière entre preuves représentant des réductions partielles et preuves ne le faisant pas, étant difficile à définir a priori, nous avons préféré partir d'un cas où cette frontière est claire, celui des *réductions totales*, pour bâtir une sémantique opérationnelle. Celle-ci une fois établie, nous pourrons ensuite en déduire une notion très précise de réduction partielle et la comparer avec la relation de déduction logique.

2.1.4 Réduction totale d'un processus

La notion de réduction totale peut être définie comme celle d'une réduction d'un processus quelconque au processus de terminaison 1,ce qui peut se formaliser ainsi :

Définition 7.2.1 *Une* réduction totale de CPL *est une preuve dans le système d'inférence de CPL dont la conclusion est un séquent de la forme* $\Gamma \vdash 1$. *Nous disons alors que le système de processus* Γ *est* totalement réductible. *Si* Γ *se réduit à un seul processus, nous disons que ce processus est totalement réductible.*

Le théorème qui suit, vient justifier notre démarche consistant à partir de la notion de réduction totale plutôt que de celle de réduction partielle.

Théorème 7.2.1 *Mis à part la règle* $TERM_R$, *une réduction totale sans coupures utilise seulement des règles gauches.*

Preuve 7.2.1 *Elle s'effectue simplement par induction sur la structure des réductions en observant la forme syntaxique des règles du système déductif de CPL.* □

Illustrons la définition d'une réduction totale en reprenant l'exemple du tampon à capacité infinie.

Exemple 7.2.1 *Considérons le système de processus formé du tampon* $!a.\underline{b}.1$, *d'un émetteur* a *dont la fonction est d'approvisionner le tampon avec une unité d'information et d'un récepteur* $b.1$ *qui est prêt à recevoir une unité d'information de la part du tampon. Montrons que ce système est totalement réductible c'est-à-dire que le séquent de CPL* $!a.\underline{b}.1,\ a,\ b.1 \vdash 1$ *est prouvable.*

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{!a.\underline{b}.1 \vdash 1}\,TERM_R}{!a.\underline{b}.1,\ 1 \vdash 1}\,TERM_L}{!a.\underline{b}.1,\ b,\ b.1 \vdash 1}\,RC_L}{!a.\underline{b}.1,\ b,\ 1,\ b.1 \vdash 1}\,TERM_L}{!a.\underline{b}.1,\ \underline{b}.1,\ b.1 \vdash 1}\,SD_L}{!a.\underline{b}.1,\ a.\underline{b}.1,\ a,\ b.1 \vdash 1}\,RC_L}{!a.\underline{b}.1,\ a,\ b.1 \vdash 1}\,REC_L
$$

La preuve ci-dessus constitue donc une réduction totale qui peut être décrite comme le chargement d'une unité d'information dans le tampon suivi de la libération de cette unité par le tampon. Si nous lisons cette réduction de bas en haut, nous pouvons la considérer comme une suite de transitions d'états ce qui nous amène à définir la sémantique opérationnelle de CPL en termes d'une relation de transition comme cela se fait habituellement dans les algèbres de processus [Plotkin, 1981].

2.2 Une relation de transition globale

A partir de cette notion de réduction totale, nous allons définir une relation de transition qui exprime la sémantique opérationnelle de CPL. Etant donné que dans une réduction totale, ce sont des systèmes de processus et non des processus isolés qui évoluent, il est naturel de commencer par définir une relation de transition globale entre systèmes de processus.

2.2.1 Définition

Considérons une réduction totale $\mathcal{R}$. Elle a nécessairement la forme :

$$\begin{array}{c} \overline{\Gamma_n \vdash 1} \\ \vdots \\ \Gamma_2 \vdash 1 \\ \hline \Gamma_1 \vdash 1 \end{array}$$

La première idée qui vient à l'esprit, est de considérer chaque inférence comme un pas de transition. La réduction entière se présenterait alors comme la suite de transitions : $\Gamma_1 \rightarrow \Gamma_2 \cdots \Gamma_n \rightarrow 1$.
Comme l'illustre bien l'exemple 7.2.1, cette traduction ne fait aucune distinction entre transitions correspondant à des actions de communication et celles qui correspondent à de simples modifications de la structure d'un processus.
Pour prendre en compte cette distinction, nous allons découper la réduction $\mathcal{R}$ en plusieurs tronçons, chacun contenant une seule action de communication, c'est-à-dire une seule inférence de type SD_L ou RC_L.
Supposons qu'il y ait n-1 inférences de communication dans $\mathcal{R}$, nous aurons ainsi n déductions $\mathcal{R}_0, \cdots, \mathcal{R}_{n-1}$.
La première $\mathcal{R}_0$ est la seule qui ne contiendra aucune action de communication car elle représente la fin de la réduction $\mathcal{R}$.
Les autres contiendrons n-1 inférences $I_1, \cdots, I_{n-1}$ de type SD_L ou RC_L. On aura ainsi la configuration suivante :

$$\mathcal{R}_0 \left\{ \begin{array}{c} \overline{\Gamma_0 \vdash 1} \\ \vdots \\ \Gamma_1 \vdash 1 \end{array} \right. \quad \cdots \quad \mathcal{R}_k \left\{ \begin{array}{c} \Gamma_k \vdash 1 \\ \vdots \\ \dfrac{\Gamma'_k \vdash 1}{\Gamma"_k \vdash 1} I_k \\ \vdots \\ \Gamma_{k+1} \vdash 1 \end{array} \right. \quad \cdots \quad \mathcal{R}_{n-1} \left\{ \begin{array}{c} \Gamma_{n-1} \vdash 1 \\ \vdots \\ \dfrac{\Gamma'_{n-1} \vdash 1}{\Gamma"_{n-1} \vdash 1} I_{n-1} \\ \vdots \\ \Gamma_n \vdash 1 \end{array} \right.$$

La réduction totale de l'exemple 7.2.1 se trouverait ainsi découpée en 4 tronçons car elle comporte 3 inférences de communication.
Pour distinguer les inférences représentant des modifications structurelles de celles représentant des actions de communication, nous allons définir deux relations :

- une *relation de réduction structurelle* auxiliaire, notée $\Rightarrow$, qui exprime les modifications possibles dans la structure d'un processus en dehors de toute action de communication ;
- la *relation de transition* proprement dite, notée $\rightarrow$, qui exprime une action de communication, éventuellement combinée avec des modifications structurelles.

On retrouve ces deux niveaux chez [Milner, 1991] qui distingue une relation de congruence structurelle de la relation de transition proprement dite. Mais notre relation de réduction structurelle est elle orientée.
La réduction finale $\mathcal{R}_0$ peut être traduite à l'aide de la relation $\Rightarrow$ de la façon suivante :

$$\begin{array}{c} \overline{\Gamma_0 \Rightarrow 1} \\ \vdots \\ \Gamma_1 \Rightarrow 1 \end{array}$$

Et une réduction partielle quelconque $\mathcal{R}_k$ contenant une action de communication sera traduite ainsi :

$$\frac{\begin{array}{c}\overline{\Gamma''_k \Rightarrow \Gamma''_k} \\ \vdots \\ \Gamma_{k+1} \Rightarrow \Gamma''_k\end{array} \quad \overline{\Gamma''_k \rightarrow \Gamma'_k} \quad \begin{array}{c}\overline{\Gamma_k \Rightarrow \Gamma_k} \\ \vdots \\ \Gamma'_k \Rightarrow \Gamma_k\end{array}}{\Gamma_{k+1} \rightarrow \Gamma_k}$$

Nous sommes maintenant en état de définir les deux relations : la relation de réduction structurelle grâce à la figure 7.1 et la relation de transition grâce à la figure 7.2

$$\frac{}{!\Gamma \Rightarrow 1}\ TERMg_S \qquad \frac{}{0, \Gamma \Rightarrow 1}\ BRK_S \qquad \frac{}{\Gamma \Rightarrow \Gamma}\ ID_S$$

$$\frac{\Gamma \Rightarrow \Gamma'}{1, \Gamma \Rightarrow \Gamma'}\ TERMl_S$$

$$\frac{P_1,\ P_2,\ \Gamma \Rightarrow \Gamma'}{P_1 \otimes P_2,\ \Gamma \Rightarrow \Gamma'}\ PAR_S$$

$$\frac{P_1,\ \Gamma \Rightarrow \Gamma'}{P_1 \& P_2,\ \Gamma \Rightarrow \Gamma'}\ ALT_S$$

$$\frac{!P,\ P,\ \Gamma \Rightarrow \Gamma'}{!P,\ \Gamma \Rightarrow \Gamma'}\ REC_S$$

$$\frac{P[y/x],\ \Gamma \Rightarrow \Gamma'}{\exists x P,\ \Gamma \Rightarrow \Gamma'}\ RES_S$$

avec y non libre dans $\exists x P,\ \Gamma$

$$\frac{P[t/x],\ \Gamma \Rightarrow \Gamma'}{\forall x P,\ \Gamma \Rightarrow \Gamma'}\ GEN_S$$

FIG. 7.1 - **Relation de réduction structurelle entre systèmes de processus de CPL**

$$\frac{}{\underline{M}.P,\ \Gamma \rightarrow P,\ M,\ \Gamma}\ SD \qquad \frac{}{M.P,\ M,\ \Gamma \rightarrow P,\ \Gamma}\ RC$$

$$\frac{\Gamma \Rightarrow \Gamma_1 \quad \Gamma_1 \rightarrow \Gamma'_1 \quad \Gamma'_1 \Rightarrow \Gamma'}{\Gamma \rightarrow \Gamma'}\ STRUCT$$

FIG. 7.2 - **Relation de transition entre systèmes de processus de CPL**

2.2.2 Exemples

Le premier est la suite de l'exemple 7.2.1 ; il permet de mieux saisir le passage d'une représentation sous forme d'une réduction totale à celle sous forme d'une suite de transitions. Les deux derniers sont

plus sophistiqués ; empruntés au Π-calculus [Milner *et al.*, 1992], ils permettent de tester la puissance d'expression de la relation de transition.

Exemple 7.2.2 Chargement et libération d'un tampon
Comme la réduction totale du système de processus $!a.\underline{b}.1,\ a,\ b.1$ *comporte 3 inférences de communication nous allons prouver la transition en 3 pas suivante :* $!a.\underline{b}.1,\ a,\ b.1 \xrightarrow{3} 1$.
Montrons chaque pas successivement.

1. Premier pas :

$$\dfrac{\dfrac{}{!a.\underline{b}.1,\ a.\underline{b}.1,\ a,\ b.1 \rightarrow !a.\underline{b}.1,\ \underline{b}.1,\ b.1}\ RC}{!a.\underline{b}.1,\ a,\ b.1 \rightarrow !a.\underline{b}.1,\ \underline{b}.1,\ b.1}\ STRUCT$$

2. Deuxième pas :

$$\dfrac{\dfrac{}{!a.\underline{b}.1,\ \underline{b}.1,\ b.1 \rightarrow !a.\underline{b}.1,\ b,\ 1,\ b.1}\ SD}{!a.\underline{b}.1,\ \underline{b}.1,\ b.1 \rightarrow !a.\underline{b}.1,\ b,\ b.1}\ STRUCT$$

3. Troisième pas :

$$\dfrac{\dfrac{}{!a.\underline{b}.1,\ b,\ b.1 \rightarrow !a.\underline{b}.1,\ 1}\ RC}{!a.\underline{b}.1,\ b,\ b.1 \rightarrow 1}\ STRUCT$$

Exemple 7.2.3 Intrusion dans un champ
Nous proposons de montrer que la transition suivante est valide :

$$\underline{m(y,x)}.P',\ R,\ \exists x((\forall z\ m(y,z).Q') \otimes S) \xrightarrow{2} P',\ R,\ Q'[x'/x][x/z],\ S[x'/x]$$

Elle correspond à cette transition du Π*-calcul :*

$$\overline{y}x.P'|R|(x)(y(z).Q'|S) \xrightarrow{\tau} P'|R|(x')(Q'\{x'/x\}\{x/z\}|S\{x'/x\}).$$

Il s'agit de traduire par cette transition une communication entre deux processus $\overline{y}x.P'$ *et* $y(z).Q'$, *le deuxième se situant dans le champ d'un canal privé* x.
Remarquons que là où il y a un pas de transition dans le Π*-calcul, il y en a deux dans CPL. Cela provient du fait que le premier est synchrone alors que le second est asynchrone. Emission et réception se confondent dans l'un alors qu'elles sont dissociées dans l'autre.*
Par ailleurs, dans l'état final de la transition du Π*-calcul, il y a encore un opérateur de restriction explicite alors que dans CPL, le fait que x' est canal privé s'exprime par des conditions sur la correction de la transition.*
La transition étant constituée de deux pas, commençons par établir le premier.

$$\dfrac{}{\underline{m(y,x)}.P',\ R,\ \exists x((\forall z\ m(y,z).Q') \otimes S) \rightarrow P',\ R,\ m(y,x),\ \exists x((\forall z\ m(y,z).Q') \otimes S)}\ SD$$

Maintenant, nous allons établir le second pas de transition

$$\dfrac{\dfrac{\dfrac{\dfrac{}{P',\ R,\ m(y,x),\ m(y,x).Q'[x'/x][x/z],\ S[x'/x] \rightarrow P',\ R,\ Q'[x'/x][x/z],\ S[x'/x]}\ RC}{P',\ R,\ m(y,x),\ \forall z\ m(y,z).Q'[x'/x],\ S[x'/x] \rightarrow P',\ R,\ Q'[x'/x][x/z],\ S[x'/x]}\ STRUCT}{P',\ R,\ m(y,x),\ (\forall z\ m(y,z).Q'[x'/x]) \otimes S[x'/x] \rightarrow P',\ R,\ Q'[x'/x][x/z],\ S[x'/x]}\ STRUCT}{P',\ R,\ m(y,x),\ \exists x((\forall z\ m(y,z).Q') \otimes S) \rightarrow P',\ R,\ Q'[x'/x][x/z],\ S[x'/x]}\ STRUCT$$

Ce deuxième pas de transition est correct à condition que x' soit différent de x et de y et qu'il ne soit pas libre dans P' et R

Exemple 7.2.4 Extrusion de champ
Nous nous proposons de démontrer la transition suivante :

$$\exists x(\underline{m(y,x)}.P' \otimes R),\ \forall z\ m(y,z).Q' \xrightarrow{2} P',\ R,\ Q'[x/z]$$

Elle correspond à la transition suivante du Π*-calcul :*

$$(x)(\overline{y}x.P'|R)|y(z).Q' \xrightarrow{\tau} (x)(P'|R|Q'\{x/z\}).$$

Il s'agit ici d'exprimer une communication entre deux processus $\overline{y}x.P'$ *et* $y(z).Q'$ *qui entraîne l'entrée du second dans le champ d'un canal privé x où se situe déjà le premier. Commençons par établir le premier pas de la transition.*

$$\dfrac{\dfrac{\dfrac{}{\underline{m(y,x)}.P',\ R,\ \forall z\ m(y,z).Q' \rightarrow P',\ m(y,x),\ R,\ \forall z\ m(y,z).Q'}\ SD}{\underline{m(y,x)}.P' \otimes R,\ \forall z\ m(y,z).Q' \rightarrow P',\ m(y,x),\ R,\ \forall z\ m(y,z).Q'}\ STRUCT}{\exists x(\underline{m(y,x)}.P' \otimes R),\ \forall z\ m(y,z).Q' \rightarrow P',\ m(y,x),\ R,\ \forall z\ m(y,z).Q'}\ STRUCT$$

La condition pour que ce pas soit correct, est que x ne soit pas libre dans Q' et qu'il soit différent de y. Et maintenant, nous allons établir le second pas de transition.

$$\dfrac{\dfrac{}{P',\ m(y,x),\ R,\ m(y,x).Q'[x/z] \rightarrow P',\ R,\ Q'[x/z]}\ RC}{P',\ m(y,x),\ R,\ \forall z\ m(y,z).Q' \rightarrow P',\ R,\ Q'[x/z]}\ STRUCT$$

2.2.3 Réduction partielle d'un système de processus et déduction logique

Le théorème suivant fait le lien entre la relation de réduction structurelle et la déduction en logique linéaire.

Théorème 7.2.2 *La transition* $\Gamma \Rightarrow \Gamma'$ *est valide si et seulement si il existe une déduction dans le système d'inférence de CPL de conclusion* $\Gamma \vdash 1$ *et d'hypothèse* $\Gamma' \vdash 1$ *sans inférences de communication.*

Preuve 7.2.2 *La condition nécessaire se démontre par induction sur la structure de l'arbre justifiant* $\Gamma \Rightarrow \Gamma'$ *et la condition suffisante par induction sur la structure de la déduction de conclusion* $\Gamma \vdash 1$. *É*□

Ce second théorème établit une correspondance entre relation de transition et déduction en logique linéaire.

Théorème 7.2.3 *La transition* $\Gamma \rightarrow \Gamma'$ *est valide si et seulement si il existe une déduction dans le système d'inférence de CPL de conclusion* $\Gamma \vdash 1$ *et d'hypothèse* $\Gamma' \vdash 1$ *qui contienne une et une seule inférence de communication.*

Preuve 7.2.3 *La condition nécessaire se démontre par induction sur la structure de l'arbre justifiant* $\Gamma \rightarrow \Gamma'$ *et la condition suffisante par induction sur la structure de la déduction de conclusion* $\Gamma \vdash 1$.□

Nous noterons $\xrightarrow{*}$ la clôture réflexive (dans un sens étendu) et transitive de $\rightarrow$ que nous pouvons définir rigoureusement ainsi :

Définition 7.2.2 $\Gamma \xrightarrow{0} \Gamma'$ *si et seulement si* $\Gamma \Rightarrow \Gamma'$.
Pour tout entier n, $\Gamma \rightarrow \Gamma_1$ *et* $\Gamma_1 \xrightarrow{n} \Gamma'$ *si et seulement si* $\Gamma \xrightarrow{n+1} \Gamma'$.
Enfin, $\Gamma \xrightarrow{*} \Gamma'$ *si et seulement si il existe un entier n tel que :* $\Gamma \xrightarrow{n} \Gamma'$.
Si $n > 0$*, nous écrirons :* $\Gamma \xrightarrow{+} \Gamma'$

Théorème 7.2.4 *La transition* $\Gamma \xrightarrow{n} \Gamma'$ *est valide si et seulement si il existe une déduction dans le système d'inférence de CPL de conclusion* $\Gamma \vdash 1$ *et d'hypothèse* $\Gamma' \vdash 1$ *avec n inférences de communication.*

Preuve 7.2.4
• `Condition nécessaire`
Nous supposons $\Gamma \xrightarrow{n} \Gamma'$ *et nous allons montrer par induction sur n que nous pouvons déduire qu'elle est vraie pour n+1. Supposons :* $\Gamma \vdash 1$ *de* $\Gamma' \vdash 1$.
Si n=0, alors nous avons :$\Gamma \Rightarrow \Gamma'$ *et d'après le théorème 7.2.2, nous en déduisons que l'on peut déduire* $\Gamma \vdash 1$ *de* $\Gamma' \vdash 1$ *sans utiliser de règle de communication.*
Maintenant, nous supposons que la propriété est vraie pour n quelconque et nous allons en déduire que $\Gamma \xrightarrow{n+1} \Gamma'$ *. Par conséquent, nous avons :* $\Gamma \rightarrow \Gamma_1$ *et* $\Gamma_1 \xrightarrow{n} \Gamma'$*. Par hypothèse d'induction, nous pouvons construire une déduction* $\mathcal{R}_1$ *de conclusion* $\Gamma_1 \vdash 1$ *à partir de l'hypothèse* $\Gamma' \vdash 1$ *et d'après le théorème 7.2.3, une autre* $\mathcal{R}_2$ *de conclusion* $\Gamma \vdash 1$ *à partir de l'hypothèse* $\Gamma_1 \vdash 1$.
Puis par composition de $\mathcal{R}_1$ *avec* $\mathcal{R}_2$*, nous obtenons une déduction de* $\Gamma \vdash 1$ *à partir de l'hypothèse* $\Gamma' \vdash 1$ *qui contient n+1 inférences de communication.*
• `Condition suffisante`
Soit $\mathcal{R}$ *une déduction de* $\Gamma \vdash 1$ *à partir de* $\Gamma' \vdash 1$*. Nous allons montrer par induction sur le nombre n d'inférences de communication de* $\mathcal{R}$ *que* $\Gamma \xrightarrow{n} \Gamma'$.
Si $n = 0$*, alors d'après le théorème 7.2.2, nous pouvons établir :* $\Gamma \Rightarrow \Gamma'$.
Ensuite, nous supposons que la propriété est vraie pour n quelconque et nous allons montrer qu'elle est vraie pour n+1. Pour cela, nous partageons la déduction $\mathcal{R}$ *en deux parties auxquelles nous pouvons appliquer l'hypothèse d'induction.*□

Corollaire 7.2.1 *La transition* $\Gamma \xrightarrow{*} \Gamma'$ *est valide si et seulement s'il existe une déduction dans le système d'inférence de CPL de conclusion* $\Gamma \vdash 1$ *et d'hypothèse* $\Gamma' \vdash 1$.

Corollaire 7.2.2 *Un système de processus* Γ *est totalement réductible si et seulement si* $\Gamma \xrightarrow{*} 1$.

2.3 Une relation de transition locale

La relation de transition mettant en jeu des systèmes de processus est comme nous l'avons vu facile à déduire de la notion de réduction totale mais elle est lourde à manipuler car il est nécessaire à chaque pas d'expliciter l'ensemble du contexte de la transition même lorsqu'il reste passif. C'est pourquoi nous allons chercher maintenant à transformer cette relation en une relation locale qui ne fasse intervenir que les processus actifs dans la transition.

2.3.1 Définition

Reprenons la réduction totale $\mathcal{R}$ du paragraphe 4.2.1. Elle comportait $n-1$ inférences de communication et nous l'avions tronçonnée en n déductions $\mathcal{R}_0$, $\mathcal{R}_1, \cdots, \mathcal{R}_n$.
Considérons une déduction $\mathcal{R}_k$ contenant une inférence de communication I_k. Elle a la forme :

$$\begin{array}{c} \Gamma_k \vdash 1 \\ \vdots \\ \dfrac{\Gamma'_k \vdash 1}{\Gamma"_k \vdash 1}\, I_k \\ \vdots \\ \Gamma_{k+1} \vdash 1 \end{array}$$

Transformons la en une preuve en utilisant le nouvel axiome ID introduit dans la sous-section 4.1.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma_k \vdash \bigotimes \Gamma_k}\,ID}{\vdots}}{\Gamma'_k \vdash \bigotimes \Gamma_k}}{\Gamma''_k \vdash \bigotimes \Gamma_k}\,I'_k \qquad \vdots \qquad \Gamma_{k+1} \vdash \bigotimes \Gamma_k}{\bigotimes \Gamma_{k+1} \vdash \bigotimes \Gamma_k}\,PAR_L$$

$\bigotimes \Gamma_k$ est le processus obtenu par composition parallèle de tous les éléments de Γ_k.
Par application de la règle $COMP$ aux séquents $\bigotimes \Gamma_1 \vdash 1$, $\bigotimes \Gamma_2 \vdash \bigotimes \Gamma_1$ $\cdots$ $\bigotimes \Gamma_n \vdash \bigotimes \Gamma_{n+1}$, nous pouvons déduire $\bigotimes \Gamma_n \vdash 1$. Nous retrouvons ainsi la réduction totale complète.
Malheureusement, chacune des $n-1$ preuves obtenues par cette transformation, n'est pas forcément correcte du fait de la condition liée à la règle RES_L. Pour les corriger, nous proposons d'insérer une inférence de type RES_R avant chaque inférence de type RES_L qui n'est pas correcte. Nous supposons aussi que nous avons découpé $\mathcal{R}$ de telle façon qu'il n'est pas nécessaire d'introduire d'inférence de type RES_R avant l'inférence de communication dans chaque preuve $\mathcal{R}_k$. Les n preuves deviennent alors :

$$\begin{array}{ccccc}
 & \dfrac{}{\Gamma_k \vdash \bigotimes \Gamma_k}\,ID & & & \\
 & \vdots & & & \\
\dfrac{}{\Gamma_0 \vdash 1}\,TERM_R & \dfrac{\Gamma'_k \vdash \bigotimes \Gamma_k}{\Gamma''_k \vdash \bigotimes \Gamma_k}\,I'_k & & \dfrac{}{\Gamma_{n-1} \vdash \bigotimes \Gamma_{n-1}}\,ID & \\
\vdots & \vdots & & \vdots & \\
\dfrac{\Gamma_1 \vdash 1}{\bigotimes \Gamma_1 \vdash 1}\,PAR_L & \bigotimes \Gamma_{k+1} \vdash \exists \vec{x_1} \bigotimes \Gamma_k & & \dfrac{\Gamma_{n-1} \vdash \bigotimes \Gamma_{n-1}}{\Gamma'_{n-1} \vdash \bigotimes \Gamma_{n-1}}\,I'_{n-1} & \\
\vdots & \vdots & & \vdots & \\
\exists \vec{x_{n-1}} \cdots \vec{x_1} \bigotimes \Gamma_1 \vdash 1 & \cdots \quad \exists \vec{x_{n-1}} \cdots \vec{x_{k+1}} \bigotimes \Gamma_{k+1} \vdash \exists \vec{x_{n-1}} \cdots \vec{x_k} \bigotimes \Gamma_k & \cdots & \bigotimes \Gamma_n \vdash \exists \vec{x_{n-1}} \bigotimes \Gamma_{n-1} &
\end{array}$$

Il est maintenant possible de traduire ces preuves à l'aide d'une relation de réduction structurelle $\Rightarrow$ et d'une relation de transition $\rightarrow$ toutes les deux locales.
La première $\mathcal{R}_0$ qui ne contient pas d'inférence de communication devient :

$$\dfrac{\dfrac{}{\bigotimes \Gamma_0 \Rightarrow 1}}{\vdots}\qquad \bigotimes \Gamma_1 \Rightarrow 1}{\exists \vec{x_{n-1}} \cdots \vec{x_1} \bigotimes \Gamma_1 \Rightarrow 1}$$

Une preuve $\mathcal{R}_k$ quelconque contenant une inférence de communication I'_k devient :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{P"_k \rightarrow P'_k} \quad \dfrac{}{\bigotimes\Gamma_k \Rightarrow \bigotimes\Gamma_k}}{\bigotimes\Gamma"_k \rightarrow \bigotimes\Gamma'_k} \quad \bigotimes\Gamma'_k \Rightarrow \bigotimes\Gamma_k}{\bigotimes\Gamma"_k \rightarrow \bigotimes\Gamma_k}}{\vdots}}{\bigotimes\Gamma_{k+1} \rightarrow \exists\vec{x_k}\bigotimes\Gamma_k}$$

$$\dfrac{\bigotimes\Gamma_{k+1} \rightarrow \exists\vec{x_k}\bigotimes\Gamma_k}{\exists\vec{x_{n-1}}\cdots\vec{x_{k+1}}\bigotimes\Gamma_{k+1} \rightarrow \exists\vec{x_{n-1}}\cdots\vec{x_{k+1}}\vec{x_k}\bigotimes\Gamma_k}$$

Dans la configuration ci-dessus, P'_k représente la partie active de I'_k et $P"_k$ la partie principale.
Maintenant, nous sommes en état de définir un système de transition d'une manière purement locale, les relations ne portant plus sur les systèmes de processus mais sur les processus eux-mêmes.
La relation de réduction structurelle est définie par la figure 7.3 et celle de transition par la figure 7.4.

$$\dfrac{}{\bigotimes!\Gamma \Rightarrow 1}\ TERMg_S \qquad \dfrac{}{0 \otimes P \Rightarrow 1}\ BRK_S \qquad \dfrac{}{P \Rightarrow P}\ ID_S$$

$$\dfrac{P \Rightarrow P'}{1 \otimes P \Rightarrow P'}\ TERMl_S$$

$$\dfrac{P_1 \otimes Q \Rightarrow P'}{(P_1 \& P_2) \otimes Q \Rightarrow P'}\ ALT_S$$

$$\dfrac{P \otimes !P \otimes Q \Rightarrow P'}{!P \otimes Q \Rightarrow P'}\ REC_S$$

$$\dfrac{P[y/x] \otimes Q \Rightarrow P'}{\exists x P \otimes Q \Rightarrow P'}\ RES_S$$

avec y non libre dans $\exists x P$, Q and P'

$$\dfrac{P[t/x] \otimes Q \Rightarrow P'}{\forall x P \otimes Q \Rightarrow P'}\ GEN_S$$

FIG. 7.3 - **Relation de réduction structurelle entre processus de CPL**

Si l'on compare relations locales et relations globales, on constate qu'il n'y a pas de modification au niveau de la relation de réduction structurelle. Simplement, au lieu de porter sur des systèmes de processus, elle porte maintenant sur des processus pour une raison de cohérence avec la relation de transition.
C'est la relation de transition qui a été fortement modifiée. Les axiomes de communication ont maintenant un caractère local et deux nouvelles règles sont apparues : PAR qui exprime l'insertion d'une transition dans un contexte et RES qui permet d'introduire une restriction sur une variable. On peut remarquer que le système obtenu ressemble beaucoup au système de transition du Π-calcul polyadique de [Milner, 1991]. Nous y reviendrons plus en détail dans le chapitre 9 quand nous aborderons l'expression du synchronisme.

$$\frac{}{\underline{M}.P \rightarrow P \otimes M}\ SD \qquad \frac{}{M.P \otimes M \rightarrow P}\ RC$$

$$\frac{P \rightarrow P'}{P \otimes Q \rightarrow P' \otimes Q}\ PAR$$

$$\frac{P[y/x] \otimes Q \rightarrow P'[y/z] \otimes Q'}{\exists x P \otimes Q \rightarrow \exists z P' \otimes Q'}\ RES$$

avec y non libre dans $\exists x P$, Q et $\exists z P'$, Q'

$$\frac{P \Rightarrow P_1 \quad P_1 \rightarrow P_1' \quad P_1' \Rightarrow P'}{P \rightarrow P'}\ STRUCT$$

FIG. 7.4 - **Relation de transition entre processus de CPL**

2.3.2 Exemples

Reprenons les exemples traités avec la relation de transition globale.

Exemple 7.2.5 Chargement et libération d'un tampon
Il s'agit de prouver la transition en 3 pas : $(!a.\underline{b}.1) \otimes a \otimes b.1 \xrightarrow{3} 1$.
Montrons chaque pas successivement.

1. Premier pas :

$$\frac{\dfrac{\dfrac{}{a.\underline{b}.1 \otimes a \rightarrow \underline{b}.1}\ RC}{(!a.\underline{b}.1) \otimes a.\underline{b}.1 \otimes a \otimes b.1 \rightarrow (!a.\underline{b}.1) \otimes \underline{b}.1 \otimes b.1}\ PAR}{(!a.\underline{b}.1) \otimes a \otimes b.1 \rightarrow (!a.\underline{b}.1) \otimes \underline{b}.1 \otimes b.1}\ STRUCT$$

2. Deuxième pas :

$$\frac{\dfrac{\dfrac{}{\underline{b}.1 \rightarrow b \otimes 1}\ SD}{\underline{b}.1 \rightarrow b}\ STRUCT}{(!a.\underline{b}.1) \otimes \underline{b}.1 \otimes b.1 \rightarrow (!a.\underline{b}.1) \otimes b \otimes b.1}\ PAR$$

3. *Troisième pas :*

$$\frac{\dfrac{\dfrac{}{b \otimes b.1 \rightarrow 1}\ RC}{(!a.\underline{b}.1) \otimes b \otimes b.1 \rightarrow (!a.\underline{b}.1) \otimes 1}\ PAR}{(!a.\underline{b}.1) \otimes b \otimes b.1 \rightarrow 1}\ STRUCT$$

Exemple 7.2.6 Intrusion dans un champ
Nous proposons de démontrer la transition :

$$\underline{m(y,x)}.P' \otimes R \otimes \exists x((\forall z\ m(y,z).Q') \otimes S) \xrightarrow{2} P' \otimes R \otimes \exists x'(Q'[x'/x][x/z] \otimes S[x'/x])$$

C'est la traduction parfaite de la transition suivante du Π*-calcul:*

$$\overline{y}x.P'|R|(x)(y(z).Q'|S) \xrightarrow{\tau} P'|R|(x')(Q'\{x'/x\}\{x/z\}|S\{x'/x\}).$$

Commençons par établir le premier pas de transition.

$$\dfrac{\dfrac{}{\underline{m(y,x)}.P' \rightarrow P' \otimes m(y,x)}\,SD}{\underline{m(y,x)}.P' \otimes R \otimes \exists x((\forall z\ m(y,z).Q') \otimes S) \rightarrow P' \otimes m(y,x) \otimes R \otimes \exists x((\forall z\ m(y,z).Q') \otimes S)}\,PAR$$

Maintenant, nous allons établir le deuxième pas de la transition.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{m(y,x) \otimes m(y,x).Q'[x'/x][x/z] \rightarrow Q'[x'/x][x/z]}\,RC}{m(y,x) \otimes \forall z\ m(y,z).Q'[x'/x] \rightarrow Q'[x'/x][x/z]}\,STRUCT}{m(y,x) \otimes \forall z\ m(y,z).Q'[x'/x] \otimes S[x'/x] \rightarrow Q'[x'/x][x/z] \otimes S[x'/x]}\,PAR}{m(y,x) \otimes \exists x((\forall z\ m(y,z).Q') \otimes S) \rightarrow \exists x'(Q'[x'/x][x/z] \otimes S[x'/x])}\,RES}{P' \otimes m(y,x) \otimes R \otimes \exists x((\forall z\ m(y,z).Q') \otimes S) \rightarrow P' \otimes R \otimes \exists x'(Q'[x'/x][x/z] \otimes S[x'/x])}\,PAR$$

Cette transition est correcte si et seulement si x' *est différent de* x *et de* y.

Exemple 7.2.7 Extrusion de champ
Nous allons démontrer la transition suivante:

$$\exists x(\underline{m(y,x)}.P' \otimes R) \otimes \forall z\ m(y,z).Q' \xrightarrow{2} \exists x(P' \otimes R \otimes Q'[x/z])$$

C'est la traduction parfaite de la transition suivante du Π*-calcul:*

$$(x)(\overline{y}x.P'|R)|y(z).Q' \xrightarrow{\tau} (x)(P'|R|Q'\{x/z\}).$$

Nous commençons par établir le premier pas de la transition.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\underline{m(y,x)}.P' \rightarrow P' \otimes m(y,x)}\,SD}{\underline{m(y,x)}.P' \otimes R \rightarrow P' \otimes m(y,x) \otimes R}\,PAR}{\exists x(\underline{m(y,x)}.P' \otimes R) \rightarrow \exists x(P' \otimes m(y,x) \otimes R)}\,RES}{\exists x(\underline{m(y,x)}.P' \otimes R) \otimes \forall z\ m(y,z).Q' \rightarrow \exists x(P' \otimes m(y,x) \otimes R) \otimes \forall z\ m(y,z).Q'}\,PAR$$

Et maintenant, nous allons établir le deuxième pas de la transition.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{m(y,x) \otimes m(y,x).Q'[x/z] \rightarrow Q'[x/z]}\,RC}{m(y,x) \otimes \forall z\ m(y,z).Q' \rightarrow Q'[x/z]}\,STRUCT}{P' \otimes m(y,x) \otimes R \otimes \forall z\ m(y,z).Q' \rightarrow P' \otimes R \otimes Q'[x/z]}\,PAR}{\exists x(P' \otimes m(y,x) \otimes R) \otimes \forall z\ m(y,z).Q' \rightarrow \exists x(P' \otimes R \otimes Q'[x/z])}\,RES$$

Ce dernier pas de transition est correct si et seulement si x *n'est pas libre dans* Q'.

Les deux derniers exemples montrent que la relation de transition locale de CPL est plus proche de celle du Π-calcul que la relation globale. Ce qui les sépare, est le caractère asynchrone de la communication dans CPL alors qu'elle est synchrone dans le Π-calcul et secondairement la relation de réduction structurelle de CPL ne recouvre pas totalement la congruence structurelle du Π-calcul. Mais nous étudierons ceci plus en détail dans le chapitre 9.

2.3.3 Réduction partielle d'un processus et déduction logique

Commençons par établir le lien entre la relation de réduction structurelle et celle de déduction.

Théorème 7.2.5 *Soit P et P' deux processus de CPL.*
La transition $P \Rightarrow P'$ est valide si et seulement si le séquent $P \vdash P'$ est prouvable en utilisant uniquement des axiomes et des règles de réduction structurelle, c'est-à-dire des règles gauches différentes de SD_L et RC_L.

Preuve 7.2.5 *La condition nécessaire s'établit par induction sur la structure de l'arbre justifiant la relation $\Gamma \Rightarrow \Gamma'$ et la condition suffisante par induction sur la structure de la preuve de $P \vdash P'$.*□

Etudions maintenant le rapport entre la relation de transition et celle de déduction.
D'après la manière dont nous avons amené la relation de transition locale, en décomposant une réduction totale il est à prévoir qu'il est une règle droite qui sera indispensable à la traduction de la notion de réduction partielle, la règle RES_R. Mais son utilisation va se faire dans des conditions bien particulières. Pour traduire ceci, nous allons introduire une nouvelle règle dérivable dans le système d'inférence de CPL : RES_{LR} qui permet de fusionner l'application successive et dans certaines conditions de RES_R et de RES_L.

$$\frac{P[y/x],\ \Gamma \vdash P'[y/z] \otimes Q'}{\exists x P,\ \Gamma \vdash (\exists z P') \otimes Q'}\ RES_{LR} \quad \text{avec y non libre dans la conclusion.}$$

L'utilisation de la règle RES_{LR} va permettre d'énoncer plus simplement le théorème établissant la relation entre réduction partielle et déduction.

Théorème 7.2.6 *La transition $P \rightarrow P'$ est valide si et seulement si le séquent $P \vdash P'$ est prouvable dans une preuve qui commence par un axiome de type ID, qui contient une seule inférence de communication, les autres inférences étant des applications de règles de réduction structurelle ou de la règle RES_{LR} (après l'inférence de communication pour cette dernière).*

Preuve 7.2.6 *La condition nécessaire se démontre par induction sur la structure de l'arbre justifiant la transition $P \rightarrow P'$ et la condition suffisante par induction sur la structure de la preuve de $P \vdash P'$.* □

Comme dans le cas global, nous pouvons définir la clôture réflexive et transitive de la relation de transition et établir des théorèmes analogues sur son rapport avec la relation de déduction.

Théorème 7.2.7 *La transition $P \xrightarrow{n} P'$ est valide si et seulement si le séquent $P \vdash P'$ est prouvable dans une preuve qui commence par un axiome de type ID, contient n inférences de communication, les autres inférences étant des applications de règles de réduction structurelle ou de la règle RES_{LR} (après la première inférence de communication pour cette dernière).*

Preuve 7.2.7 • `Condition nécessaire`
Nous supposons $P \xrightarrow{n} P'$ et nous allons prouver par induction sur n que le séquent $P \vdash P'$ est prouvable dans les conditions décrites par le théorème.
Si n=0 ou n=1, alors il n'y a qu'à appliquer les théorèmes 7.2.5 ou 7.2.6.
Ensuite, nous supposons que la propriété est vraie pour $n \geq 1$ et nous allons montrer qu'elle est vraie pour n+1.
Supposons que nous ayons $P \xrightarrow{n+1} P'$ .Nous avons alors : $P \rightarrow P_1$ et $P_1 \xrightarrow{n} P'$. Par hypothèse d'induction, nous pouvons construire une preuve $\mathcal{P}_1$ qui contient n inférences de communication, les

autres inférences étant des applications de règles de réduction structurelle ou de la règle RES_{LR} (après la première inférence de communication pour cette dernière) et qui se présente ainsi :

$$\frac{}{\Gamma'' \vdash P''}\ ID$$

$$\vdots$$

$$\frac{\Gamma'_1 \vdash \exists \vec{x''} P''}{P'_1 \vdash \exists \vec{x''} P''}\ PAR_L$$

$$\vdots$$

$$\underbrace{\exists \vec{x_1} P'_1}_{P_1} \vdash \underbrace{\exists \vec{x'} \vec{x''} P''}_{P'}$$

D'après le théorème 7.2.6, nous pouvons construire une preuve $\mathcal{P}_2$ qui contient une inférence de communication, les autres inférences étant des applications de règles de réduction structurelle ou de la règle RES_{LR} et qui se présente ainsi :

$$\frac{}{\Gamma'_1 \vdash P'_1}\ ID$$

$$\vdots$$

$$P \vdash \underbrace{\exists \vec{x_1} P'_1}_{P_1}$$

A l'aide des preuves $\mathcal{P}_1$ et $\mathcal{P}_2$, nous allons en construire une nouvelle qui a la forme :

$$\frac{}{\Gamma'' \vdash P''}\ ID$$

$$\vdots$$

$$\Gamma'_1 \vdash \exists \vec{x''} P''$$

$$\vdots$$

$$P \vdash \underbrace{\exists \vec{x'} \vec{x''} P''}_{P'}$$

Il est facile de vérifier que les inférences de type RES_L et RES_{LR} sont correctes. La preuve contient n+1 inférences de communication et les autres inférences sont toutes des applications de règles de réduction structurelle ou de RES_{LR} (pour cette dernière, après la première inférence de communication). La propriété est donc démontrée pour n+1.

- **Condition suffisante**

Soit $\mathcal{P}$ une preuve de $P \vdash P'$ qui satisfait les conditions décrites dans le théorème.
Nous allons montrer par induction sur le nombre n d'inférences de communication de $\mathcal{P}$ que $P \xrightarrow{n} P'$.
Si $n = 0$, alors d'après le théorème 7.2.5, nous pouvons établir $P \Rightarrow P'$.
Si $n = 1$, alors d'après le théorème 7.2.6, nous pouvons établir $P \rightarrow P'$.
Maintenant nous supposons que la propriété est vraie pour $n \geq 1$ et nous allons montrer qu'elle est vraie pour n+1. Considérons une preuve $\mathcal{P}$ qui a n+1 inférences de communication. Nous pouvons la découper juste avant la dernière inférence de communication en une preuve $\mathcal{P}_1$ et une déduction $\mathcal{D}_2$ qui

se présentent ainsi :

$$\begin{array}{cc} \dfrac{}{\Gamma" \vdash P"}\,ID & \\ \vdots & \\ \Gamma' \vdash \exists \vec{x'} P" & \end{array} \qquad \begin{array}{c} \Gamma' \vdash \exists \vec{x'} P" \\ \vdots \\ P \vdash \underbrace{\exists \vec{x} \vec{x'} P"}_{P'} \end{array}$$

Par hypothèse d'induction, nous pouvons déduire de $\mathcal{P}_1$ la transition $\bigotimes \Gamma' \xrightarrow{n} \exists \vec{x'} P"$. Puis par application de la règle RES, nous pouvons en dériver la transition $\exists \vec{x} \bigotimes \Gamma' \xrightarrow{n} \exists \vec{x} \exists \vec{x'} P"$ c'est-à-dire $\exists \vec{x} \bigotimes \Gamma' \xrightarrow{n} P'$. A partir de la déduction $\mathcal{D}_2$, nous pouvons construire une preuve $\mathcal{P}_2$ qui a la forme suivante :

$$\begin{array}{c} \dfrac{}{\Gamma' \vdash \bigotimes \Gamma'}\,ID \\ \vdots \\ P \vdash \exists \vec{x} \bigotimes \Gamma' \end{array}$$

Il est facile de vérifier que les inférences de type RES_L et RES_{LR} y sont correctes. La preuve vérifie les conditions d'application du théorème 7.2.6 donc la transition $P \rightarrow \exists \vec{x} \bigotimes \Gamma'$ est dérivable.
On peut donc déduire : $P \xrightarrow{n+1} P'$ □

Corollaire 7.2.3 *La transition $P \xrightarrow{*} P'$ est valide si et seulement si le séquent $P \vdash P'$ est prouvable dans une preuve qui commence par un axiome de type ID et dont les autres inférences sont des applications de règles de réduction ou de la règle RES_{LR} (après la première inférence de communication pour cette dernière).*

Corollaire 7.2.4 *Un processus P est totalement réductible si et seulement si $P \xrightarrow{*} 1$.*

2.4 Conclusion

Le calcul de CPL peut maintenant être présenté sous deux formes : à l'aide de la relation de transition et sous forme logique. Le rapport entre les deux est précisé par le corollaire 7.2.3. Il n'est pas toutefois définitivement figé. En effet, nous pouvons nous demander s'il n'est pas possible d'étendre la sémantique opérationnelle de façon à ce qu'elle recouvre davantage la notion de déduction logique. N'est-il pas possible par exemple d'envisager que tout séquent $P \vdash P'$ démontrable dans une preuve qui n'utilise pas les règles SD_R et RC_R puisse représenter une réduction d'un processus P en un processus P'? La question demande à être approfondie.
Comparons la sémantique opérationnelle de CPL avec celle du système le plus proche, ACL. Dans sa présentation initiale [Kobayashi and Yonezawa, 1994a], ACL est muni d'une sémantique opérationnelle se confondant avec la déduction logique. Comme l'interprétation est disjonctive, une preuve d'un séquent de la forme $\vdash P$ est vue comme une réduction totale du processus P et chaque inférence de cette preuve comme une transition élémentaire de la conclusion vers le prémisse. La relation de transition ne distingue donc pas les actions de communication des modifications structurelles de processus et elle est globale. En outre, le fait d'utiliser des séquents sans partie gauche induit chez Kobayashi et Yonazewa une forme particulière de terminaison des calculs qui s'exprime par l'axiome :

$$\frac{}{\vdash \top, \Delta}\,\top$$

Cela signifie que le processus $\top$ termine en détruisant tous les autres qui se réduisent en parallèle et qui sont représentés ici par Δ. Il n'est donc pas possible d'exprimer dans ACL le fait qu'un processus obtenu

par composition parallèle de deux autres se réduit totalement lorsque chacune de ses composantes fait de même.
Considérons par exemple le processus $a.0 \mid \overline{a}.0$ de CCS [Milner, 1989]. D'après [Kobayashi and Yonezawa, 1994a], il peut être traduit dans ACL par le processus $(a^{\perp} \otimes \perp) \wp (a \wp \perp)$ qui n'est pas totalement réductible car le séquent $\vdash (a^{\perp} \otimes \perp) \wp (a \otimes \perp)$ n'est pas prouvable.
Par contre, dans CPL, le même processus est traduit par $(a \multimap 1) \otimes (a \otimes 1)$ qui est totalement réductible car le séquent $(a \multimap 1) \otimes (a \otimes 1) \vdash 1$ est prouvable. Dans CPL, on a donc une forme "douce" de terminaison d'un calcul qui n'existe pas dans ACL. Mais on a aussi la forme "brutale" de ACL. Ainsi, le séquent $0, \Gamma \vdash 1$ est prouvable donc le processus 0 peut détruire tout un ensemble de processus s'exécutant en parallèle.
En étendant ACL à l'ordre supérieur, [Kobayashi and Yonezawa, 1994b] ont modifié quelque peu la relation de transition attachée à leur calcul en la rendant locale.
La relation de transition définissant la sémantique opérationnelle de Hcc [Lincoln and Saraswat, 1992] est très liée à l'approche choisie, celle de la programmation parallèle avec contraintes. Elle est globale mais distingue bien les interrogations de l'ensemble de contraintes qui peuvent être considérées comme des réceptions de message, des modifications structurelles des processus. Elle permet aussi d'établir un lien entre réduction partielle d'un processus et déduction logique même si celui-ci est complexe.

Chapitre 8

La sémantique dénotationnelle de CPL

Introduction

La sémantique dénotationnelle vise à établir une équivalence entre processus fondée sur une propriété que l'on cherche à exprimer. Ce que nous cherchons à exprimer ici, c'est *la capacité pour un processus donné d'interagir avec le monde extérieur.*
C'est aussi ce que cherche à capter la notion de *bissimulation* [Park, 1980] qui prévaut dans la communauté des spécialistes du parallélisme. Pourquoi ne nous sommes donc pas appuyés sur celle-ci pour définir la sémantique dénotationnelle de CPL?
Tout d'abord, elle est très liée au déroulement même des calculs : elle se vérifie pas à pas au fur et à mesure que les processus se réduisent. Cela signifie d'une part qu'elle n'est pas facile d'utilisation et d'autre part qu'elle peut être porteuse de plus de sens que l'on ne voudrait.
Ensuite, elle ne fournit pas un objet aisément manipulable qui représente la dénotation d'un processus.
Enfin et surtout, il nous a paru judicieux de tenir compte et d'exploiter au maximum le terrain sur lequel nous nous situons, qui est le terrain logique.
Pour toutes ces raisons, nous nous sommes orientés dans une autre direction tracée par [Miller, 1992] avec la notion de co-agent et reprise ensuite par [Kobayashi and Yonezawa, 1994a; Volpe, 1994]. Nous l'avons adaptée à CPL sous forme de la notion de *co-processus.* Ainsi, un co-processus d'un processus P est une formule Q telle que le séquent $P, Q \vdash 1$ est prouvable.
Il y a au centre de cette sémantique une notion de dualité liée à la *sémantique des phases* telle qu'elle a pu être définie par [Girard, 1987] pour la logique linéaire.
Pour caractériser l'interaction d'un processus avec le monde extérieur, il n'est pas suffisant de considérer des co-processus isolément mais il faut prendre un ensemble de co-processus qui constitueront ce que nous appellerons une *interface.*
A partir de là, nous pourrons définir une relation de *satisfaction d'une interface par un processus* qui fera le lien entre les deux mondes duaux : celui des processus et celui des interfaces.
Se plaçant dans le monde des interfaces, nous allons pouvoir élaborer un calcul sur celles-ci qui repose sur la déduction logique et qui s'articule autour des notions de *réduction* et de *relativisation* d'une interface.
Dans ce calcul, nous verrons qu'il est un type d'interfaces qui joue un rôle particulier : les *interfaces linéaires* qui sont formées de processus qui sont des suites d'envoi et de réception de messages.
Les comparaisons entre interfaces vont pouvoir être transposées dans le monde des processus. Nous obtiendrons ainsi des *relations de pré-ordre et d'équivalence entre processus.* Cela va permettre d'étendre la notion de réduction partielle d'un processus en une notion de *réalisation d'un processus par un autre.*
Dans le chapitre 7, nous avons vu qu'un processus P se réduit en un processus P' si et seulement si le séquent $P \vdash P'$ est prouvable avec certaines restrictions. Dans ce chapitre, nous allons voir qu'un processus P réalise un processus P', c'est-à-dire qu'il satisfait son interface, si et seulement si le séquent $P \vdash P'$ est prouvable sans restriction.

1 Notion d'interface d'un processus

1.1 Co-processus d'un processus et interface satisfaite par un processus

Nous partons de l'idée que la sémantique dénotationnelle doit nous permettre de caractériser l'interaction des processus avec le monde extérieur par opposition au mécanisme interne de réduction de chacun d'eux. Pour cela, nous allons utiliser des testeurs que nous qualifierons de co-processus.

Définition 8.1.1 *Un* co-processus *de CPL est une formule de logique linéaire intuitionniste construite à partir des formules atomiques de* $\mathcal{M}[\mathcal{D}, \mathcal{V}]$.

Remarque 8.1.1 *Un co-processus de CPL n'est pas nécessairement un processus de CPL. Par exemple,* $(M \multimap 1) \multimap 1$ *est un co-processus mais pas un processus de CPL.*

Un co-processus n'a de sens que par rapport à un processus vis-à-vis duquel il joue le rôle de testeur. D'où la définition :

Définition 8.1.2 *Un* co-processus *Q de CPL est un* co-processus d'un processus *P de CPL si et seulement si le séquent* $P, Q \vdash 1$ *est prouvable dans LL.*

Cette définition signifie qu'un co-processus, placé en parallèle avec un processus va lui permettre de se réduire totalement. Nous retrouvons ici la notion de co-agent établie par [Miller, 1992] et reprise par [Kobayashi and Yonezawa, 1993] mais avec des différences. Pour Miller, un co-agent Q d'un agent P est tel que $\vdash P, Q$ est prouvable. Cette définition est liée au caractère disjonctif de l'interprétation choisie, ce qui induit une conception particulière de la terminaison des calculs déjà mentionnée dans la conclusion du chapitre précédent.
Illustrons notre définition à l'aide d'un exemple.

Exemple 8.1.1 *Considérons de nouveau le processus* P_{fact} *traité au début du chapitre précédent. Considérons aussi un processus* P_{arith} *doté des opérations arithmétiques nécessaires à* P_{fact} *et où les entiers sont implantés en notation décimale. Citons quelques co-processus de* P_{fact} *:*

- *comme* P_{fact} *est totalement réductible, tout autre processus totalement réductible ;*
- $P_{fact} \multimap 1$ *qui n'est pas un processus de CPL ;*
- $\underline{(fact\ in : 5)}.(fact\ out : 120).1 \otimes P_{arith}$
- $\underline{(fact\ in : 5)}.\forall\, x(fact\ out : x).1 \otimes P_{arith}$
- $\underline{(fact\ in : 5)}.1 \otimes \forall\, x(fact\ out : x).1 \otimes P_{arith}$

Chacun des co-processus décrit un aspect de l'interaction entre P_{fact} *et le monde extérieur. Ainsi, le troisième caractérise le fait que* P_{fact} *est capable de calculer correctement la valeur de !5 alors que le quatrième constitue une caractérisation plus faible : il dit simplement que* P_{fact} *est capable de calculer !5 mais il ne dit rien quant au résultat trouvé.*

Comme le montre l'exemple qui vient d'être traité, il n'est pas suffisant de considérer un seul co-processus pour caractériser complètement la capacité d'interaction d'un processus avec l'extérieur. Il faut en utiliser plusieurs d'où la notion de interface.

Définition 8.1.3 *Une* interface de CPL *est un ensemble de co-processus de CPL.*
Un processus P satisfait une interface *I si et seulement si tous les éléments de I sont des co-processus de P.*

Pour illustrer cette définition, reprenons l'exemple 8.1.1.

Exemple 8.1.2 *Soit* I_1 *l'interface formée de l'unique co-processus* $\underline{(fact\ in : 5)}.(fact\ out : 120).1 \otimes P_{arith}$. *Le processus* P_{fact} *satisfait* I_1. *Il est clair que tout processus qui satisfait* I_1, *calcule correctement la factorielle du nombre 5 mais on ne peut rien dire de son comportement si on lui demande de calculer la*

factorielle d'un autre entier.
Elargissons l'interface et considérons maintenant la interface I_2 constituée des trois processus :

$$\underline{(fact\ in:5)}.(fact\ out:120).1 \otimes P_{arith}$$

$$\underline{(fact\ in:5)}.\forall\ x(fact\ out:x).1 \otimes P_{arith}$$

$$\underline{(fact\ in:5)}.1 \otimes \forall\ x(fact\ out:x).1 \otimes P_{arith}$$

P_{fact} satisfait I_2 .Il est facile de montrer que les processus qui satisfont I_2 sont les mêmes que ceux qui satisfont I_1. L'interface bien qu'elle ait été étendue, n'a donc pas été précisée. Nous analyserons ce phénomène quand nous en viendrons à la notion de réduction d'une interface.
On peut trouver une interface qui caractérise tous les processus qui calculent correctement la factorielle de n'importe quel entier. Elle serait formée de tous les processus de la forme : $\underline{(fact\ in:n)}.(fact\ out:m).1$ où n est un entier quelconque et m est un entier égal à !n. Cette interface étant infinie, il est donc nécessaire de la remplacer par une interface finie équivalente. Encore faut-il définir précisément ce qu'on entend par interface équivalente, ce qu'on fera à travers la notion de réduction d'une interface.

1.2 Réduction d'une interface

Comme nous venons de le voir dans le dernier exemple, la définition extensive d'une interface est difficile à utiliser pratiquement. Il est nécessaire de pouvoir la remplacer par une interface équivalente plus petite. Commençons par définir précisément ce qu'on entend par là.

Définition 8.1.4 Une interface I_2 se déduit d'une interface I_1 *si et seulement si, pour tout élément $Q_1 \in I_1$, il existe un élément $Q_2 \in I_2$ tel que le séquent $Q_1 \vdash Q_2$ est prouvable dans LL.*
Nous écrirons alors :$I_1 \vdash I_2$.
Deux interfaces I_1 et I_2 sont équivalentes *si et seulement si elles se déduisent mutuellement l'une de l'autre.*
Nous écrirons alors :$I_1 \dashv\vdash I_2$.

La première relation est un pré-ordre tandis que la seconde est une relation d'équivalence. A partir de celles-ci, nous pouvons définir la notion de réduction d'une interface.

Définition 8.1.5 *Une interface I_2 est une* réduction d'une interface *I_1 si et seulement si $I_2 \subseteq I_1$ et $I_2 \dashv\vdash I_1$.*
Nous écrirons alors : $I_2 \sqsubseteq I_1$ ou $I_1 \sqsupseteq I_2$.

Le théorème suivant est immédiat.

Théorème 8.1.1 *Une interface I_2 est une réduction d'une interface I_1 si et seulement si $I_2 \subseteq I_1$ et $I_1 \vdash I_2$.*

Pour illustrer ces notions reprenons l'exemple 8.1.2.

Exemple 8.1.3 *Il est facile d'établir que les séquents suivants sont prouvables :*

$$\underline{(fact\ in:5)}.\forall\ x(fact\ out:x).1 \otimes P_{arith} \vdash \underline{(fact\ in:5)}.(fact\ out:120).1 \otimes P_{arith}$$

$$\underline{(fact\ in:5)}.1 \otimes \forall\ x(fact\ out:x).1 \otimes P_{arith} \vdash \underline{(fact\ in:5)}.(fact\ out:120).1 \otimes P_{arith}$$

On en déduit : $I_1 \sqsubseteq I_2$.

Pour chaque interface, il serait intéressant de pouvoir trouver une réduction minimum. Malheureusement, ce n'est pas toujours possible. Considérons par exemple l'interface :

$$S = \{\forall x\ m(x),\ \forall x\ m(t(x)), \cdots, \forall x\ m(t^n x), \cdots\}$$

où m est un prédicat de message et t un opérateur de données.
S a une infinité de réductions de la forme :

$$I_k = \{\forall x\ m(t^k(x)),\ \forall x\ m(t^{k+1}(x)), \cdots, \forall x\ m(t^{k+n} x), \cdots\}$$

mais pas de plus petite réduction.
Une interface n'ayant de sens que par rapport aux processus qui la satisfont, il est essentiel d'étudier la

compatibilité des relations entre interfaces qui viennent d'être définies, avec celle de satisfaction d'une interface par un processus.
Le théorème suivant va ainsi venir justifier la définition de la relation de déduction entre interfaces.

Théorème 8.1.2 *Soit deux interfaces I_1 et I_2 telles que $I_1 \vdash I_2$.*
Alors tout processus qui satisfait I_2, satisfait I_1.

Preuve 8.1.1 *Soit un processus P qui satisfait I_2. Soit Q_1 un co-processus quelconque appartenant à I_1. Comme $I_1 \vdash I_2$, il existe un co-processus Q_2 tel que $Q_1 \vdash Q_2$. Et puisque P satisfait I_2, par définition, le séquent $P, Q_2 \vdash 1$ est prouvable. Et comme $Q_1 \vdash Q_2$, par une coupure sur les deux séquents, on peut inférer $P, Q_1 \vdash 1$.Donc Q_1 est un co-processus de P et comme c'est un élément quelconque de I_1, cela signifie que P satisfait I_1.* □

Corollaire 8.1.1 *Un processus satisfait une interface I si et seulement s'il satisfait une interface équivalente à I.*

Corollaire 8.1.2 *Un processus satisfait une interface I si et seulement s'il satisfait une réduction de I.*

1.3 Interface absolue et interfaces relatives d'un processus

Parmi toutes les interfaces satisfaites par un processus donné, il en est une qui est privilégiée : c'est celle qui inclut toutes les autres et que nous appellerons l' *interface absolue du processus.*

Définition 8.1.6 *L'*interface absolue d'un processus *P est l'ensemble de tous ses co-processus. Elle est notée $\mathcal{I}(P)$.*

Contrairement à une interface quelconque, il est possible de réduire au maximum l'interface absolue d'un processus.

Théorème 8.1.3 *l'interface absolue d'un processus P a une plus petite réduction qui est $\{P \multimap 1\}$.*

Preuve 8.1.2 *Comme le séquent $P, P \multimap 1 \vdash 1$ est prouvable, $(P \multimap 1) \in \mathcal{I}(P)$.*
Soit Q un co-processus quelconque de P. Par définition, le séquent $P, Q \vdash 1$ est prouvable.Donc par application de $\multimap_R$, on peut inférer $Q \vdash P \multimap 1$. On a ainsi montré que $\{P \multimap 1\}$ est une réduction de $\mathcal{I}(P)$.□

Les processus de CPL permettent de modéliser des programmes, des systèmes qui évoluent en général dans des environnements très typés. Par exemple, un programme Prolog s'exécute lorsqu'un but qui est une suite de formules atomiques lui est fourni. Un programme impératif attend qu'on lui fournisse un certain nombre de données en entrée dans un format précis et en retourne d'autres en sortie.
Dans CPL, il possible de modéliser ceci en figeant la forme des co-processus utilisés pour tester les processus. Pour cela, nous allons utiliser le concept d'environnement.

Définition 8.1.7 *Un* environnement *est un ensemble de co-processus.*

Formellement, un environnement se définit comme une interface mais l'usage n'est pas le même. Pour mieux le comprendre, voici quelques exemples d'environnements.

Exemple 8.1.4 – *L'environnement $\mathcal{P}$ constitué par les processus de CPL.*

- *L'environnement $\mathcal{N}$ des* co-processus normaux *constitués par les processus de CPL dans lesquels il n'y a pas de généralisation sur les canaux quand ils sont utilisés en tant que tels. Expliquons cette définition. Elle sous-entend déjà que les messages de CPL sont dans le format $(c : x_1 \cdots x_n)$ où c constitue le canal par lequel est transmis le message correspondant et $x_1 \cdots x_n$ représente le contenu du message. Et nous nous interdisons de placer au sein d'un processus ce message dans le champ d'un opérateur $\forall c$. L'environnement $\mathcal{N}$ joue un rôle très important car il donne au masquage de canaux un caractère inviolable comme nous le verrons dans de futurs exemples.*

- *L'environnement $\mathcal{PP}$ constitué par les processus de CPL qui ne contiennent aucun envoi de messages. Nous appelerons ces co-processus des* co-processus passifs *et nous les retrouvons dans la modélisation des programmes Prolog.*

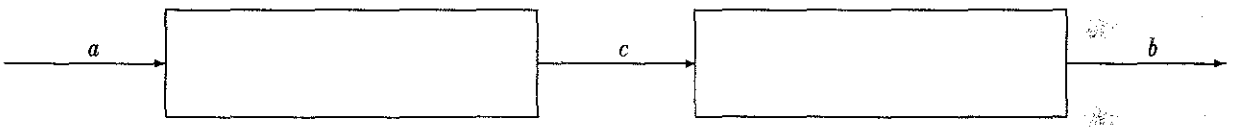
- *L'environnement constitué par les processus de la forme $\underline{(x\ in:t)}.(x\ out:t').1$ que nous qualifierons de* co-processus fonctionnels. *Ils joueront un rôle dans la modélisation des programmes impératifs. En effet t représente les données fournies en entrée au processus associé et t' les données récupérées en sortie du même processus une fois qu'il a été réduit. Nous pouvons étendre cet environnement en utilisant la quantification.*

- *Nous pouvons enfin définir un environnement à partir d'une* signature. *Par exemple, nous pouvons associer au processus P_{fact} de l'exemple 1.2.2, l'environnement formé par les co-processus dont les constituants atomiques sont des messages de la forme (fact in :t), (fact out :t t'), (zero in :), (zero out : t), (unite in :), (unite out : t), (egal in :t), (egal out : t t'), (mult in :t), (mult out : t t'), (decr in : t), (decr out : t t') où t et t' sont des termes quelconques de $\mathcal{D}[\mathcal{V}]$.*

Définition 8.1.8 *L'*interface d'un processus P relative à un environnement $\mathcal{E}$ *est l'ensemble des co-processus de P qui appartiennent à $\mathcal{E}$.*
On la notera $\mathcal{I}_{\mathcal{E}}(P)$.

Illustrons cette définition par un exemple qui montre bien l'importance de l'environnement $\mathcal{N}$.

Exemple 8.1.5 *Considérons deux tampons à capacité infinie tels que le port de sortie du premier soit connecté au port d'entrée du second par le biais d'un canal interne c.*



L'ensemble peut être représenté par le processus suivant : $P \equiv \exists c\ (!(a.\underline{c}.1)\otimes!(c.\underline{b}.1))$ Considérons le co-processus $Q \equiv \forall x\ \underline{a}.x.\underline{x}.b$. C'est un co-processus de P car du fait de la généralisation, il peut recevoir une unité d'information de n'importe canal x, même privé, pour la retourner ensuite sur le même canal. Il peut donc dans notre exemple shunter le canal c bien que celui-ci soit masqué.
Un tel co-processus est exclu de l'environnement $\mathcal{N}$. Examinons donc quelle est l'interface de P relativement à $\mathcal{N}$. Il est facile de montrer que celle-ci se réduit à l'ensemble des processus de la forme $\underline{a}.b.\cdots\underline{a}.b.1$. Dans la prochaine section, nous trouverons les outils pour mener à bien une telle démonstration. Ce qui nous importe pour l'instant, c'est le résultat. Or, il prouve que notre ensemble est équivalent dans l'environnement $\mathcal{N}$ à un seul tampon à capacité infinie. Par contre, ceci n'est plus vrai dans l'environnement $\mathcal{P}$.

Habituellement, nous travaillerons dans l'environnement $\mathcal{P}$ ou dans une restriction de celui-ci. Quand nous parlerons d'interface relative d'un processus sans rien ajouter, cela sous-entendra interface relative à $\mathcal{P}$.

2 Processus et interfaces linéaires

Pour simplifier les calculs, il serait intéressant de restreindre l'environnement $\mathcal{P}$ au maximum mais en conservant la propriété suivante : *toutes les interfaces relatives de processus doivent avoir une réduction incluse dans cet environnement.* L'environnement constitué par les processus linéaires répond à ce critère. C'est ce que nous allons montrer maintenant.

2.1 Notion de processus linéaire et opérations associées

Définition 8.2.1 *Un* processus linéaire *de CPL est un processus L de CPL défini inductivement à l'aide de la grammaire suivante :*

$$L ::= 1 \mid \underline{M}.L \mid M.L \mid \forall x\ L \mid \exists x L$$

Une interface linéaire *est un ensemble de processus linéaires.*
*L'*environnement linéaire $\mathcal{L}$ *est l'ensemble de tous les processus linéaires.*

Voici quelques exemples de processus linéaires.
Exemple 8.2.1

$$L_1 = \underline{(fact\ in: 5)}.(fact\ out: 120).1$$
$$L_2 = \underline{(fact\ in: 5)}.\forall\ x(fact\ out: x).1$$
$$L_3 = \underline{(fact\ in: 5)}.\exists\ x(fact\ out: x).1$$
$$L_4 = \underline{(fact\ in: 5)}.\exists\ x(fact\ out: x).\underline{(fact\ in: 3)}.\exists\ x(fact\ out: x).1$$

Nous allons maintenant définir deux opérations sur le processus linéaires : *la dualité* et *l'entrelacement.*

Définition 8.2.2 *Le dual* $\overline{L}$ *d'un processus linéaire* L *est défini inductivement ainsi :*

$$\overline{1} = 1;\quad \overline{\underline{M}.L} = M.\overline{L};\quad \overline{M.L} = \underline{M}.\overline{L};\quad \overline{\forall x L} = \exists x \overline{L};\quad \overline{\exists x L} = \forall x \overline{L}.$$

La dualité est évidemment involutive. Les théorèmes suivants donnent d'autres propriétés importantes de cette opération

Théorème 8.2.1 *Pour tout processus linéaire* L*, le séquent* $L,\ \overline{L} \vdash 1$ *est prouvable dans CPL.*

Preuve 8.2.1 *Elle s'effectue par induction sur la structure de* L.□

Théorème 8.2.2 *Soit un processus linéaire* L *et un co-processus* Q*. Le séquent* $L,\ Q \vdash 1$ *est prouvable si et seulement si le séquent* $Q \vdash \overline{L}$ *est prouvable.*

Preuve 8.2.2 *Elle découle de cette propriété : quand nous construisons une preuve de* $L,\ Q \vdash 1$*, nous pouvons construire en parallèle une preuve de* $Q \vdash \overline{L}$ *en remplaçant chaque inférence gauche produisant un sous-processus de* L *par une inférence droite duale produisant un sous-processus de* $\overline{L}$*. La propriété est aussi vraie en sens inverse.*□

Corollaire 8.2.1 $\{\overline{L}\}$ *est une réduction de l'interface absolue du processus linéaire* L.

Venons-en maintenant à la deuxième opération importante sur les processus linéaires : *l'entrelacement.*

Définition 8.2.3 *Soi* L_1 *et* L_2 *deux processus linéaires dont toutes les variables liées sont distinctes entre elles et distinctes des variables libres présentes dans les deux processus (si ce n'est pas le cas, on effectue un renommage).*
L'ensemble $L_1 * L_2$ *de tous les processus linéaires obtenus par entrelacement de* L_1 *et de* L_2 *est défini inductivement ainsi :*
- *si* $L_1 = 1$*, alors* $L_1 * L_2 = \{L_2\}$;
- *si* $L_2 = 1$*, alors* $L_1 * L_2 = \{L_1\}$;
- *si* $L_1 = \alpha_1 L_1'$ *et* $L_2 = \alpha_2 L_2'$ *avec* α_1 *et* α_2 *éléments de* $\{M.,\ \underline{M}.,\ \forall x,\ \exists x\}$*, alors*

$L_1 * L_2 = \{\alpha_1 L \mid L \in L_1' * L_2\} \cup \{\alpha_2 L \mid L \in L_1 * L_2'\}$.

Exemple 8.2.2 *Considérons les processus* L_2 *et* L_3 *de l'exemple 8.2.1. Avant de définir* $L_2 * L_3$*, nous renommons la variable* x *de* L_3 *en* y*. Les deux processus se présentent alors ainsi :*

$$L_2 = \underline{(fact\ in: 5)}.\forall\ x(fact\ out: x).1$$
$$L_3 = \underline{(fact\ in: 5)}.\exists\ y(fact\ out: y).1$$

Alors, l'ensemble $L_2 * L_3$ *est constitué de 14 processus. En voici quelques-uns :*

$$\underline{(fact\ in: 5)}.\forall\ x(fact\ out: x).\underline{(fact\ in: 5)}.\exists\ y(fact\ out: y).1$$
$$\underline{(fact\ in: 5)}.\forall\ x\underline{(fact\ in: 5)}.(fact\ out: x).\exists\ y(fact\ out: y).1$$
$$\underline{(fact\ in: 5)}.\underline{(fact\ in: 5)}.\forall\ x(fact\ out: x).\exists\ y(fact\ out: y).1$$
$$\underline{(fact\ in: 5)}.\underline{(fact\ in: 5)}.\exists\ y(fact\ out: y).\forall\ x(fact\ out: x).1$$
$$\underline{(fact\ in: 5)}.\exists\ y\underline{(fact\ in: 5)}.(fact\ out: y).\forall\ x(fact\ out: x).1$$

Le théorème suivant va jouer un rôle clé dans la réduction de toute interface relative à une interface linéaire.

Théorème 8.2.3 *Soit L_1 et L_2 deux processus linéaires de CPL.*
$\overline{L_1} * \overline{L_2}$ est une réduction de $\mathcal{I}_P(L_1 \otimes L_2)$.

Preuve 8.2.3 *Nous allons commencer par prouver que tout élément L de $\overline{L_1} * \overline{L_2}$ est un co-processus de $L_1 \otimes L_2$. Nous allons procéder par induction sur la taille l de L.*
Si $l = 1$, alors $L = L_1 = L_2 = 1$. Comme $1, 1 \vdash 1$ est prouvable, alors la propriété est vraie.
Nous supposons que la propriété est vraie pour l quelconque et nous allons montrer qu'elle est aussi vraie pour $l + 1$.
Soit L un processus linéaire de taille $l+1$, obtenu par entrelacement de deux processus linéaires $\overline{L_1}$ et $\overline{L_2}$. Selon la tête de L, nous pouvons distinguer les cas suivants :

1. $\mathsf{L} = \underline{\mathsf{M}}.\mathsf{L}'$
   Alors, l'un des deux processus $\overline{L_1}$ ou $\overline{L_2}$ est de la forme : $\underline{M}.\overline{L_1'}$. Supposons par exemple qu'il s'agit de $\overline{L_1}$.
   Par hypothèse d'induction, nous avons : $L_1' \otimes L_2,\ L' \vdash 1$. Comme dans les preuves, les inférences de type PAR_L peuvent être descendues au maximum, on en déduit : $L_1',\ L_2,\ L' \vdash 1$.
   Ensuite, nous pouvons inférer successivement : $M,\ M.L_1',\ L_2,\ L' \vdash 1$, $M.L_1',\ L_2,\ \underline{M}.L' \vdash 1$ et $L_1 \otimes L_2,\ \underline{M}.L' \vdash 1$ en utilisant les règles RC_L , SD_L et PAR_L.

2. $\mathsf{L} = \mathsf{M}.\mathsf{L}'$
   Alors nous avons par exemple : $\overline{L_1} = M.\overline{L_1'}$. Par hypothèse d'induction, le séquent $L_1' \otimes L_2,\ L' \vdash 1$ est prouvable. On peut en déduire : $L_1',\ L_2,\ L' \vdash 1$.
   Ensuite, nous pouvons inférer successivement $M,\ L_1',\ L_2,\ M.L' \vdash 1$, $\underline{M}.L_1',\ L_2,\ M.L' \vdash 1$ et $L_1 \otimes L_2,\ M.L' \vdash 1$ à l'aide des règles RC_L , SD_L et PAR_L.

3. $\mathsf{L} = \forall x \mathsf{L}'$
   Alors, nous avons par exemple : $\overline{L_1} = \forall x \overline{L_1'}$. Par hypothèse d'induction, le séquent $L_1' \otimes L_2,\ L' \vdash 1$ est prouvable. Nous pouvons en déduire : $L_1',\ L_2,\ L' \vdash 1$.
   Nous pouvons ensuite inférer : $L_1',\ L_2,\ \forall x L' \vdash 1$ en utilisant la règle GEN_L. Puis, comme x n'est pas libre dans L_2, nous pouvons utiliser la règle RES_L pour inférer $\exists x L_1',\ L_2,\ \forall x L' \vdash 1$ et nous terminons par l'application de la règle PAR_L.

4. $\mathsf{L} = \exists x \mathsf{L}'$
   Alors, nous avons par exemple : $\overline{L_1} = \exists x \overline{L_1'}$. Par hypothèse d'induction, le séquent $L_1' \otimes L_2,\ L' \vdash 1$ est démontrable et donc $L_1',\ L_2,\ L' \vdash 1$ aussi.
   D'où nous pouvons utiliser la règle GEN_L pour inférer : $\forall x L_1',\ L_2,\ L' \vdash 1$. Et comme x n'est pas libre dans L_2, nous obtenons $\forall x L_1',\ L_2,\ \exists x L' \vdash 1$ par application de la règle RES_L et nous terminons avec PAR_L .

Maintenant, nous allons montrer que pour tout co-processus Q de $L_1 \otimes L_2$, il existe un élément L de $\overline{L_1} * \overline{L_2}$ tel que : $Q \vdash L$. Nous procéderons par induction sur la structure de L_1. Selon la forme de L_1, nous pouvons distinguer les cas suivants :

1. $\mathsf{L}_1 = 1$
   Par hypothèse, nous avons : $1 \otimes L_2,\ Q \vdash 1$ d'où nous pouvons déduire $L_2,\ Q \vdash 1$. Par application du théorème 8.2.2, nous obtenons alors : $Q \vdash \overline{L_2}$ ce qu'il fallait démontrer.

2. $\mathsf{L}_1 = \underline{\mathsf{M}}.\mathsf{L}_1'$
   Par hypothèse, nous avons : $\underline{M}.L_1' \otimes L_2,\ Q \vdash 1$. Comme nous pouvons descendre au maximum les inférences de type PAR_L dans les preuves de CPL, nous en déduisons : $\underline{M}.L_1',\ L_2,\ Q \vdash 1$. Nous pouvons faire de même pour les inférences de type SD_L et nous obtenons alors : $M,\ L_1',\ L_2,\ Q \vdash 1$. Par application de la règle PAR_L, nous pouvons inférer $M,\ L_1' \otimes L_2,\ Q \vdash 1$. Par hypothèse d'induction, nous pouvons trouver un élément L' de $\overline{L_1'} * \overline{L_2}$ tel que : $M,\ Q \vdash L$. D'où nous inférons grâce à la règle RC_R : $Q \vdash M.L$. Comme $M.L$ appartient à $\overline{L_1} * \overline{L_2}$, la démonstration est terminée.

3. $\mathsf{L}_1 = \mathsf{M}.\mathsf{L}'_1$
Par hypothèse, nous avons : $M.L'_1 \otimes L_2,\ Q \vdash 1$ *d'où nous déduisons* $M.L'_1,\ L_2,\ Q \vdash 1$.
Considérons une preuve de ce séquent ; elle a la forme suivante :

$$\begin{array}{c} \vdots \\ \dfrac{L'_1,\ L'_2,\ Q' \vdash 1}{M.L'_1,\ M,\ L'_2,\ Q' \vdash 1}\ RC_L \\ \vdots \\ M.L'_1,\ L_2,\ Q \vdash 1 \end{array}$$

Dans cette preuve, L'_2 *représente un sous-processus de* L_2.
Par hypothèse d'induction, nous pouvons trouver un élément L' *de* $\overline{L'_1} * \overline{L'_2}$ *tel que :* $Q' \vdash L'$.
Nous pouvons alors modifier la preuve ci-dessus en remplaçant chaque inférence gauche qui contribue à la construction de L_2 *par une inférence droite correspondante. Nous obtenons la preuve suivante :*

$$\begin{array}{c} \vdots \\ \dfrac{Q' \vdash L'}{M,\ Q' \vdash \underline{M}.L'}\ SD_R \\ \vdots \\ Q \vdash L \end{array}$$

Il est facile de montrer que le processus linéaire L *appartient à* $\overline{L_1} * \overline{L_2}$.

4. $\mathsf{L}_1 = \forall \mathsf{x} \mathsf{L}'_1$
Par hypothèse, nous avons : $(\forall x L'_1) \otimes L_2,\ Q \vdash 1$ *d'où nous pouvons déduire :* $\forall x L'_1,\ L_2,\ Q \vdash 1$ *car nous pouvons descendre au maximum les inférences de type* PAR_L *dans les preuves de CPL.*
Nous procédons ensuite comme dans le cas précédent en considérant une preuve du séquent, en utilisant l'hypothèse d'induction et en remplaçant ensuite certaines inférences gauches par des inférences droites correspondantes.

5. $\mathsf{L}_1 = \exists \mathsf{x} \mathsf{L}'_1$
Par hypothèse, nous avons : $(\exists x L'_1) \otimes L_2,\ Q \vdash 1$ *d'où nous pouvons déduire :* $\exists x L'_1,\ L_2,\ Q \vdash 1$.
Comme nous pouvons descendre au maximum les inférences de type REC_L *dans les preuves, nous en déduisons :* $L'_1[y/x],\ L_2,\ Q \vdash 1$ *avec* y *non libre dans* $\exists x L'_1,\ L_2,\ Q \vdash 1$.
Par hypothèse d'induction, nous pouvons trouver un élément L' *de* $\overline{L'_1} * \overline{L_2}$ *tel que :* $Q \vdash L'[y/x]$.
D'où nous pouvons inférer à l'aide de la règle GEN_R *le séquent* $Q \vdash \forall x L'$. *Comme* $\forall x L'$ *appartient à* $\overline{L_1} * \overline{L_2}$, *nous avons montré ce que nous voulions.*

□

Les processus linéaires vont être les constituants des interfaces linéaires. La comparaison logique de ces interfaces entre elles nécessite d'étudier la démontrabilité de séquents de la forme $L_1 \vdash L_2$ où L_1 et L_2 sont des processus linéaires.
Le théorème suivant nous fournit les principaux séquents de cette forme qui sont prouvables (nous avons laissé de côté ceux qui étaient fournis par des équivalences logiques).

Théorème 8.2.4 *Les séquents de CPL suivants sont prouvables :*

1	$\underline{M_1}.M_2.L \vdash M_2.\underline{M_1}.L$	
2	$\underline{M}.M.L \vdash L$	$L \vdash M.\underline{M}.L$
3	$\exists x \forall y L \vdash \forall y \exists x L$	
4	$\forall x L \vdash L[t/x]$	$L[t/x] \vdash \exists x L$
5	$\forall x L \vdash \forall x L[t/x]$	$\exists x L[t/x] \vdash \exists x L$
6	$\underline{M}.\forall x L \vdash \forall x \underline{M}.L$	$\exists x M.L \vdash M.\exists x L$
7	$\forall x M.L \vdash M.\forall x L$	$\underline{M}.\exists x L \vdash \exists x \underline{M}.L$
8	$M.\forall x L \vdash \forall x M.L$	$\exists x \underline{M}.L \vdash \underline{M}.\exists x L$

Pour les séquents des lignes 6, 7, et 8, x ne doit pas être libre dans M.

En utilisant ce théorème, nous pouvons parfois réduire considérablement une interface linéaire. Ainsi, l'ensemble $L_2 * L_3$ de l'exemple 8.2.2 qui est constitué de 14 éléments, peut être considéré comme une interface linéaire. Et à l'aide du théorème ci-dessus, on peut le réduire à un ensemble formé des deux processus suivants :

$$\underline{(\text{fact in}:5)}.\forall\, x(\text{fact out}:\text{x}).\underline{(\text{fact in}:5)}.\exists\, y(\text{fact out}:\text{y}).1$$

$$\underline{(\text{fact in}:5)}.\exists\, y(\text{fact out}:\text{y}).\underline{(\text{fact in}:5)}.\forall\, x(\text{fact out}:\text{x}).1$$

Nous allons montrer en quoi les interfaces linéaires sont intéressantes : *il est en effet possible de trouver pour n'importe quelle interface relative d'un processus une réduction de celle-ci qui est linéaire.*

2.2 Réduction d'une interface relative en une interface linéaire

Rappelons que lorsque nous parlons d'interfaces relatives, nous voulons dire par là des interfaces dont tous les éléments sont des processus.
Nous allons prouver deux lemmes nécessaires à la démonstration du théorème central qui montre que toute interface relative se réduit à une interface linéaire. Le premier va permettre de montrer que l'interface relative du processus 0 qui est $\mathcal{P}$ tout entier se réduit à une interface linéaire.

Lemme 8.2.1 *Pour tout processus P de CPL, il existe un processus linéaire L tel que le séquent $P \vdash L$ est prouvable dans CPL.*

Preuve 8.2.4 *Nous allons procéder par induction sur la structure de P. Selon la forme de P, nous avons les cas suivants :*

1. P = M
   Pour L, il suffit de choisir $\underline{M}.1$.

2. P = 1 ou P = 0
   Pour L, nous choisirons 1.

3. P = $\underline{M}$.P'
   Par hypothèse d'induction, il existe un processus linéaire L' tel que le séquent $P' \vdash L'$ est prouvable. Par application des règles SD_R et SD_L, nous pouvons ensuite inférer les séquents $M, P' \vdash \underline{M}.L'$ et $\underline{M}.P' \vdash \underline{M}.L'$ ce qu'il fallait prouver.

4. P = M.P'
Par hypothèse d'induction, il existe un processus linéaire L' tel que le séquent $P' \vdash L'$ est prouvable. Par application des règles RC_L et RC_R, nous pouvons ensuite inférer les séquents M, $M.P' \vdash L'$ et $M.P' \vdash M.L'$ ce qu'il fallait prouver.

5. $P = P_1 \otimes P_2$
Par hypothèse d'induction, il existe deux processus linéaires L_1 et L_2 tels que : $P_1 \vdash L_1$ et $P_2 \vdash L_2$. Par application des règles PAR_R et PAR_L, nous pouvons inférer successivement P_1, $P_2 \vdash L_1 \otimes L_2$ et $P_1 \otimes P_2 \vdash L_1 \otimes L_2$.
Ensuite, nous considérons un élément quelconque L de $L_1 * L_2$ et en appliquant le théorème 8.2.3, nous pouvons établir que : $L_1 \otimes L_2$, $\overline{L} \vdash 1$ car $\overline{L} \in \overline{L_1} * \overline{L_2}$.
En utilisant le théorème 8.2.2, nous en déduisons $L_1 \otimes L_2 \vdash L$. Enfin à l'aide de la règle $COMP$, nous obtenons : $P_1 \otimes P_2 \vdash L$.

6. $P = P_1 \& P_2$
Par hypothèse d'induction, il existe un processus linéaire L tel que $P_1 \vdash L$. Ensuite, en utilisant la règle ALT_L, nous inférons : $P_1 \& P_2 \vdash L$.

7. $P = \forall x P'$
Par hypothèse d'induction, il existe un processus linéaire L tel que $P_1 \vdash L$. Ensuite, en utilisant la règle GEN_L, nous inférons : $\forall x P' \vdash L$.

8. $P = \exists x P'$
Par hypothèse d'induction, il existe un processus linéaire L tel que $P_1 \vdash L$. Ensuite, en utilisant les règles REI_R et REI_L nous pouvons inférer successivement : $P' \vdash \exists x L$ et $\exists x P' \vdash \exists x L$.

9. $P = !P'$
En utilisant la règle $TERM_R$, nous obtenons : $!P \vdash 1$. □

Le deuxième lemme va aider à construire une réduction linéaire de l'interface relative d'un processus obtenu par composition parallèle à partir d'une réduction de l'interface relative de ses deux composants.

Lemme 8.2.2 *Si P_1 et P_2 sont deux processus de CPL et si L_1 et L_2 sont deux co-processus linéaires respectifs de P_1 et P_2, alors tout élément de $L_1 * L_2$ est un co-processus de $P_1 \otimes P_2$.*

Preuve 8.2.5 *Par hypothèse, nous avons : P_1, $L_1 \vdash 1$ et P_2, $L_2 \vdash 1$. Alors P_1 et P_2 sont des co-processus respectifs de L_1 et L_2 et le théorème 8.2.2 nous permet de déduire : $P_1 \vdash \overline{L_1}$ et $P_2 \vdash \overline{L_2}$.*
En appliquant ensuite les règles PAR_R et PAR_L, nous pouvons inférer successivement : P_1, $P_2 \vdash \overline{L_1} \otimes \overline{L_2}$ et $P_1 \otimes P_2 \vdash \overline{L_1} \otimes \overline{L_2}$.
Puis nous considérons un élément quelconque L de $L_1 * L_2$. D'après le théorème 8.2.3, nous avons : $\overline{L_1} \otimes \overline{L_2}$, $L \vdash 1$. Enfin à l'aide de la règle $COMP$, nous pouvons inférer : $P_1 \otimes P_2$, $L \vdash 1$. □

Les deux lemmes que nous venons d'établir, vont nous permettre d'aborder maintenant le théorème central de cette partie.

Théorème 8.2.5 *Pour tout processus P de CPL, il existe une interface linéaire qui est une réduction de $\mathcal{I}_{\mathcal{P}}(P)$ et qui a les mêmes variables libres que P.*
En outre, si 0 n'est pas un sous-processus de P, alors les éléments de cette interface ne comprennent que des messages présents dans P avec éventuellement un renommage de variables liées.

Preuve 8.2.6 *Voir en annexe E* □

La démonstration de ce théorème est constructive et donc fournit un algorithme qui permet de calculer une réduction linéaire de n'importe quelle interface relative d'un processus. Il peut être optimisé de façon notamment à réduire au maximum la taille des interfaces construites au fur et à mesure. Montrons le concrètement en calculant une réduction linéaire de l'interface relative d'un tampon de capacité infinie.

Exemple 8.2.3 *Nous nous proposons avec l'algorithme utilisé dans la démonstration précédente, de calculer une réduction linéaire de l'interface relative du processus $!a.\underline{b}.1$ qui représente un tampon à capacité*

infinie.
Tout d'abord : $\mathcal{I}_{\mathcal{P}}(a.\underline{b}.1) \sqsupseteq \{\underline{a}.b.1\}$.
Il faut ensuite déterminer une réduction linéaire de $\mathcal{I}_{\mathcal{P}}(a.\underline{b}.1 \otimes \cdots \otimes a.\underline{b}.1)$ *Nous nous contenterons de le faire pour une conjonction de deux processus de la forme* $a.\underline{b}.1$. *On l'étendra par récurrence.*
$\mathcal{I}_{\mathcal{P}}(a.\underline{b}.1 \otimes a.\underline{b}.1) \sqsupseteq (\underline{a}.b.1) * (\underline{a}.b.1)$ *c'est-à-dire :* $\mathcal{I}_{\mathcal{P}}(a.\underline{b}.1 \otimes a.\underline{b}.1) \sqsupseteq \{\underline{a}.b.\underline{a}.b.1,\ \underline{a}.\underline{a}.b.b.1\}$. *Or cette interface formée de deux processus, peut encore se réduire au singleton* $\{\underline{a}.b.\underline{a}.b.1\}$.
On en déduit que $\mathcal{I}_{\mathcal{P}}(!a.\underline{b}.1)$ *se réduit à l'ensemble des processus de la forme* $\underline{a}.b.\cdots\underline{a}.b.1$.
On peut remarquer que le tampon à une place satisfait une telle interface donc dans un cadre asynchrone, un tampon à une place peut réaliser relativement un tampon infini (nous donnerons un sens précis à cette expression quand nous aborderons la comparaison des processus).

3 Comparaison de processus

Nous avons défini l'interface d'un processus comme un objet logique caractérisant ses possibilités d'interaction avec le monde extérieur. Nous avons montré qu'il était possible de relativiser l'interface d'un processus en fonction de l'environnement. Nous avons vu qu'il était possible de réduire une interface, c'est-à-dire la remplacer par une interface plus petite en taille mais équivalente en signification, ceci afin de faciliter les calculs.
Or, quels calculs est-il intéressant de faire? On peut chercher à savoir si un processus donné satisfait une interface donnée mais on peut aussi comparer deux processus du point de vue de leurs interfaces. La comparaison peut se faire dans l'absolu mais elle peut se faire aussi relativement à un environnement. Dans l'un ou l'autre des cas, la notion de réduction d'interface vient nous faciliter la tâche.

3.1 Comparaison absolue de processus

Naturellement, ce sont les interfaces absolues des processus qui vont permettre d'effectuer cette comparaison. C'est la relation d'inclusion qui va permettre de comparer les interfaces et par là même les processus.

Définition 8.3.1 *Un processus* P_1 réalise *un processus* P_2 *si et seulement si* $\mathcal{I}(P_1) \supseteq \mathcal{I}(P_2)$.
On dit aussi que P_1 *est une* réalisation *de* P_2 *et on écrit :* $P_1 \geq P_2$.

La relation $\geq$ est une relation de pré-ordre du fait que la relation d'inclusion sur les ensembles est une relation d'ordre.
Le théorème suivant montre que la relation $\geq$ se confond avec la relation de déduction logique entre processus. Nous avions vu que la relation $P \vdash P'$ pouvait être interprétée comme "le processus P se réduit au processus P'" dans certaines conditions. Maintenant nous avons une autre interprétation de la même relation mais cette fois sans restriction aucune.

Théorème 8.3.1 *Un processus* P_1 réalise *un processus* P_2 *si et seulement si le séquent* $P_1 \vdash P_2$ *est prouvable dans CPL.*

Preuve 8.3.1 *Commençons par prouver la condition suffisante qui est la plus simple. Nous supposons que le séquent* $P_1 \vdash P_2$ *est prouvable et nous nous proposons de montrer que* $\mathcal{I}(P_1) \supseteq \mathcal{I}(P_2)$.
Soit Q *un co-processus quelconque de* P_2. *Par définition :* $P_2,\ Q \vdash 1$. *Par une coupure sur* P_2, *nous pouvons en déduire :* $P_1,\ Q \vdash 1$. *Donc* Q *est un co-processus de* P_1. *On a donc démontré que* $\mathcal{I}(P_1) \supseteq \mathcal{I}(P_2)$.
Passons maintenant à la condition nécessaire. Nous supposons que $\mathcal{I}(P_1) \supseteq \mathcal{I}(P_2)$ *et nous voulons montrer que* $P_1 \vdash P_2$.
Nous savons que $P_2 \multimap 1$ *est un co-processus de* P_2 *donc d'après les hypothèses, c'est aussi un co-processus de* P_1. *Le séquent* $P_1,\ P_2 \multimap 1 \vdash 1$ *est donc prouvable.*
Considérons une preuve $\mathcal{P}$ *de celui-ci où l'inférence* I *produisant* $P_2 \multimap 1$ *a été descendue au maximum.*

Selon le type d'obstacle rencontré par I dans ce mouvement, nous sommes amenés à distinguer les cas suivants :

1. I est la dernière inférence de $\mathcal{P}$
   La preuve $\mathcal{P}$ se termine selon deux configurations possibles :

   (a)

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ P_1 \vdash 1\end{array}}{P_1,\ 1 \vdash 1}\ 1_L \qquad \begin{array}{c}\vdots\\ \vdash P_2\end{array}}{P_1,\ P_2 \multimap 1 \vdash 1}\ \multimap_L$$

   De cette configuration, nous pouvons déduire $1 \vdash P_2$ et comme $P_1 \vdash 1$, nous obtenons par coupure $P_1 \vdash P_2$.

   (b)

$$\dfrac{\dfrac{\dfrac{}{\vdash 1}\ 1_R}{1 \vdash 1}\ 1_L \qquad \begin{array}{c}\vdots\\ P_1 \vdash P_2\end{array}}{P_1,\ P_2 \multimap 1 \vdash 1}\ \multimap_L$$

   Ici, nous avons directement le résultat cherché : $P_1 \vdash P_2$

2. I est bloquée par une inférence de type $\otimes_L$
   Donc $\mathcal{P}$ a la forme suivante :

$$\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\\ P_1',\ \Gamma' \vdash 1\end{array}}{P_1',\ 1,\ \Gamma' \vdash 1}\ 1_L \qquad \begin{array}{c}\vdots\\ P_1'',\ \Gamma'' \vdash P_2\end{array}}{P_1',\ P_1'',\ P_2 \multimap 1,\ \Gamma',\ \Gamma'' \vdash 1}\ \multimap_L}{P_1' \otimes P_1'',\ P_2 \multimap 1,\ \Gamma',\ \Gamma'' \vdash 1}\ \otimes_L}{\begin{array}{c}\vdots\\ P_1,\ P_2 \multimap 1 \vdash 1\end{array}}$$

   Nous pouvons remplacer la preuve par celle-ci :

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\\ P_1',\ \Gamma' \vdash 1\end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ P_1'',\ \Gamma'' \vdash P_2\end{array}}{P_1'',\ 1,\ \Gamma'' \vdash P_2}\ 1_L}{P_1',\ P_1'',\ \Gamma',\ \Gamma'' \vdash P_2}\ cut}{P_1' \otimes P_1'',\ \Gamma',\ \Gamma'' \vdash P_2}\ \otimes_L}{\begin{array}{c}\vdots\\ P_1,\ \vdash P_2\end{array}}$$

   Le séquent $P_1,\ \vdash P_2$ est donc prouvable.

3. *I est bloquée par une inférence de type $c!_L$*
   Nous procédons de la même façon que dans le cas précédent.

□

Après avoir interprété partiellement la déduction logique d'un processus à partir d'un autre comme une réduction du second au premier, nous pouvons grâce au théorème précédent, l'interpréter maintenant sans restriction comme une relation de réalisation du deuxième processus par le premier. Illustrons cette notion à travers quelques exemples qui montrent ses différentes facettes.

Exemple 8.3.1

1. Un exemple simple.
   Nous avons prouvé (section 2 du chapitre 7) le séquent $!\underline{b}.a.1 \vdash !a.\underline{b}.1$. *On peut donc dire qu'un tampon infini initialement plein réalise un tampon infini initialement vide.*

2. La séquentialisation d'un programme parallèle calculant la factorielle d'un entier.
   Reprenons l'exemple 1.2.2. Le processus P_{fact} *contient divers sous-processus qui sont composés en parallèle. Nous allons écrire une nouvelle définition* P'_{fact} *qui calcule toujours la factorielle d'un entier mais où toutes les actions sont séquentialisées. Cela signifie que l'opérateur* $\otimes$ *est banni. Nous obtenons donc la définition suivante :*

$$
\begin{array}{l}
P'_{fact} = \; !\ \forall\, n\ (fact\ in : n). \\
\qquad\qquad \underline{(zero\ in :)}. \\
\qquad\qquad \underline{(unite\ in :)}. \\
\qquad\qquad \exists\, n_c\ f_c\ \underline{(n_c : n)}. \\
\qquad\qquad\qquad \forall\, u\ \ (unite\ out : u). \\
\qquad\qquad\qquad\qquad \underline{(f_c : u)}. \\
\qquad\qquad\qquad\qquad \forall\, z\ (zero\ out : z). \\
\qquad\qquad\qquad\qquad\quad !\ \forall\, f\ m\ (f_c : f). \\
\qquad\qquad\qquad\qquad\qquad\qquad (n_c : m). \\
\qquad\qquad\qquad\qquad\qquad\qquad \underline{(egal\ in : (m,z))}. \\
\qquad\qquad\qquad\qquad\qquad\qquad \&\ (egal\ out : (m,z)\ true). \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \underline{(fact\ out : n\ f)}. \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad 1 \\
\qquad\qquad\qquad\qquad\qquad\qquad \&\ (egal\ out : (m,z)\ false). \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \underline{(mult\ in : (f,m))}. \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \forall\, f'\ (mult\ out : (f,m)\ f'). \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \underline{(f_c : f')}. \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \underline{(decr\ in : m)}. \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \forall\, m'\ \ (decr\ out : m\ m'). \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \underline{(n_c : m')}. \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad 1
\end{array}
$$

   On peut montrer que le séquent $P_{fact} \vdash P'_{fact}$ *est prouvable. Donc d'après le théorème précédent, toute interface qui est satisfaite par* P'_{fact}, *est satisfaite par* P_{fact}. *En particulier si l'on montre que* P'_{fact} *calcule correctement la factorielle d'un entier, nous pouvons en déduire que* P_{fact} *aussi.*
   La relation de réalisation en sens inverse est fausse.
   On peut aller plus loin dans la séquentialisation en se débarrassant non seulement de la composition parallèle mais aussi de la composition alternative.
   Le lemme 8.2.1 le permet de manière générale puisqu'il montre que de n'importe quel processus de CPL, on peut toujours déduire un processus linéaire.
   D'une certaine façon, on retrouve ici la relation de raffinement *entre programmes parallèles [Back, 1979,1980, 1988,Chandy and Misra, 1988].*

3. Le dépliage en programmation logique
   Soit le programme Prolog p_1 *suivant :*

$$a :\!-\ b,\ c.$$

b :- d, e.
b.

Chaque clause peut être représenté dans CPL par un processus récurrent tel que la tête de clause se traduit par une émission et le corps par une réception. La première clause par exemple, sera représentée par le processus $b.c.\underline{a}.1$. *Et le programme* p_1 *tout entier, sera représenté par le processus*

$$P_1 \equiv !(b.c.\underline{a}.1) \otimes !(d.e.\underline{b}.1) \otimes !\underline{b}.1.$$

Par dépliage du prédicat b dans la première clause du programme p_1, *nous obtenons le programme* p_2 *suivant :*

a :- d, e, c.
a :- c.
b :- d, e.
b.

Ce programme est alors représenté par le processus $P_2 \equiv !(d.e.c.\underline{a}.1) \otimes !(c.\underline{a}.1) \otimes !(d.e.\underline{b}.1) \otimes !\underline{b}.1$.
Le séquent $P_1 \vdash P_2$ *est facile à prouver donc le dépliage en programmation logique est un exemple de réalisation d'un processus par un autre.*

4. L'héritage en programmation orientée objets
Reprenons l'exemple de la représentation des polygones et des rectangles en programmation orientée objets.
La démontrabilité du séquent

$$P_{translation} \otimes P_{rotation} \otimes P_{perimetre} \otimes P_{centre} \vdash P_{translation} \otimes P_{rotation} \otimes P_{perimetre}$$

montre que le processus représentant les routines de la classe rectangle *réalise celui représentant les routines de la classe* polygone. *Ainsi se trouve traduite la relation d'*héritage *entre classes dans la programmation orientée objets.*

Comme à partir de toute relation de pré-ordre, à partir de la relation $\geq$, il est possible de définir une relation d'équivalence entre processus. Deux processus P_1 et P_2 seront équivalents si et seulement si $P_1 \geq P_2$ et $P_2 \geq P_1$. Or d'après le théorème que nous venons de démontrer, cela équivaut à dire que P_1 et P_2 sont logiquement équivalents. On obtient donc une équivalence entre processus qui est très forte. Elle peut se traduire sous deux formes à l'aide du théorème suivant :

Théorème 8.3.2 *Deux processus* P_1 *et* P_2 *de CPL ont même interface absolue si et seulement si ils sont logiquement équivalents.*

Preuve 8.3.2 *Elle découle immédiatement du théorème précédent.*

Les relations que nous venons d'étudier sont très fortes. De la même façon que nous avons cherché à relativiser la notion d'interface par rapport à un environnement donné, nous allons relativiser les relations de pré-ordre et d'équivalence entre processus qui en découlent.

3.2 Comparaison relative de processus

Elle va s'effectuer en comparant les interfaces relatives à un environnement donné des processus d'où la définition suivante.

Définition 8.3.2 *Un processus* P_1 réalise un processus P_2 relativement à un environnement $\mathcal{E}$ *si et seulement si* $\mathcal{I}_{\mathcal{E}}(P_1) \supseteq \mathcal{I}_{\mathcal{E}}(P_2)$.
Nous dirons aussi que P_1 *est* une réalisation de P_2 relativement à $\mathcal{E}$ *et nous écrirons :* $P_1 \geq^{\mathcal{E}} P_2$.

Il est évident que la relation $\geq^{\mathcal{E}}$ est une relation de pré-ordre et que, si P_1 réalise P_2 dans l'absolu, il le réalise relativement à n'importe quel environnement.
Le théorème suivant peut aider à déterminer pratiquement si deux processus vérifient la relation.

Théorème 8.3.3 $P_1 \geq^{\mathcal{E}} P_2$ *si et seulement si* P_1 *satisfait une réduction de* $\mathcal{I}_{\mathcal{E}}(P_2)$.

Preuve 8.3.3 *La condition nécessaire est évidente et la condition suffisante découle du corollaire 8.1.2.*□

Si nous choisissons comme environnement, celui $\mathcal{P}$ des processus de CPL, nous pouvons appliquer les résultats relatifs aux interfaces linéaires d'où le théorème :

Théorème 8.3.4 *$P_1 \geq^{\mathcal{P}} P_2$ si et seulement si P_1 satisfait une réduction linéaire de $\mathcal{I}_{\mathcal{P}}(P_2)$.*

Preuve 8.3.4 *La condition nécessaire découle du théorème 8.2.5 et la condition suffisante du corollaire 8.1.2.*□

Du fait que la relation $\geq^{\mathcal{E}}$ est un pré-ordre, nous pouvons en déduire une relation d'équivalence.

Définition 8.3.3 *Deux processus P_1 et P_2 sont dits* équivalents relativement à l'environnement *$\mathcal{E}$ si et seulement si $P_1 \geq^{\mathcal{E}} P_2$ et $P_2 \geq^{\mathcal{E}} P_1$, c'est-à-dire s'ils ont la même interface re lative à $\mathcal{E}$.*
Nous écrirons alors : $P_1 \overset{\mathcal{E}}{\sim} P_2$.

Pour qu'une relation d'équivalence soit utilisable dans un calcul, il faut qu'elle soit compatible avec les opérations de ce calcul, ce que montre le théorème suivant :

Théorème 8.3.5 *La relation $\overset{\mathcal{E}}{\sim}$ est une relation de congruence sur les processus de CPL.*

Preuve 8.3.5 *La propriété de relation d'équivalence provient immédiatement du fait qu'elle se traduit par l'égalité entre ensembles qui est elle-même une relation d'équivalence.*
Montrons maintenant que $\overset{\mathcal{E}}{\sim}$ est compatible avec les opérations sur les processus.
Soit deux processus P_1 et P_2 tels que $P_1 \overset{\mathcal{E}}{\sim} P_2$. Nous effectuons la même opération sur les deux processus et nous obtenons deux nouveaux processus P_1' et P_2'. Il s'agit de montrer que $P_1' \overset{\mathcal{E}}{\sim} P_2'$.
Soit Q un co-processus de P_1' qui appartient à $\mathcal{E}$. Il s'agit de montrer que c'est un co-processus de P_2'.
Par définition : $P_1',\ Q \vdash 1$. Soit une preuve $\mathcal{P}$ de ce séquent. Selon le type d'opération effectuée sur P_1 et P_2, nous devons distinguer deux cas :

1. L'opération sur P_1 et P_2 n'est pas une généralisation.
   Selon le tableau 6.1, nous pouvons descendre les inférences produisant P_1' à la fin de $\mathcal{P}$. Alors le prémisse de la dernière inférence a la forme $P_1,\ Q' \vdash 1$ et puisque $P_1 \overset{\mathcal{E}}{\sim} P_2$, nous en déduisons $P_2,\ Q' \vdash 1$ et enfin $P_2',\ Q \vdash 1$ en appliquant la même règle qu'à la fin de $\mathcal{P}$.

2. L'opération sur P_1 et P_2 est une généralisation.
   Alors $\mathcal{P}$ a la forme suivante :

$$\begin{array}{c} \vdots \\ \dfrac{P_1[t/x],\ Q' \vdash 1}{\forall x P_1,\ Q' \vdash 1}\ GEN_L \\ \vdots \\ \forall x P_1,\ Q \vdash 1 \end{array}$$

   Nous pouvons supposer que x n'est pas libre dans Q' et appliquer le lemme E.0.2. Donc il existe un co-processus Q" tel que Q"$[t/x] = Q'$ et le séquent $P_1,\ Q$" $\vdash 1$ est prouvable. Comme $P_1 \overset{\mathcal{E}}{\sim} P_2$, nous en déduisons : $P_2,\ Q$" $\vdash 1$ et $P_2[t/x],\ Q$"$[t/x] \vdash 1$ c'est-à-dire : $P_2[t/x],\ Q' \vdash 1$.
   Nous pouvons supposer que dans la preuve $\mathcal{P}$, les identificateurs liés aux inférences REI_L au-dessous de celle produisant $\forall x P_1$ ne sont pas libres dans $\forall x P_2$. Donc nous pouvons remplacer P_1 par P_2 dans $\mathcal{P}$ et nous obtenons une preuve de $\forall x P_1,\ Q \vdash 1$.

□

Le théorème suivant va nous aider dans la pratique à déterminer si deux processus sont équivalents relativement à un environnement donné.

Théorème 8.3.6 *$P_1 \overset{\mathcal{E}}{\sim} P_2$ si et seulement s'il existe deux réductions respectives de $\mathcal{I}_{\mathcal{E}}(P_1)$ et de $\mathcal{I}_{\mathcal{E}}(P_2)$ qui sont équivalentes.*

Preuve 8.3.6 *Elle découle immédiatement du théorème 8.1.2 et du corollaire 8.1.2.*

On peut particulariser ce théorème pour l'environnement $\mathcal{P}$ en utilisant les résultats concernant les interfaces linéaires. D'où :

Théorème 8.3.7 $P_1 \overset{\mathcal{P}}{\sim} P_2$ *si et seulement si pour tout processus linéaire L, la démontrabilité de $P_1 \vdash L$ est équivalente à celle de $P_2 \vdash L$.*

Preuve 8.3.7 *Elle découle immédiatement des théorèmes 8.2.2 et 8.2.5.*□

Nous allons illustrer ces relations de comparaison entre processus par quelques exemples qui nous aideront aussi à cerner les relations complexes avec la notion de bissimulation.

3.3 Exemples de comparaison de processus

Exemple 8.3.2 *Nous nous proposons de comparer la représentation $P_1 \equiv !a\underline{b}.1$ du tampon infini avec celle $P_2 \equiv \exists s\ (a.\underline{b}.\underline{s}.1 \otimes !(s.a.\underline{b}.\underline{s}.1) \otimes \underline{a}.b.s.1)$ du tampon à une place.*
Nous avons déjà montré que $\mathcal{I}_{\mathcal{P}}(P_1)$ se réduit à l'ensemble des processus de la forme $\underline{a}.b.\cdots\underline{a}.b.1$.
Il est facile de vérifier que P_2 vérifie cette interface mais P_2 possède des co-processus linéaires comme $\forall s\ (\underline{a}.b.s.a.\underline{b}.\underline{s}.1)$ qui ne sont pas des co-processus de P_1. Donc on a la relation :$P_2 \overset{\mathcal{P}}{>} P_1$.
Maintenant si l'on se place dans l'environnement $\mathcal{N}$, on a : $\mathcal{I}_{\mathcal{N}}(P_1) = \mathcal{I}_{\mathcal{N}}(P_2) = \{\underline{a}.b.\cdots\underline{a}.b.1\}$.
Dans le Π-calcul, P_1 et P_2 deviendraient respectivement les processus $!a.\overline{b}.0$ et $(s)(a.\overline{b}.\overline{s}.0 \mid !s.a.\overline{b}.\overline{s}.0 \mid \overline{a}.b.s.0)$ qui sont faiblement équivalents.

Exemple 8.3.3
Soit P_1 et P_2 les processus suivants :

$$P_1 = p.\underline{q}.1 \otimes r.\underline{s}.1$$

$$P_2 = (p.\underline{q}.r.\underline{s}.1)\&(r.\underline{s}.p.\underline{q}.1)$$

Nous commençons par les comparer du point de vue de leur interface absolue. Bien entendu, nous le faisons directement en étudiant la démontrabilité des séquents $P_1 \vdash P_2$ et $P_2 \vdash P_1$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{r.\underline{s}.1 \vdash r.\underline{s}.1}\,ID}{1,\ r.\underline{s}.1 \vdash r.\underline{s}.1}\,TERM_L}{q, \underline{q}.1,\ r.\underline{s}.1 \vdash \underline{q}.r.\underline{s}.1}\,SD_R}{\underline{q}.1,\ r.\underline{s}.1 \vdash \underline{q}.r.\underline{s}.1}\,SD_L}{p,\ p.\underline{q}.1,\ r.\underline{s}.1 \vdash \underline{q}.r.\underline{s}.1}\,RC_L}{p.\underline{q}.1,\ r.\underline{s}.1 \vdash p.\underline{q}.r.\underline{s}.1}\,RC_R \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{p.\underline{q}.1 \vdash p.\underline{q}.1}\,ID}{1,\ p.\underline{q}.1 \vdash p.\underline{q}.1}\,TERM_L}{s, \underline{s}.1,\ p.\underline{q}.1 \vdash \underline{s}.p.\underline{q}.1}\,SD_R}{\underline{s}.1,\ p.\underline{s}.1 \vdash \underline{q}.p.\underline{s}.1}\,SD_L}{r,\ r.\underline{s}.1,\ p.\underline{q}.1 \vdash \underline{s}.p.\underline{q}.1}\,RC_L}{r.\underline{s}.1,\ p.\underline{q}.1 \vdash r.\underline{s}.p.\underline{q}.1}\,RC_R}{\dfrac{r.\underline{s}.1,\ p.\underline{q}.1 \vdash (p.\underline{q}.r.\underline{s}.1)\&(r.\underline{s}.p.\underline{q}.1)}{r.\underline{s}.1 \otimes p.\underline{q}.1 \vdash (p.\underline{q}.r.\underline{s}.1)\&(r.\underline{s}.p.\underline{q}.1)}\,PAR_L}\,ALT_R$$

Nous avons donc $P_1 \vdash P_2$ mais : $P_1 \not\vdash P_2$. En conclusion : $P_1 > P_2$.
Comparons maintenant les deux processus du point de vue de leur interface relative. Nous allons commencer par en construire une réduction linéaire en utilisant l'algorithme qui est en filigrane dans la démonstration du théorème 8.2.5. Commençons par P_1.

$$\mathcal{I}_{\mathcal{P}}(p.\underline{q}.1) \sqsupseteq \{\underline{p}.q.1\}$$

$$\mathcal{I}_{\mathcal{P}}(r.\underline{s}.1) \sqsupseteq \{\underline{r}.s.1\}$$

Par entrelacement et ensuite par réduction, nous obtenons :

$$\mathcal{I}_{\mathcal{P}}(P_1) \sqsupseteq \{\underline{p}.q.\underline{r}.s.1,\ \underline{r}.s.\underline{p}.q.1\}$$

Maintenant, nous construisons une réduction linéaire de $\mathcal{I}_{\mathcal{P}}(P_2)$.

$$\mathcal{I}_{\mathcal{P}}(p.\underline{q}.r.\underline{s}.1) \sqsupseteq \{\underline{p}.q.\underline{r}.s.1\}$$

$$\mathcal{I}_{\mathcal{P}}(r.\underline{s}.1,\ p.\underline{q}.1) \sqsupseteq \{\underline{r}.s.\underline{p}.q.1\}$$

Par union, nous obtenons :

$$\mathcal{I}_{\mathcal{P}}(P_2) \sqsupseteq \{\underline{p}.q.\underline{r}.s.1,\ \underline{r}.s.\underline{p}.q.1\}$$

En conclusion, $P_1 \overset{\mathcal{P}}{\sim} P_2$.
Dans CCS, les processus correspondants $p.\overline{q}.0 \mid r.\overline{s}.0$ *et* $(p.\overline{q}.r.\overline{s}.0) \mid (r.\overline{s}.p.\overline{q}.0)$ *ne sont pas faiblement équivalents.*

Exemple 8.3.4
Soit un processus P de CPL quelconque. Comme le séquent $!P \vdash !P \otimes P$ *est prouvable, alors :* $!P \geq !P \otimes P$.
Voyons ce qu'il en est de la relation inverse. Elle est fausse dans l'absolu. Nous allons examiner ce qu'il en est relativement à l'environnement $\mathcal{P}$.
Considérons un co-processus Q de !P qui est un processus. Par définition, le séquent $!P,\ Q \vdash 1$ *est prouvable. Soit une preuve de ce séquent. Alors deux cas se présentent.*

1. *Q est totalement réductible et !P reste passif dans toute la preuve.*
2. *Q n'est pas totalement réductible et !P est donc nécessairement introduit par une inférence de type* REC_L *qui, selon le tableau 6.1, peut être descendue au maximum dans* $\mathcal{P}$*.Donc le séquent* $!P,\ P,\ Q \vdash 1$ *est prouvable et grâce à la règle* PAR_L, $!P \otimes P,\ Q \vdash 1$ *aussi.*

Par conséquent, si P n'est pas totalement réductible, alors : $!P \otimes P \overset{\mathcal{P}}{\not\geq} !P$.
Si l'on appelle $\mathcal{P}'$ *l'environnement constitué par les processus qui ne sont pas totalement réductibles, on a :* $!P \otimes P \overset{\mathcal{P}'}{\geq} !P$ *et donc :* $!P \otimes P \overset{\mathcal{P}'}{\sim} !P$ *Dans le* Π*-calcul,* $!P$ *et* $!P \mid P$ *sont deux processus congruents structurellement quel que soit P.*

Exemple 8.3.5 *Voyons si l'opération "." de préfixation d'un processus par une action est distributive par rapport à la composition alternative.*
Soit P_1 *et* P_2 *deux processus et M un message quelconques de CPL. Considérons deux cas selon que l'action qui va préfixer les processus est une émission ou une réception.*

- L'action est une émission
  Nous allons comparer les processus $\underline{M}.(P_1 \& P_2)$ *et* $(\underline{M}.P_1)\&(\underline{M}.P_2)$. *Dans le cas général, nous avons :* $\underline{M}.(P_1 \& P_2) > (\underline{M}.P_1)\&(\underline{M}.P_2)$. *Nous avons l'équivalence si* $P_1 \geq P_2$ *ou* $P_2 \geq P_1$.
- L'action est une réception
  Les processus $M.(P_1 \& P_2)$ *et* $(M.P_1)\&(M.P_2)$ *sont logiquement équivalents.*

Il est remarquable de constater que dans CCS [Milner, 1989], on n'a pas cette propriété. Ainsi les processus de CCS $a.(b.0+c.0)$ *et* $(a.b.0)+(a.c.0)$ *ne sont pas faiblement équivalents. Par contre, ils sont équivalents d'après leurs traces.*

Conclusion

Nous disposons donc maintenant d'une sémantique dénotationnelle pour CPL qui exprime l'interaction des processus avec le monde extérieur à travers la notion d'*interface*.
Nous avons exploité la relation de déduction de trois façons différentes :

- pour définir le rapport entre un processus et une interface avec la notion de ***satisfaction***;
- pour ***réduire*** une interface;
- pour comparer deux processus à travers la relation de ***réalisation***.

La notion de ***relativisation d'une interface par rapport à un environnement*** joue un rôle important dans la sémantique dénotationnelle de CPL.
Un point à approfondir est le lien entre celle-ci et la sémantique des phases de la logique linéaire [Girard, 1987]. Si la sémantique du Π-calcul de [Miller, 1992] peut être considérée comme une application de la sémantique des phases, la relation est plus complexe pour CPL.
Miller n' a pas développé de calcul sur les multi-ensembles de co-agents comme nous avons pu le faire sur les interfaces. Par contre, il a enrichi la notion de co-agent en s'appuyant sur la négation par l'échec. Si $\vdash P$, F n'est pas prouvable, Miller pose alors en axiome $\vdash P, \neg F$. Malheureusement, si $\vdash Q, F$ est prouvable, alors $\vdash P \oplus Q, F$ et $\vdash P \oplus Q, \neg F$ sont démontrables simultanément. Pour conserver la consistance du système, il doit introduire une hiérarchie entre les preuves qui complique beaucoup les choses.
[Kobayashi and Yonezawa, 1993] ont repris l'idée de co-agent de [Miller, 1992] pour définir une équivalence logique faible entre processus. Mais la classe de co-agents est nettement plus étendue ce qui donne une relation très forte donc d'un intérêt limité. On peut même dire qu'en utilisant plutôt l'opérateur ! que des équations avec point fixe pour définir les processus récursifs (voir à ce sujet l'exemple 1.2.1), ils auraient pu parfaitement identifier cette relation avec l'équivalence logique.
Pour obtenir des équivalences plus faibles, ils s'attachent à établir des relations davantage liées à l'exécution des calculs et inspirées par les travaux effectués à ce sujet dans les algèbres de processus : équivalences par tests [Hennessy and de Nicola, 1983], par bissimulation [Park, 1980] ou de traces.
Une question importante reste posée : quel rapport entre l'approche de la sémantique dénotationnelle des calculs de processus adoptée dans cette thèse et les approches classiques via la notion de bissimulation? Les exemples traités à la fin de ce chapitre montrent que la question est complexe. Il apparaît que dans CPL, la notion d'interaction entre un processus et l'extérieur est vue dans sa ***globalité*** indépendamment de la stratégie de réduction. Avec la bissimulation, l'interaction est envisagée pas à pas donc elle est très liée à la stratégie de réduction. Ainsi, dans le prochain chapitre, en fixant une stratégie de réduction synchrone, nous obtiendrons une équivalence entre processus plus forte que celle trouvée ici et qui se rapproche davantage de la bissimulation.
On pourrait en déduire un peu hâtivement que l'équivalence dans CPL est un affaiblissement de la notion de bissimulation. Ce n'est pas vrai car deux processus peuvent être équivalents au sens de la bissimulation et pas logiquement équivalents dans CPL. Le contraire peut être vrai aussi.

Chapitre 9

SCPL : une restriction synchrone de CPL

Introduction

Dans CPL, la communication est asynchrone. Or, dans certains phénomènes, la notion de temps, d'ordre dans les évènements est fondamentale et exige une modélisation où la communication est représentée sous une forme synchrone : l'émission et la réception d'un message ne sont plus alors indépendantes mais elles sont liées ce qui se ramène à les considérer comme deux aspects d'une seule et unique action [Milner, 1983]. Les calculs qui ont marqué l'histoire du développement de la théorie du parallélisme, CSP [Hoare, 1985], CCS [Milner, 1980], la machine chimique abstraite [Berry and Boudol, 1990] et le Π-calcul [Milner *et al.*, 1992] sont tous primitivement des calculs synchrones.
Dans ce chapitre, nous verrons qu'il est possible d'*exprimer le synchronisme dans CPL par une restriction sur le système déductif.* Nous partirons tout d'abord des réductions de processus qui dans CPL, sont des preuves de séquents de la forme $P \vdash 1$. Nous verrons que synchroniser la communication dans celles-ci, revient à rapprocher les inférences produisant des messages de celles les consommant. Généralisant cette remarque à toute preuve de CPL, nous distinguerons les *preuves synchrones* des *preuves asynchrones.* Nous verrons qu'il n'est pas toujours possible de transformer une preuve asynchrone en une preuve synchrone. Nous verrons ensuite qu'il est possible de spécialiser le système d'inférence de CPL pour représenter uniquement les preuves synchrones.
Nous obtiendrons ainsi un nouveau calcul de processus, SCPL (Synchronous Concurrent Programming Logic), issu de CPL par une restriction du système déductif et secondairement de la syntaxe des processus qui ne peuvent plus être des messages.
L'inconvénient est qu'il n'est *pas complet* relativement à la logique linéaire : tout séquent de SCPL prouvable en logique linéaire n'est pas nécessairement prouvable dans le système d'inférence de SCPL. En particulier, tout processus de SCPL réductible dans CPL, ne l'est plus forcément dans SCPL.
Pour éviter ce problème, nous explorerons une autre voie de définition d'un calcul synchrone à partir de CPL qui consiste à restreindre non pas le système déductif mais *la syntaxe des processus.* Nous verrons qu'il faut être prudent à ce sujet, les choses étant plus compliquées qu'il n'y paraît. La solution obtenue est relativement lourde.
Nous reprendrons ensuite le même cheminement que celui suivi pour définir CPL : établissement d'une *sémantique opérationnelle* puis d'une *sémantique dénotationnelle.*
Enfin, nous illustrerons le pouvoir d'expression de SCPL en y traduisant le Π-calcul polyadique [Milner, 1991].

1 Stratégies de preuves synchrones : le système formel de SCPL

1.1 Réductions et preuves de CPL synchrones et asynchrones

Dans l'introduction, nous avons dit ce que nous entendions par communication synchrone. Voyons tout d'abord comment le phénomène se manifeste dans les réductions totales de processus de CPL. Considérons une réduction totale $\mathcal{R}_1$ du système de processus $\underline{a}.\underline{b}.1,\ a.b.1$.

$$
\begin{array}{l}
\dfrac{}{\vdash 1}\ TERM_R \\[4pt]
\dfrac{\vdash 1}{1 \vdash 1}\ TERM_L \\[4pt]
\dfrac{1 \vdash 1}{b,\ b.1 \vdash 1}\ RC_L \\[4pt]
\dfrac{b,\ b.1 \vdash 1}{a,\ b,\ a.b.1 \vdash 1}\ RC_L \\[4pt]
\dfrac{a,\ b,\ a.b.1 \vdash 1}{a,\ b,\ 1,\ a.b.1 \vdash 1}\ TERM_L \\[4pt]
\dfrac{a,\ b,\ 1,\ a.b.1 \vdash 1}{a,\ \underline{b}.1,\ a.b.1 \vdash 1}\ SD_L \\[4pt]
\dfrac{a,\ \underline{b}.1,\ a.b.1 \vdash 1}{\underline{a}.\underline{b}.1,\ a.b.1 \vdash 1}\ SD_L
\end{array}
$$

Dans cette réduction, la transmission des messages a et b n'est pas synchrone. Nous dirons que $\mathcal{R}_1$ est une *réduction asynchrone*. En rapprochant les inférences de type SD_L des inférences correspondantes de type RC_L, nous pouvons d'une certaine façon synchroniser la communication dans $\mathcal{R}_1$. Nous obtenons la réduction totale $\mathcal{R}_2$ suivante :

$$
\begin{array}{l}
\dfrac{}{\vdash 1}\ TERM_R \\[4pt]
\dfrac{\vdash 1}{1 \vdash 1}\ TERM_L \\[4pt]
\dfrac{1 \vdash 1}{1,\ 1 \vdash 1}\ TERM_L \\[4pt]
\dfrac{1,\ 1 \vdash 1}{b,\ 1,\ b.1 \vdash 1}\ RC_L \\[4pt]
\dfrac{b,\ 1,\ b.1 \vdash 1}{\underline{b}.1,\ b.1 \vdash 1}\ SD_L \\[4pt]
\dfrac{\underline{b}.1,\ b.1 \vdash 1}{a,\ \underline{b}.1,\ a.b.1 \vdash 1}\ RC_L \\[4pt]
\dfrac{a,\ \underline{b}.1,\ a.b.1 \vdash 1}{\underline{a}.\underline{b}.1,\ a.b.1 \vdash 1}\ SD_L
\end{array}
$$

Maintenant, les inférences de type SD_L suivent immédiatement les inférences de type RC_L correspondantes. Nous dirons que $\mathcal{R}_2$ est une *réduction synchrone*.
Mais il n'est pas toujours possible de transformer une réduction asynchrone en une réduction synchrone. Considérons une réduction totale $\mathcal{R}_3$ du système de processus $\underline{a}.\underline{b}.1,\ b.a.1$:

$$
\begin{array}{l}
\dfrac{}{\vdash 1}\ TERM_R \\[4pt]
\dfrac{\vdash 1}{1 \vdash 1}\ TERM_L \\[4pt]
\dfrac{1 \vdash 1}{a,\ a.1 \vdash 1}\ RC_L \\[4pt]
\dfrac{a,\ a.1 \vdash 1}{a,\ b,\ b.a.1 \vdash 1}\ RC_L \\[4pt]
\dfrac{a,\ b,\ b.a.1 \vdash 1}{a,\ b,\ 1,\ b.a.1 \vdash 1}\ TERM_L \\[4pt]
\dfrac{a,\ b,\ 1,\ b.a.1 \vdash 1}{a,\ \underline{b}.1,\ b.a.1 \vdash 1}\ SD_L \\[4pt]
\dfrac{a,\ \underline{b}.1,\ b.a.1 \vdash 1}{\underline{a}.\underline{b}.1,\ b.a.1 \vdash 1}\ SD_L
\end{array}
$$

Il est évident qu'il n'est pas possible de synchroniser la communication dans $\mathcal{R}_3$.
Par ailleurs, l'exemple suivant montre qu'il ne suffit pas que les inférences de type SD_L suivent immédiatement les inférences correspondantes de type RC_L pour qu'on puisse parler de communication synchrone. Considérons la réduction suivante :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash 1}\,TERM_R}{1 \vdash 1}\,TERM_L}{a,\ a.1 \vdash 1}\,RC_L}{\underline{a}.a.1 \vdash 1}\,SD_L$$

On ne peut pas parler ici de réduction synchrone car l'émetteur du message a devient ensuite le récepteur du même message. Il n'est donc pas possible d'interpréter les inférences SD_L et RC_L comme émission et réception simultanée du message a.
Après ces exemples introductifs, nous pouvons maintenant donner une définition générale qui dépasse même le cadre des réductions totales pour s'appliquer à toute preuve de CPL.

Définition 9.1.1 *Une preuve de CPL est dite* synchrone *si et seulement si toute inférence introduisant un message est suivie immédiatement d'une inférence utilisant comme formule active le même message et si le récepteur est indépendant de l'émetteur.*
Dans le cas contraire, nous dirons que la preuve est asynchrone.

1.2 La syntaxe et le système déductif de SCPL

L'idée est de spécialiser le système d'inférence de CPL de façon à n' obtenir que des preuves synchrones. Auparavant, il faut restreindre légèrement la syntaxe des processus de CPL. En effet, dans SCPL, les messages seront consommés dès leur émission donc ils n'ont pas d'existence propre en tant que processus.

Définition 9.1.2 *Un* processus de SCPL *est un processus strict de CPL, c'est-à-dire un processus qui n'est pas un message.*
Un système de processus de SCPL *est un multi-ensemble de processus de SCPL.*
Un séquent de SCPL *est un séquent dont la partie gauche est un système de processus de SCPL et dont la partie droite est un processus de SCPL.*

Le système d'inférence de SCPL est destiné à produire les preuves synchrones de CPL. Comme elles se caractérisent par un couplage de chaque inférence produisant un message avec celle qui le consomme. Il suffit de fusionner les deux inférences en une seule pour obtenir une preuve de SCPL. Les règles de communication sont donc modifiées en conséquence mais par contre les règles d'opération restent inchangées. La figure 9.1 donne une présentation complète du système d'inférence de SCPL.

Le théorème suivant garantit la correction de SCPL relativement à la logique linéaire.

Théorème 9.1.1 *Tout séquent prouvable dans le système d'inférence de SCPL est prouvable en logique linéaire.*

Preuve 9.1.1 *Les règles de communication de SCPL sont les seules nouvelles. Or, il est évident qu'elles sont admissibles dans CPL. Donc SCPL est correct relativement à CPL et comme CPL est lui-même correct relativement à la logique linéaire, on a par transition le résultat cherché.*□

Comme nous l'avons déjà indiqué, le système d'inférence de SCPL n'est pas complet par rapport à la logique linéaire. Ainsi, le séquent de SCPL $\underline{a}.\underline{b}.1,\ b.a.1 \vdash 1$ n'est pas prouvable dans SCPL mais il est prouvable en logique linéaire. La symétrie du système d'inférence de SCPL entraîne que celui-ci vérifie la propriété d'élimination des coupures qui est essentielle au fondement de la sémantique opérationnelle et de la sémantique dénotationnelle sur la notion de déduction.

Théorème 9.1.2 *Tout séquent prouvable dans SCPL admet dans celui-ci une preuve qui n'utilise pas la règle $COMP$.*

Règles de communication :

Communication effective

$$\frac{P_1,\ P_2,\ \Gamma \vdash P'}{\underline{M}.P_1,\ M.P_2,\ \Gamma \vdash P'}\ COM_L$$

Transfert d'émission

$$\frac{P,\ \Gamma \vdash P'}{\underline{M}.P,\ \Gamma \vdash \underline{M}.P'}\ SDt$$

Transfert de réception

$$\frac{P,\ \Gamma \vdash P'}{M.P,\ \Gamma \vdash M.P'}\ RCt$$

Règles d'opération

Terminaison

$$\frac{\Gamma \vdash P'}{1,\ \Gamma \vdash P'}\ TERM_L \qquad \frac{}{!\Gamma \vdash 1}\ TERM_R$$

Interruption

$$\frac{}{0,\ \Gamma \vdash P}\ BRK$$

Composition parallèle

$$\frac{P_1,\ P_2,\ \Gamma \vdash P'}{P_1 \otimes P_2,\ \Gamma \vdash P'}\ PAR_L \qquad \frac{\Gamma_1,\ !\Gamma \vdash P_1' \quad \Gamma_2,\ !\Gamma \vdash P_2'}{\Gamma_1,\ \Gamma_2,\ !\Gamma \vdash P_1' \otimes P_2'}\ PAR_R$$

où Γ_1 et Γ_2 ne contiennent pas de processus récurrents ;

Composition alternative

$$\frac{P_1,\ \Gamma \vdash P'}{P_1 \& P_2,\ \Gamma \vdash P'}\ ALT_L \qquad \frac{\Gamma \vdash P_1' \quad \Gamma \vdash P_2'}{\Gamma \vdash P_1' \& P_2'}\ ALT_R$$

Récurrence

$$\frac{!P,\ P,\ \Gamma \vdash P'}{!P,\ \Gamma \vdash P'}\ REC_L \qquad \frac{!\Gamma \vdash P'}{!\Gamma \vdash !P'}\ REC_R$$

Restriction

$$\frac{P[y/x],\ \Gamma \vdash P'}{\exists x P,\ \Gamma \vdash P'}\ RES_L \qquad \frac{\Gamma \vdash P'[t/x]}{\Gamma \vdash \exists x P'}\ RES_R$$

Généralisation

$$\frac{P[t/x],\ \Gamma \vdash P'}{\forall x P,\ \Gamma \vdash P'}\ GEN_L \qquad \frac{\Gamma \vdash P'[y/x]}{\Gamma \vdash \forall x P'}\ GEN_R$$

Pour les règles RES_L et GEN_R, la variable y ne doit pas être libre dans la conclusion

Règle de composition

$$\frac{\Gamma_1,\ !\Gamma \vdash P \quad P,\ \Gamma_2,\ !\Gamma \vdash P'}{\Gamma_1,\ \Gamma_2,\ !\Gamma \vdash P'}\ COMP$$

où Γ_1 et Γ_2 ne contiennent pas de processus récurrents ;

FIG. *9.1* - Le système d'inférence de SCPL

Preuve 9.1.2 *Elle s'effectue de la même façon que celle du théorème d'élimination des coupures dans CLL qui a été présentée dans la première partie de cette thèse, par une montée des inférences de type COMP dans les preuves de SCPL.*□

2 De la synchronisation intégrée à la syntaxe des processus

Le système déductif de SCPL n'est pas complet par rapport à la logique linéaire. Voyons s'il n'existe pas une voie plus intéressante qui consisterait à restreindre la syntaxe sur les processus de CPL de telle façon que les seuls séquents prouvables dans CPL y seraient ceux pour lesquels existent des preuves synchrones. CPL étant correct et complet vis-à-vis de la logique linéaire, nous aurions ainsi un système synchrone correct et complet vis-à-vis de la logique linéaire.

2.1 Des solutions qui n'en sont pas

Une première idée consiste à utiliser des messages jouant le rôle d'accusé de réception. Un émetteur aurait la forme $\underline{M_1}.M_2.P_1$ et un récepteur la forme $M_1.\underline{M_2}.P_2$. Le premier envoie le message M_1 et est bloqué tant qu'il n'a pas reçu l'accusé de réception M_2. Le second attend le message M_1 et, une fois qu'il l'a reçu, l'indique en retournant l'accusé de réception M_2. Malheureusement, cette solution comporte une faille : plusieurs processus peuvent envoyer des messages identiques ce qui peut brouiller les pistes quant aux destinataires.
Cherchons par exemple à réduire le processus :

$$\underline{M_1}.M_2.1,\ M_1.\underline{M_2}.(\underline{M_1}.M_2.K_1.\underline{K_2}.1 \otimes \underline{K_1}.K_2.M_1.\underline{M_2}.1)$$

Commençons par le faire de façon synchrone dans CPL ou ce qui revient au même dans SCPL (nous noterons en caractères gras les formules principales de chaque inférence).

$$\dfrac{\dfrac{\dfrac{\dfrac{\underline{M_1}.M_2.K_1.\underline{K_2}.1,\ \underline{K_1}.K_2.M_1.\underline{M_2}.1 \vdash 1}{\underline{M_1}.M_2.K_1.\underline{K_2}.1 \otimes \underline{K_1}.K_2.M_1.\underline{M_2}.1 \vdash 1}\,PAR_L}{\mathbf{1},\ \underline{M_1}.M_2.K_1.\underline{K_2}.1 \otimes \underline{K_1}.K_2.M_1.\underline{M_2}.1 \vdash 1}\,TERM_L}{\mathbf{M_2}.1,\ \underline{\mathbf{M_2}}.(\underline{M_1}.M_2.K_1.\underline{K_2}.1 \otimes \underline{K_1}.K_2.M_1.\underline{M_2}.1) \vdash 1}\,COM_L}{\underline{\mathbf{M_1}}.M_2.1,\ \mathbf{M_1}.\underline{M_2}.(\underline{M_1}.M_2.K_1.\underline{K_2}.1 \otimes \underline{K_1}.K_2.M_1.\underline{M_2}.1) \vdash 1}\,COM_L$$

La réduction est bloquée ; elle ne peut pas terminer.
Essayons maintenant de réduire le système de processus de façon asynchrone.

$$
\begin{array}{l}
\dfrac{}{\vdash \mathbf{1}}\,TERM_R \\[2ex]
\dfrac{\vdash \mathbf{1}}{\mathbf{1} \vdash 1}\,TERM_L \\[2ex]
\dfrac{\mathbf{1} \vdash 1}{\mathbf{1},\ 1, \vdash 1}\,TERM_L \\[2ex]
\dfrac{\mathbf{1},\ 1, \vdash 1}{\mathbf{M_2}.1,\ \underline{\mathbf{M_2}}.1, \vdash 1}\,COM_L \\[2ex]
\dfrac{\mathbf{M_2}.1,\ \underline{\mathbf{M_2}}.1, \vdash 1}{M_2.1,\ \mathbf{M_1},\ \mathbf{M_1}.\underline{M_2}.1, \vdash 1}\,RC_L \\[2ex]
\dfrac{M_2.1,\ \mathbf{M_1},\ \mathbf{M_1}.\underline{M_2}.1, \vdash 1}{M_2.1,\ M_1,\ \mathbf{K_2},\ \mathbf{K_2}.M_1.\underline{M_2}.1, \vdash 1}\,RC_L \\[2ex]
\dfrac{M_2.1,\ M_1,\ \mathbf{K_2},\ \mathbf{K_2}.M_1.\underline{M_2}.1, \vdash 1}{M_2.1,\ M_1,\ K_2,\ \mathbf{1},\ K_2.M_1.\underline{M_2}.1, \vdash 1}\,TERM_L \\[2ex]
\dfrac{M_2.1,\ M_1,\ K_2,\ \mathbf{1},\ K_2.M_1.\underline{M_2}.1, \vdash 1}{M_2.1,\ M_1,\ \underline{\mathbf{K_2}}.1,\ K_2.M_1.\underline{M_2}.1, \vdash 1}\,SD_L \\[2ex]
\dfrac{M_2.1,\ M_1,\ \underline{\mathbf{K_2}}.1,\ K_2.M_1.\underline{M_2}.1, \vdash 1}{M_2.1,\ M_1,\ \mathbf{K_1},\ \mathbf{K_1}.\underline{K_2}.1,\ K_2.M_1.\underline{M_2}.1, \vdash 1}\,RC_L \\[2ex]
\dfrac{M_2.1,\ M_1,\ \mathbf{K_1},\ \mathbf{K_1}.\underline{K_2}.1,\ K_2.M_1.\underline{M_2}.1, \vdash 1}{M_2.1,\ M_1,\ K_1,\ \mathbf{M_2},\ \mathbf{M_2}.K_1.\underline{K_2}.1,\ K_2.M_1.\underline{M_2}.1, \vdash 1}\,RC_L \\[2ex]
\dfrac{M_2.1,\ M_1,\ K_1,\ \mathbf{M_2},\ \mathbf{M_2}.K_1.\underline{K_2}.1,\ K_2.M_1.\underline{M_2}.1, \vdash 1}{M_2.1,\ K_1,\ M_2,\ \underline{\mathbf{M_1}}.M_2.K_1.\underline{K_2}.1,\ K_2.M_1.\underline{M_2}.1, \vdash 1}\,SD_L \\[2ex]
\dfrac{M_2.1,\ K_1,\ M_2,\ \underline{\mathbf{M_1}}.M_2.K_1.\underline{K_2}.1,\ K_2.M_1.\underline{M_2}.1, \vdash 1}{M_2.1,\ M_2,\ \underline{M_1}.M_2.K_1.\underline{K_2}.1,\ \underline{\mathbf{K_1}}.K_2.M_1.\underline{M_2}.1, \vdash 1}\,SD_L \\[2ex]
\dfrac{M_2.1,\ M_2,\ \underline{M_1}.M_2.K_1.\underline{K_2}.1,\ \underline{\mathbf{K_1}}.K_2.M_1.\underline{M_2}.1, \vdash 1}{M_2.1,\ M_2,\ \underline{M_1}.M_2.K_1.\underline{K_2}.1 \otimes \underline{K_1}.K_2.M_1.\underline{M_2}.1, \vdash 1}\,PAR_L \\[2ex]
\dfrac{M_2.1,\ M_2,\ \underline{M_1}.M_2.K_1.\underline{K_2}.1 \otimes \underline{K_1}.K_2.M_1.\underline{M_2}.1, \vdash 1}{M_2.1,\ \underline{\mathbf{M_2}}.(\underline{M_1}.M_2.K_1.\underline{K_2}.1 \otimes \underline{K_1}.K_2.M_1.\underline{M_2}.1) \vdash 1}\,SD_L \\[2ex]
\dfrac{M_2.1,\ \underline{\mathbf{M_2}}.(\underline{M_1}.M_2.K_1.\underline{K_2}.1 \otimes \underline{K_1}.K_2.M_1.\underline{M_2}.1) \vdash 1}{\underline{\mathbf{M_1}}.M_2.1,\ \mathbf{M_1}.\underline{M_2}.(\underline{M_1}.M_2.K_1.\underline{K_2}.1 \otimes \underline{K_1}.K_2.M_1.\underline{M_2}.1) \vdash 1}\,COM_L
\end{array}
$$

La réduction totale est possible donc l'utilisation d'accusés de réception ne garantit pas la synchronisation à 100%.
[Kobayashi and Yonezawa, 1994a] ont alors imaginé un mécanisme plus sophistiqué utilisant une réponse à l'accusé de réception. Un émetteur aurait alors la forme $\underline{M_1}.M_2.\underline{M_3}.P_1$ et un récepteur la forme $M_1.\underline{M_2}.M_3.P_2$. Le premier envoie le message M_1 et est bloqué en attendant l'accusé de réception M_2. Après l'avoir reçu, il l'indique en retournant une réponse M_3. Le second attend le message M_1 puis après l'avoir reçu, envoie l'accusé de réception M_2. Ensuite, il est bloqué tant qu'il n'a pas reçu la réponse M_3. La synchronisation n'est pas garantie par un tel mécanisme comme le montre l'exemple suivant qui prouve que le théorème 5.1 de [Kobayashi and Yonezawa, 1994a] est incorrect.
Considérons les processus P_1 et P_2 suivants :

$$\underline{M_1}.M_2.\underline{M_3}.1 \quad \text{et} \quad M_1.\underline{M_2}.M_3.\underline{M_1}.M_2.\underline{M_3}.M_1.\underline{M_2}.M_3.1$$

Nous allons chercher à réduire totalement le système de processus

$$\underline{M_1}.M_2.\underline{M_3}.(P_1 \otimes P_2),\ M_1.\underline{M_2}.M_3.\underline{M_1}.M_2.\underline{M_3}.M_1.\underline{M_2}.M_3.1$$

Nous allons d'abord essayer de la faire de façon synchrone.

$$
\begin{array}{cl}
\underline{M_1}.M_2.\underline{M_3}.M_1.\underline{M_2}.M_3.1 \vdash 1 & TERM_L \\ \hline
\mathbf{1},\ \underline{M_1}.M_2.\underline{M_3}.M_1.\underline{M_2}.M_3.1 \vdash 1 & COM_L \\ \hline
\underline{\mathbf{M_3}}.1,\ \mathbf{M_3}.\underline{M_1}.M_2.\underline{M_3}.M_1.\underline{M_2}.M_3.1 \vdash 1 & COM_L \\ \hline
\mathbf{M_2}.\underline{M_3}.1,\ \underline{\mathbf{M_2}}.M_3.\underline{M_1}.M_2.\underline{M_3}.M_1.\underline{M_2}.M_3.1 \vdash 1 & COM_L \\ \hline
\underline{M_1}.M_2.\underline{M_3}.1,\ (M_1.\underline{M_2}.M_3.\underline{M_1}.M_2.\underline{M_3}.M_1.\underline{M_2}.M_3.1) \vdash 1 & PAR_L \\ \hline
(\underline{M_1}.M_2.\underline{M_3}.1) \otimes (M_1.\underline{M_2}.M_3.\underline{M_1}.M_2.\underline{M_3}.M_1.\underline{M_2}.M_3.1) \vdash 1 & TERM_L \\ \hline
(\underline{M_1}.M_2.\underline{M_3}.1) \otimes (M_1.\underline{M_2}.M_3.\underline{M_1}.M_2.\underline{M_3}.M_1.\underline{M_2}.M_3.1),\ \mathbf{1} \vdash 1 & COM_L \\ \hline
\underline{\mathbf{M_3}}.(\underline{M_1}.M_2.\underline{M_3}.1) \otimes (M_1.\underline{M_2}.M_3.\underline{M_1}.M_2.\underline{M_3}.M_1.\underline{M_2}.M_3.1)),\ \mathbf{M_3}.1 \vdash 1 & COM_L \\ \hline
\mathbf{M_2}.\underline{M_3}.((\underline{M_1}.M_2.\underline{M_3}.1) \otimes (M_1.\underline{M_2}.M_3.\underline{M_1}.M_2.\underline{M_3}.M.\underline{M_2}.M_3.1)),\ \underline{\mathbf{M_2}}.M_3.1 \vdash 1 & COM_L \\ \hline
\underline{\mathbf{M_1}}.M_2.\underline{M_3}.((\underline{M_1}.M_2.\underline{M_3}.1) \otimes (M_1.\underline{M_2}.M_3.\underline{M_1}.M_2.\underline{M_3}.M_1.\underline{M_2}.M_3.1)),\ \mathbf{M_1}.\underline{M_2}.M_3.1 \vdash 1 &
\end{array}
$$

La réduction ne termine pas. Essayons donc maintenant de le faire de manière asynchrone.

$$
\begin{array}{cl}
 & TERM_R \\ \hline
\vdash \mathbf{1} & TERM_L \\ \hline
\mathbf{1} \vdash 1 & TERM_L \\ \hline
\mathbf{1},\ 1 \vdash 1 & COM_L \\ \hline
\underline{\mathbf{M_3}}.1,\ \mathbf{M_3}.1 \vdash 1 & COM_L \\ \hline
\mathbf{M_2}.\underline{M_3}.1,\ \underline{\mathbf{M_2}}.M_3.1 \vdash 1 & RC_L \\ \hline
\mathbf{M_1},\ M_2.\underline{M_3}.1,\ \mathbf{M_1}.\underline{M_2}.M_3.1 \vdash 1 & TERM_L \\ \hline
M_1,\ M_2.\underline{M_3}.1,\ M_1.\underline{M_2}.M_3.1,\ \mathbf{1} \vdash 1 & COM_L \\ \hline
M_1,\ M_2.\underline{M_3}.1,\ \underline{\mathbf{M_3}}.M_1.\underline{M_2}.M_3.1,\ \mathbf{M_3}.1 \vdash 1 & RC_L \\ \hline
M_1,\ \mathbf{M_2},\ \mathbf{M_2}.\underline{M_3}.1,\ M_2.\underline{M_3}.M_1.\underline{M_2}.M_3.1,\ M_3.1 \vdash 1 & SD_L \\ \hline
M_2,\ M_2.\underline{M_3}.1,\ \underline{\mathbf{M_1}}.M_2.\underline{M_3}.M_1.\underline{M_2}.M_3.1,\ M_3.1 \vdash 1 & RC_L \\ \hline
\mathbf{M_3},\ M_2,\ M_2.\underline{M_3}.1,\ \mathbf{M_3}.\underline{M_1}.M_2.\underline{M_3}.M_1.\underline{M_2}.M_3.1,\ M_3.1 \vdash 1 & SD_L \\ \hline
M_3,\ M_2.\underline{M_3}.1,\ \underline{\mathbf{M_2}}.M_3.\underline{M_1}.M_2.\underline{M_3}.M_1.\underline{M_2}.M_3.1,\ M_3.1 \vdash 1 & COM_L \\ \hline
M_3,\ \underline{\mathbf{M_1}}.M_2.\underline{M_3}.1,\ \mathbf{M_1}.\underline{M_2}.M_3.\underline{M_1}.M_2.\underline{M_3}.M.\underline{M_2}.M_3.1,\ M_3.1 \vdash 1 & PAR_L \\ \hline
M_3,\ (\underline{M_1}.M_2.\underline{M_3}.1) \otimes (M_1.\underline{M_2}.M_3.\underline{M_1}.M_2.\underline{M_3}.M.\underline{M_2}.M_3.1),\ M_3.1 \vdash 1 & SD_L \\ \hline
\underline{\mathbf{M_3}}.((\underline{M_1}.M_2.\underline{M_3}.1) \otimes (M_1.\underline{M_2}.M_3.\underline{M_1}.M_2.\underline{M_3}.M.\underline{M_2}.M_3.1)),\ \underline{M_3}.M_2.1 \vdash 1 & COM_L \\ \hline
\mathbf{M_2}.\underline{M_3}.((\underline{M_1}.M_2.\underline{M_3}.1) \otimes (M_1.\underline{M_2}.M_3.\underline{M_1}.M_2.\underline{M_3}.M.\underline{M_2}.M_3.1)),\ \underline{\mathbf{M_2}}.M_3.1 \vdash 1 & COM_L \\ \hline
\underline{\mathbf{M_1}}.M_2.\underline{M_3}.((\underline{M_1}.M_2.\underline{M_3}.1) \otimes (M_1.\underline{M_2}.M_3.\underline{M_1}.M_2.\underline{M_3}.M.\underline{M_2}.M_3.1)),\ \mathbf{M_1}.\underline{M_2}.M_3.1 \vdash 1 &
\end{array}
$$

La réduction totale asynchrone du système de processus est donc possible. Le mécanisme proposé par [Kobayashi and Yonezawa, 1994a] n'est donc pas sûr. Si l'on veut intégrer le synchronisme dans la syntaxe des processus, il faut envisager les choses différemment.

2.2 Solution proposée

Une solution sûre consiste à simuler la communication synchrone par le moyen d'un processus spécial $P_{synchro}$ jouant le rôle de synchroniseur.
Tout processus qui souhaite envoyer un message, émet une requête au processus $P_{synchro}$ par le biais du canal *synchro in*. Ce dernier reçoit alors le nombre n de messages déjà envoyés sur le canal *nb-messages*. Il incrémente ce nombre de 1 puis le retourne au processus qui a fait la requête sur le canal *synchro out* et le conserve aussi en mémoire par le biais du canal *nb-messages*. Le processus qui souhaite envoyer le message, peut alors le marquer du numéro $n+1$. Il attendra ensuite que le processus qui reçoit le message lui retourne comme accusé de réception le numéro $n+1$ sur un canal spécial réservé à cette tâche, *reception*. Nous avons défini ci-dessous dans CPL $P_{synchro}$, un émetteur quelconque P_s et un récepteur P_r.

$$
\begin{array}{l}
P_{synchro} = \otimes\,!\ (\text{synchro in}:).\\
\qquad\qquad \forall\ \text{n}\ (\text{nb-messages}:\text{n}).\\
\qquad\qquad\qquad \underline{(\text{synchro out}:\text{n+1})}.\\
\qquad\qquad\qquad \underline{(\text{nb-messages out}:\text{n+1})}.\\
\qquad\qquad\qquad \underline{1}\\
\qquad\qquad \otimes\ \underline{(\text{nb-messages}:0)}\\
\qquad\qquad\quad \underline{1}
\end{array}
$$

$$
\begin{array}{l}
P_s = \underline{(\text{synchro in}:)}.\\
\qquad \forall\ \text{n}\ (\text{synchro out}:\text{n}).\\
\qquad\quad \underline{(\text{c}:\text{n t})}.\\
\qquad\quad \underline{(\text{reception}:\text{n})}.\\
\qquad\quad \text{P}
\end{array}
$$

$$
\begin{array}{l}
P_r = \forall\ \text{n x}\ (\text{c}:\text{n x}).\\
\qquad\qquad \underline{(\text{reception}:\text{n})}.\\
\qquad\qquad \text{P'}
\end{array}
$$

La solution proposée ici est relativement lourde si bien que nous avons préféré nous en tenir à la solution de départ.

3 La sémantique opérationnelle de SCPL

3.1 Définition

Il est possible de la définir à travers une relation de transition $\rightsquigarrow$ qui découle de la relation de transition de CPL. La relation de réduction structurelle $\Rightarrow$ est exactement la même. Quant à la relation de transition, l'unique différence se situe au niveau des axiomes qui définissent les actions de communication. Dans CPL, nous avions deux axiomes : un, SD, pour l'émission et un autre, RC, pour la réception. Ici, comme les deux actions se confondent, nous n'avons plus qu'un seul axiome COM qui effectue en même temps émission et réception. Les autres règles ne sont pas modifiées.
La relation de transition $\rightsquigarrow$ est définie par la figure 9.2.
Les liens entre la relation de transition et celle de déduction ne changent pas parce que nous sommes passés de CPL à SCPL. Ainsi, les théorèmes 7.2.5, 7.2.6 et 7.2.7 peuvent être transposés tels quels dans SCPL.

$$\frac{}{\underline{M}.P_1 \otimes M.P_2 \rightsquigarrow P_1 \otimes P_2}\ COM$$

$$\frac{P \rightsquigarrow P'}{P \otimes Q \rightsquigarrow P' \otimes Q}\ PAR$$

$$\frac{P[y/x] \otimes Q \rightsquigarrow P'[y/z] \otimes Q'}{\exists x P \otimes Q \rightsquigarrow \exists z P' \otimes Q'}\ RES$$

avec y non libre dans $\exists x P$, Q et $\exists z P'$, Q'

$$\frac{P \Rightarrow P_1 \quad P_1 \rightsquigarrow P_1' \quad P_1' \Rightarrow P'}{P \rightsquigarrow P'}\ STRUCT$$

FIG. 9.2 - **Relation de transition entre processus de SCPL**

3.2 Exemples

Pour illustrer la définition de la relation de transition dans SCPL, reprenons les exemples 7.2.6 et 7.2.7 déjà étudiés dans CPL.

Exemple 9.3.1 Intrusion dans un champ
Nous nous proposons de prouver dans SCPL la transition :

$$\underline{m(y,x)}.P' \otimes R \otimes \exists x((\forall z\ m(y,z).Q') \otimes S) \rightsquigarrow P' \otimes R \otimes \exists x'(Q'[x'/x][x/z] \otimes S[x'/x])$$

Nous rappelons qu'elle est la traduction dans SCPL de la transition suivante du Π*-calcul :*

$$\overline{y}x.P'|R|(x)(y(z).Q'|S) \xrightarrow{\tau} P'|R|(x')(Q'\{x'/x\}\{x/z\}|S\{x'/x\}).$$

Voici la preuve de la transition :

$$\frac{\dfrac{\dfrac{\dfrac{\dfrac{}{\underline{m(y,x)}.P' \otimes m(y,x).Q'[x'/x][x/z] \rightsquigarrow P' \otimes Q'[x'/x][x/z]}\ COM}{\underline{m(y,x)}.P' \otimes \forall z\ m(y,z).Q'[x'/x] \rightsquigarrow P' \otimes Q'[x'/x][x/z]}\ STRUCT}{\underline{m(y,x)}.P' \otimes (\forall z\ m(y,z).Q'[x'/x]) \otimes S[x'/x] \rightsquigarrow P' \otimes Q'[x'/x][x/z] \otimes S[x'/x]}\ PAR}{\underline{m(y,x)}.P' \otimes \exists x((\forall z\ m(y,z).Q') \otimes S) \rightsquigarrow P' \otimes \exists x'(Q'[x'/x][x/z] \otimes S[x'/x])}\ RES}{\underline{m(y,x)}.P' \otimes R \otimes \exists x((\forall z\ m(y,z).Q') \otimes S) \rightsquigarrow P' \otimes R \otimes \exists x'(Q'[x'/x][x/z] \otimes S[x'/x])}\ PAR$$

Cette transition est correcte si et seulement si x' est différent de x et de y et s'il n'est pas libre dans P'. On remarque que la condition "x' non libre dans P'" n'était pas nécessaire dans CPL. On retrouve les mêmes différences entre les diverses présentations du Π*-calcul selon que les transitions y sont marquées ou pas.*

Exemple 9.3.2 Extrusion de champ
Nous nous proposons de montrer que la transition suivante est prouvable dans SCPL :

$$\exists x(\underline{m(y,x)}.P' \otimes R) \otimes \forall z\ m(y,z).Q' \rightsquigarrow \exists x(P' \otimes R \otimes Q'[x/z])$$

Nous rappelons qu'elle est la traduction dans SCPL de la transition du Π*-calcul :*

$$(x)(\overline{y}x.P'|R)|y(z).Q' \xrightarrow{\tau} (x)(P'|R|Q'\{x/z\}).$$

Voici une preuve de la transition.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\underline{m(y,x)}.P' \otimes m(y,x).Q'[x/z] \rightsquigarrow P' \otimes Q'[x/z]}\,COM}{\underline{m(y,x)}.P' \otimes R \otimes m(y,x).Q'[x/z] \rightsquigarrow P' \otimes R \otimes Q'[x/z]}\,PAR}{\underline{m(y,x)}.P' \otimes R \otimes \forall z m(y,z).Q' \rightsquigarrow P' \otimes R \otimes Q'[x/z]}\,STRUCT}{\exists x(\underline{m(y,x)}.P' \otimes R) \otimes \forall z m(y,z).Q' \rightsquigarrow \exists x(P' \otimes R \otimes Q'[x/z])}\,RES$$

Cette transition est correcte si et seulement si x n'est pas libre dans Q'.

4 La sémantique dénotationnelle de SCPL

Les processus de SCPL étant aussi des processus de CPL, nous pouvons leur appliquer tous les résultats relatifs à la sémantique dénotationnelle de CPL tels qu'ils ont été vus dans le chapitre précédent.
Ce que nous nous proposons d'établir ici, c'est une sémantique dénotationnelle propre à SCPL. Elle va viser à exprimer une notion plus restreinte que dans CPL : *l'interaction synchrone des processus de SCPL avec le monde extérieur.*

4.1 Co-processus et interfaces synchrones

4.1.1 Co-processus synchrone d'un processus

Ainsi la notion de *co-processus synchrone* d'un processus de SCPL est une restriction de celle de co-processus d'un processus dans CPL.

Définition 9.4.1 *Un processus Q de SCPL est un* co-processus synchrone *d'un processus P de SCPL si et seulement si le séquent $P, Q \vdash 1$ est prouvable dans le système déductif de SCPL.*

L'exemple suivant illustre bien la différence entre la notion de co-processus dans CPL et celle de co-processus synchrone dans SCPL.

Exemple 9.4.1 *Nous nous proposons de modéliser l'interaction entre un ordinateur, son clavier et son écran sous une forme simplifiée : celle-ci se résumera au fait que, au fur et à mesure que les caractères sont tapés au clavier, ils sont transmis à l'écran par l'ordinateur. Le processus représentant l'ordinateur $P_{ordinateur}$ peut être défini ainsi :*

$$P_{ordinateur} \equiv \exists e\ (\underline{(e:)}.1 \otimes (e:).1 \otimes !\ (e:).\forall(c(clavier:\ c).\underline{(ecran:\ c)}.\underline{(e:)}.1))$$

Dans cette définition, le canal e va permettre de modéliser une zone d'exclusion mutuelle où l'ordinateur dès qu'il a reçu un caractère du clavier, ne peut pas en recevoir de nouveau tant qu'il n'a pas transmis ce caractère à l'écran.
Le clavier et l'écran peuvent être représentés par les composantes parallèles $P_{clavier}$ et P_{ecran} d'un co-processus. Ces deux composantes ont la forme :

$$P_{clavier} \equiv \underline{(clavier:\ c_1)}.\cdots.\underline{(clavier:\ c_n)}.1$$

$$P_{ecran} \equiv (ecran:\ c'_1).\cdots.(ecran:\ c'_n).1$$

Ce modèle doit vérifier la propriété de synchronisation essentielle que les caractères doivent être affichés à l'écran dans l'ordre où ils ont été tapés au clavier. Supposons que l'on tape au clavier un "a" suivi d'un "b". Sur l'écran doit s'afficher "ab" et non "ba". En termes de processus, cela veut dire que l'on doit avoir comme co-processus de $P_{ordinateur}$:

$$(\underline{(clavier:\ a)}.\underline{(clavier:\ b)}.1) \otimes ((ecran:\ a).(ecran:\ b).1)$$

Par contre, on ne doit pas avoir comme co-processus de $P_{ordinateur}$:

$$((\underline{clavier : a}).(\underline{clavier : b}).1) \otimes ((ecran : b).(ecran : a).1)$$

Dans CPL, malheureusement les deux sont des co-processus de $P_{ordinateur}$ car on peut y permuter deux actions de réception qui se suivent.
Par contre dans SCPL, seul le premier est un co-processus synchrone de $P_{ordinateur}$. La propriété de conservation de l'ordre d'entrée des caractères au clavier est donc bien captée dans SCPL.

4.1.2 Interface synchrone d'un processus

Nous ne considérerons ici que des interfaces constituées de processus de SCPL. La notion de *satisfaction synchrone d'une interface par un processus de SCPL* repose sur celle de co-processus synchrone.

Définition 9.4.2 *Un processus P de SCPL* satisfait de manière synchrone une interface *I si et seulement tout élément de I est un co-processus synchrone de P.*

La relation de déduction entre processus de SCPL étant dans SCPL une restriction de celle dans CPL, il en est de même de la relation de déduction et de réduction entre interfaces.

Définition 9.4.3 *Une interface I_2* se déduit de manière synchrone d'une interface *I_1 si et seulement si pour tout élément Q_1 de I_1, il existe un élément Q_2 de I_2 tel que le séquent $Q_1 \vdash Q_2$ est prouvable dans le système déductif de SCPL.*
Nous écrirons alors : $I_1 \overset{s}{\vdash} I_2$

Définition 9.4.4 *Une interface I_2 est une* réduction synchrone *d'une interface I_1 si et seulement si elle est une partie de I_1 qui s'en déduit de manière synchrone.*
Nous écrirons alors : $I_1 \overset{s}{\sqsupseteq} I_2$.

Pour illustrer cette définition, poursuivons avec l'exemple 9.4.1.

Exemple 9.4.2 *Soit I l'interface constituée par les processus suivants :*

$$((\underline{clavier : a}).(\underline{clavier : b}).1) \otimes ((ecran : a).(ecran : b).1)$$

$$((\underline{clavier : a}).(\underline{clavier : b}).1) \otimes ((ecran : b).(ecran : a).1)$$

Dans CPL, I se réduit à

$$\{(\underline{clavier : a}).(\underline{clavier : b}).1) \otimes ((ecran : a).(ecran : b).1)\}$$

Dans SCPL : I est irréductible de manière synchrone.

Les relations de déduction et de réduction synchrones entre interfaces sont compatibles avec la relation de satisfaction synchrone d'une interface par un processus.
Comme dans CPL, on peut dans SCPL relativiser l'interface synchrone d'un processus par rapport à un environnement donné.

Définition 9.4.5 *Soit $\mathcal{E}$ un environnement.*
L'interface synchrone d'un processus de SCPL *P* relative à *$\mathcal{E}$ est l'ensemble de tous les co-processus synchrones de P qui appartiennent à $\mathcal{E}$. Nous l'écrirons alors : $\mathcal{I}_{\mathcal{E}}{}^{s}(P)$.*

Le théorème 8.2.5 qui montrait le rôle clé des interfaces linéaires dans la réduction des interfaces relatives de processus dans CPL, est encore valable dans SCPL où il s'énonce ainsi :

Théorème 9.4.1 *Pour tout processus P de SCPL, il existe une interface linéaire I telle que :*
- I est une réduction synchrone dans SCPL de $\mathcal{I}_{\mathcal{P}}{}^{s}(P)$;
- les variables libres de I sont aussi libres dans P ;
- si 0 est un sous-processus de P, alors les éléments de I contiennent uniquement des messages présents dans P avec éventuellement un renommage de leurs variables liées.

Preuve 9.4.1 *Il suffit de reprendre la preuve du théorème 8.2.5 et de vérifier que toutes les déductions qui y sont effectuées peuvent se faire de manière synchrone.*□

Exemple 9.4.3 *Pour suivant avec l'exemple 9.4.1, considérons l'environnement $\mathcal{N}$ des co-processus normaux. Il est alors possible de montrer que l'interface synchrone de $P_{ordinateur}$ relative à $\mathcal{N}$ se réduit de manière synchrone à l'interface linéaire constituée de processus de la forme :*

$$\exists c_1(\underline{(clavier : c_1)}.(ecran : c_1).\cdots\exists c_n(\underline{(clavier : c_n)}.(ecran : c_n)\cdots).1$$

4.2 Comparaison synchrone de processus de SCPL

Comme dans CPL, à partir de la notion d'interface synchrone d'un processus relative à un environnement, nous pouvons définir une relation de pré-ordre, notée $\overset{\mathcal{E}}{\leq}_s$ et une congruence $\overset{\mathcal{E}}{\sim}_s$ entre processus de SCPL $\mathcal{E}$.
Par ailleurs, nous pouvons exprimer aussi directement la notion de réalisation d'un processus par un autre dans SCPL mais le théorème est moins fort que le théorème correspondant de CPL.

Théorème 9.4.2 *Soit P_1 et P_2 deux processus de SCPL.*
Si le séquent $P_1 \vdash P_2$ est prouvable dans SCPL, alors quel que soit l'environnement $\mathcal{E}$, on a $P_1 \overset{\mathcal{E}}{\geq}_s P_2$.

Preuve 9.4.2 *De la même façon que pour le théorème correspondant de CPL.*□

Pour illustrer ces relations entre processus de SCPL, reprenons quelques exemples déjà étudiés dans CPL.

Exemple 9.4.4
Nous avions vu que dans CPL le tampon infini représenté par le processus $P_1 \equiv !a\underline{b}.1$ était équivalent au tampon à une place représenté par le processus $P_2 \equiv \exists s\ (a.\underline{b}.\underline{s}.1 \otimes !(s.a.\underline{b}.\underline{s}.1) \otimes \underline{a}.b.s.1)$ relativement à l'environnement $\mathcal{N}$ des co-processus normaux. Ce n'est plus vrai dans SCPL. En effet, le processus $\underline{a}.\underline{a}.b.b.1$ est un co-processus normal synchrone de P_1 mais pas de P_2. C'est logique car il exprime le chargement dans le tampon de deux unités d'information suivi de leur libération. Les opérations n'étant pas permutables dans SCPL, il n'est pas possible de charger deux unités d'information dans un tampon à une place.

Exemple 9.4.5
Soit les processus P_1 et P_2 définis ainsi :

$$P_1 = p.\underline{q}.1 \otimes r.\underline{s}.1$$

$$P_2 = (p.\underline{q}.r.\underline{s}.1)\&(r.\underline{s}.p.\underline{q}.1)$$

Tout d'abord, nous allons étudier si l'un est réalisable de manière synchrone à partir de l'autre, c'est-à-dire si les séquents $P_1 \vdash P_2$ et $P_2 \vdash P_1$ sont prouvables dans SCPL.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{r.\underline{s}.1 \vdash r.\underline{s}.1}\,ID}{1,\ r.\underline{s}.1 \vdash r.\underline{s}.1}\,TERM_L}{\underline{q}.1,\ r.\underline{s}.1 \vdash \underline{q}.r.\underline{s}.1}\,SDt}{p.\underline{q}.1,\ r.\underline{s}.1 \vdash p.\underline{q}.r.\underline{s}.1}\,RCt \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{p.\underline{q}.1 \vdash p.\underline{q}.1}\,ID}{1,\ p.\underline{q}.1 \vdash p.\underline{q}.1}\,TERM_L}{\underline{s}.1,\ p.\underline{s}.1 \vdash \underline{q}.p.\underline{s}.1}\,SDt}{r.\underline{s}.1,\ p.\underline{q}.1 \vdash r.\underline{s}.p.\underline{q}.1}\,RCt}{r.\underline{s}.1,\ p.\underline{q}.1 \vdash (p.\underline{q}.r.\underline{s}.1)\&(r.\underline{s}.p.\underline{q}.1)}\,ALT_R}{r.\underline{s}.1 \otimes p.\underline{q}.1 \vdash (p.\underline{q}.r.\underline{s}.1)\&(r.\underline{s}.p.\underline{q}.1)}\,PAR_L$$

Donc dans SCPL, nous avons $P_1 \vdash P_2$ *mais* $P_1 \nvdash P_2$.
Maintenant, comparons les deux processus du point de vue de leur interface synchrone relative.
D'après le théorème 9.4.2, nous pouvons tout d'abord déduire de la démontrabilité de $P_1 \vdash P_2$ *dans SCPL que :* $P_2 \overset{\mathcal{P}}{\leq_s} P_1$ *Il s'agit de savoir si l'on a l'équivalence.*
Commençons par construire une réduction linéaire synchrone de $\mathcal{I}_{\mathcal{P}}{}^s(P_1)$.

$$\mathcal{I}_{\mathcal{P}}{}^s(p.\underline{q}.1) \overset{s}{\sqsupseteq} \{\underline{p}.q.1\}$$

$$\mathcal{I}_{\mathcal{P}}{}^s(r.\underline{s}.1) \overset{s}{\sqsupseteq} \{\underline{r}.s.1\}$$

Par entrelacement, nous obtenons :

$$\mathcal{I}_{\mathcal{P}}{}^s(P_1) \overset{s}{\sqsupseteq} \{\underline{p}.q.\underline{r}.s.1,\ \underline{p}.\underline{r}.q.s.1,\ \underline{p}.\underline{r}.s.q;1,\ \underline{r}.\underline{p}.q.s.1,\ \underline{r}.\underline{p}.s.q.1,\ \underline{r}.s.\underline{p}.q.1\}$$

Au contraire de CPL, dans SCPL, il n'est pas possible de réduire cette interface de manière synchrone.
Maintenant, nous allons construire une réduction linéaire synchrone de $\mathcal{I}_{\mathcal{P}}{}^s(P_2)$.

$$\mathcal{I}_{\mathcal{P}}{}^s(p.\underline{q}.r.\underline{s}.1) \overset{s}{\sqsupseteq} \{\underline{p}.q.\underline{r}.s.1\}$$

$$\mathcal{I}_{\mathcal{P}}{}^s(r.\underline{s}.1,\ p.\underline{q}.1) \overset{s}{\sqsupseteq} \{\underline{r}.s.\underline{p}.q.1\}$$

Par union, nous avons :

$$\mathcal{I}_{\mathcal{P}}{}^s(P_2) \overset{s}{\sqsupseteq} \{\underline{p}.q.\underline{r}.s.1,\ \underline{r}.s.\underline{p}.q.1\}$$

En conclusion : $P_2 \overset{\mathcal{P}}{<_s} P_1$.

5 Traduction du Π-calcul dans SCPL

Le Π-calcul [Milner, 1991; Milner *et al.*, 1992] est un formalisme très puissant pour ce qui est de ses capacités d'expression dans le domaine du parallélisme. Le fait de pouvoir traduire le Π-calcul dans SCPL illustre le pouvoir d'expression de SCPL. Parmi les diverses présentations du Π-calcul, nous avons choisi la plus complète et en même temps la plus proche de SCPL celle du Π-calcul polyadique [Milner, 1991]. Nous commencerons par traduire sa syntaxe dans celle de SCPL. Cela nous permettra de délimiter un fragment restreint SCPL_0 de SCPL correspondant au Π-calcul. Dans ce fragment, nous allons pouvoir spécialiser le système déductif de SCPL utilisé pour réduire les processus et à partir de là, nous ferons de même avec sa relation de transition. Nous pourrons ensuite établir une correspondance entre cette relation et la relation de transition du Π-calcul.

5.1 Traduction de la syntaxe

Considérons la syntaxe normalisée du Π-calcul polyadique [Milner, 1991] où les abstractions et les concrétions ont été standardisées. La syntaxe des processus peut alors être définie ainsi :

$$P_N ::= 0 \mid \alpha.\lambda\vec{x}P \mid \alpha.\nu\vec{y}[\vec{x}]P \mid P_N + P_N$$

$$P ::= P_N \mid P|P \mid !P \mid \nu x P$$

- P_N représente un processus normal et P un processus quelconque.
- $\alpha.\lambda\vec{x}P$ et $\alpha.\nu\vec{y}[\vec{x}]P$ représentent respectivement une abstraction et une concrétion normalisées ; pour la seconde, la syntaxe est restreinte par la condition : $\vec{y} \subset \vec{x}$.

- α est le nom d'une action choisi parmi les éléments d'un ensemble $\mathcal{N}$.
- $\vec{x}$ et $\vec{y}$ sont des vecteurs de noms.

Nous proposons maintenant de traduire tout processus P du Π-calcul en un processus $[[P]]$ de SCPL. Nous choisissons comme ensemble de variables, l'ensemble $\mathcal{N}$ des noms du Π-calcul et l'ensemble $\emptyset$ comme ensemble des opérateurs sur les variables.
Les messages auront la forme : $(\alpha : \vec{x})$ où α est le nom d'un canal de communication et $\vec{x}$ le contenu du message qui transite par ce canal.
L'ensemble $\mathcal{P}_s$ des processus de SCPL est ainsi fixé.
Il s'agit ensuite de définir une application $P \longrightarrow [[P]]$ de l'ensemble des processus du Π-calcul dans $\mathcal{P}$.
Nous procédons inductivement à l'aide des équations suivantes :

$$
\begin{aligned}
[[0]] &= 1 \\
[[\alpha.\lambda\vec{x}P]] &= \forall\vec{x}(\alpha : \vec{x}).[[P]] \\
[[\alpha.\nu\vec{y}[\vec{x}]P]] &= \exists\vec{y}\underline{(\overline{\alpha} : \vec{x})}.[[P]] \\
[[P_{N1} + P_{N2}]] &= \overline{[[P_{N1}]]}\&[[P_{N2}]] \\
[[P_1|P_2]] &= [[P_1]] \otimes [[P_2]] \\
[[!P]] &= ![[P]] \\
[[\nu xP]] &= \exists x[[P]]
\end{aligned}
$$

En ce qui concerne les concrétions qui sont de la forme $\alpha.\nu\vec{y}[\vec{x}]P$, nous supposons que α n'appartient pas à $\vec{y}$, ce qui est toujours possible par renommage au sein de $\vec{y}$.
Le sous-ensemble $\mathcal{P}_0$ de $\mathcal{P}_s$ constitué par les processus images de processus du Π-calcul, peut être défini directement à l'aide de la grammaire suivante :

$$P_N \ ::= \ 1 \mid \forall\vec{x}(\alpha : \vec{x}).P \mid \exists\vec{y}\underline{(\alpha : \vec{x})}.P \mid P_N\&P_N$$

$$P \ ::= \ P_N \mid P \otimes P \mid !P \mid \exists xP$$

Pour que la définition soit complète, il faut ajouter une restriction qui est que pour les processus de la forme $\exists\vec{y}\underline{(\alpha : \vec{x})}.P$, on a la condition : $\vec{y} \subset \vec{x}$.
L'application $P \longrightarrow [[P]]$ n'est pas injective car : $[[\ \nu\vec{y_1}\alpha.\nu\vec{y_2}[\vec{x}]P]] \ = \ [[\alpha.\nu\vec{y_1}\nu\vec{y_2}[\vec{x}]P]]$.
L'application va nous permettre de comparer la sémantique opérationnelle du Π-calcul avec celle de SCPL. En préalable, pour simplifier la comparaison,il est utile de spécialiser le système déductif de SCPL au fragment qui nous intéresse et qui est défini par $\mathcal{P}_0$. Etant donné notre objectif, nous ne considérerons pas le système dans sa totalité. Nous nous en tiendrons aux règles qui permettent de construire des réductions totales de processus, c'est-à-dire aux règles de réduction. De cette façon, nous allons construire un calcul $SCPL_0$ qui est une restriction de SCPL. Cette spécialisation va induire une spécialisation de la relation de transition qu'il sera ensuite plus facile de comparer avec celle du Π-calcul.

5.2 Spécialisation du système de réduction des processus de SCPL

Nous utilisons encore la méthode exposée dans la première partie de cette thèse qui consiste à s'appuyer sur la permutabilité des inférences pour normaliser les preuves.
Considérons une réduction totale dans SCPL d'un système de processus qui sont tous des éléments de $\mathcal{P}_0$. Comme nous l'avons toujours fait, nous allons procéder par étapes en déplaçant les inférences de la réduction, type par type.

1. D'après le tableau 6.2, les inférences de type GEN_L peuvent être montées au maximum, c'est-à-dire, vu la syntaxe des processus de $\mathcal{P}_0$, jusqu'aux inférences de type COM_L qui produisent leur formule active. On peut donc fusionner ces inférences en une seule et ainsi remplacer les deux règles GEN_L et COM_L par une nouvelle $COM1_L$ qui a la forme suivante :

$$\frac{P_1[\vec{z}/\vec{x}],\ P_2,\ \Gamma \vdash 1}{\forall\vec{x}(\alpha : \vec{x}).P_1, \underline{(\alpha : \vec{z})}.P_2,\ \Gamma \vdash 1}\ COM1_L$$

2. Ensuite, nous montons les inférences de type RES_L. Or, d'après le tableau 6.2, elles peuvent être bloquées éventuellement par des inférences de type $COM1_L$ qui sont en position de permutation avec elles mais pas permutables. Envisageons une telle éventualité et montrons que cela nous mène à une absurdité. Nous aurions une configuration qui se présenterait ainsi :

$$\dfrac{\dfrac{P_1[\vec{z}/\vec{x}],\ P_2,\ (\exists\vec{y}\underline{(\beta:\vec{z_1})}.P_3)[u/v],\ \Gamma \vdash 1}{\forall\vec{x}(\alpha:\vec{x}).P_1,\ \underline{(\alpha:\vec{z})}.P_2,\ (\exists\vec{y}\underline{(\beta:\vec{z_1})}.P_3)[u/v],\ \Gamma \vdash 1}\ COM1_L}{\forall\vec{x}(\alpha:\vec{x}).P_1,\ \underline{(\alpha:\vec{z})}.P_2,\ \exists v\exists\vec{y}\underline{(\beta:\vec{z_1})}.P_3,\ \Gamma \vdash 1}\ RES_L$$

L'inférence que nous voulons monter est celle de type RES_L et l'inférence qui la bloque, est celle de type $COM1_L$. Ce blocage ne peut s'expliquer que par le fait que u est libre dans $P_1[\vec{z}/\vec{x}]$ mais pas dans $\forall\vec{x}(\alpha:\vec{x}).P_1$. Donc u est libre dans $\vec{z}$ ce qui contredit le fait que u n'est pas libre dans $\underline{(\alpha:\vec{z})}.P_2$ à cause de la condition liée à la règle RES_L.
Donc ce genre de blocage n'est pas possible et nous pouvons monter les inférences de type RES_L au maximum c'est-à-dire jusqu'aux inférences de type $COM1_L$ qui introduisent leur formule active. Nous pouvons alors les fusionner avec ces dernières et remplacer les règles RES_L et $COM1_L$ par une nouvelle $COM2_L$ définie ainsi :

$$\dfrac{P_1[\vec{z}/\vec{x}],\ P_2[\vec{v}/\vec{y}],\ \Gamma \vdash 1}{\forall\vec{x}(\alpha:\vec{x}).P_1,\ \exists\vec{y}\underline{(\alpha:\vec{w})}.P_2,\ \Gamma \vdash 1}\ COM2_L$$

avec $\vec{z} = \vec{w}[\vec{v}/\vec{y}]$ et les variables de $\vec{v}$ non libres dans la conclusion

3. Nous terminons en montant les inférences de type ALT_L. D'après le tableau 6.1, ce mouvement ne rencontre aucun obstacle. Et vu la syntaxe des processus, les inférences seront arrêtées par les inférences de type $COM1_L$ qui produisent leur formule active. Nous pouvons alors les fusionner avec celles-ci et remplacer les règles ALT_L et $COM1_L$ par une nouvelle COM'_L qui a la forme suivante :

$$\dfrac{P_1[\vec{z}/\vec{x}],\ P_2[\vec{v}/\vec{y}],\ \Gamma \vdash 1}{(\forall\vec{x}(\alpha:\vec{x}).P_1)+\cdots,\ (\exists\vec{y}\underline{(\alpha:\vec{w})}.P_2)+\cdots,\ \Gamma \vdash 1}\ COM'_L$$

avec $\vec{z} = \vec{w}[\vec{v}/\vec{y}]$ et les variables de $\vec{v}$ non libres dans la conclusion

En conclusion, nous obtenons un ensemble de règles qui vont permettre de produire les réductions totales de systèmes de processus de $\mathcal{P}_0$. Il constituera le *système de réduction de* $SCPL_0$ qui est défini par la figure 9.3.

5.3 Le système de transition de $SCPL_0$

De la spécialisation du système d'inférence de SCPL, nous pouvons déduire facilement une spécialisation du système de transition de SCPL.
La relation de réduction structurelle de $SCPL_0$ s'obtient par suppression des règles inutiles dans la définition de celle de SCPL telle qu'elle est donnée par la figure 7.3. Elle se présente alors comme la figure 9.4 l'indique.
Pour ce qui concerne la règle de transition, seule la règle de communication est modifiée. D'où sa définition pour $SCPL_0$ donnée par la figure 9.5.
Il est facile de montrer que pour les processus de $SCPL_0$, la relation de transition telle qu'elle est définie ici, se confond avec celle de SCPL. C'est d'ailleurs pour cela que nous utilisons le même symbole pour les désigner.

5.4 Comparaison entre relation de transition de $SCPL_0$ et relation de transition du Π-calcul

Considérons les deux processus du Π-calcul P et Q. Il s'agit d'étudier la correspondance entre $[[P]] \rightsquigarrow [[Q]]$ et $P \rightarrow Q$. Le résultat de cette étude se traduit par le théorème suivant :

Théorème 9.5.1 *Si dans $SCPL_0$, la transition $[[P]] \rightsquigarrow [[Q]]$ est prouvable sans l'aide de l'axiome $TERMg_S$ pour produire $[[Q]]$ et si la relation de congruence du Π-calcul est étendue par la règle $\nu x\alpha.C \equiv$*

Règle de communication :

$$\frac{P_1[\vec{z}/\vec{x}],\ P_2[\vec{v}/\vec{y}],\ \Gamma \vdash 1}{(\forall\vec{x}(\alpha:\vec{x}).P_1)+\cdots,\ (\exists\vec{y}\underline{(\alpha:\vec{w})}.P_2)+\cdots,\ \Gamma \vdash 1}\ COM'_L$$

avec $\vec{z} = \vec{w}[\vec{v}/\vec{y}]$ *et les variables de* $\vec{v}$ *non libres dans la conclusion*

Règles d'opération :

Terminaison

$$\frac{\Gamma \vdash 1}{1,\ \Gamma \vdash 1}\ TERM_L \qquad \frac{}{!\Gamma \vdash 1}\ TERM_R$$

Composition parallèle

$$\frac{P_1,\ P_2,\ \Gamma \vdash 1}{P_1 \otimes P_2,\ \Gamma \vdash 1}\ PAR_L$$

Récurrence

$$\frac{!P,\ P,\ \Gamma \vdash 1}{!P,\ \Gamma \vdash 1}\ REC_L$$

Restriction

$$\frac{P[y/x],\ \Gamma \vdash 1}{\exists x P,\ \Gamma \vdash 1}\ RES_L$$

avec y non libre dans la conclusion

FIG. 9.3 - **Le système de réduction de** $SCPL_0$

$\alpha.\nu x\ C$ où x n'est pas le nom correspondant à α, alors dans le Π-calcul, la transition $P \rightarrow Q$ est prouvable.

Preuve 9.5.1 *Nous procéderons par induction sur la structure de la preuve de $[[P]] \rightsquigarrow [[Q]]$ dans $SCPL_0$. Selon le type de la dernière inférence I de la preuve, nous pouvons distinguer les cas suivants :*

1. I est un axiome de type COM'.
   P a la forme $(\alpha\lambda\vec{x}.P_1+\cdots)|(\nu\vec{y_1}\overline{\alpha}.\nu\vec{y_2}[\vec{z}]P_2+\cdots)$ et Q la forme $\nu\vec{y_1}\vec{y_2}(P_1[\vec{z}/\vec{x}] \mid P_2)$.
   Par la règle COMM du Π-calcul, nous pouvons établir :

   $$(\alpha\lambda\vec{x}.P_1+\cdots)|(\overline{\alpha}.\nu\vec{y_2}[\vec{z}]P_2+\cdots) \rightarrow \nu\vec{y_2}(P_1[\vec{z}/\vec{x}] \mid P_2).$$

   Nous pouvons supposer que les noms de $\vec{y_1}$ ne sont pas libres dans P ; sinon, nous effectuons un renommage dans P en utilisant la règle numéro 1 définissant la congruence structurelle entre processus [Milner, 1991].
   Ensuite à l'aide de la règle RES, nous pouvons inférer :

   $$\nu\vec{y_1}((\alpha\lambda\vec{x}.P_1+\cdots)|(\overline{\alpha}.\nu\vec{y_2}[\vec{z}]P_2+\cdots)) \rightarrow \nu\vec{y_1}\nu\vec{y_2}(P_1[\vec{z}/\vec{x}] \mid P_2).$$

   Enfin, la règle STRUCT nous permet d'obtenir :

   $$(\alpha\lambda\vec{x}.P_1+\cdots)|(\nu\vec{y_1}\overline{\alpha}.\nu\vec{y_2}[\vec{z}]P_2+\cdots) \rightarrow \nu\vec{y_1}\nu\vec{y_2}(P_1[\vec{z}/\vec{x}] \mid P_2),\ i.e.\ P \rightarrow Q.$$

2. I est une inférence de type PAR.
   Le résultat est immédiat car les règles PAR sont les mêmes dans le Π-calcul et dans $SCPL_0$.

$$\frac{}{\bigotimes !\Gamma \Rightarrow 1}\ TERMg_S \qquad \frac{}{P \Rightarrow P}\ ID_S$$

$$\frac{P \Rightarrow P'}{1 \otimes P \Rightarrow P'}\ TERMl_S$$

$$\frac{P \otimes !P \otimes Q \Rightarrow P'}{!P \otimes Q \Rightarrow P'}\ REC_S$$

$$\frac{P[y/x] \otimes Q \Rightarrow P'}{\exists x P \otimes Q \Rightarrow P'}\ RES_S$$

avec y non libre dans $\exists x P$, Q et P'

FIG. 9.4 - **Relation de réduction structurelle entre processus de** $SCPL_0$

$$\frac{}{(\forall \vec{x}(\alpha : \vec{x}).P_1) + \cdots,\ (\exists \vec{y}(\underline{\alpha : \vec{z}}).P_2) + \cdots \rightsquigarrow \exists \vec{y}(P_1[\vec{z}/\vec{x}] \otimes P_2)}\ COM'$$

$$\frac{P \rightsquigarrow P'}{P \otimes Q \rightsquigarrow P' \otimes Q}\ PAR$$

$$\frac{P[y/x] \otimes Q \rightsquigarrow P'[y/z] \otimes Q'}{\exists x P \otimes Q \rightsquigarrow \exists z P' \otimes Q'}\ RES$$

avec y non libre dans $\exists x P$, Q et $\exists z P'$, Q'

$$\frac{P \Rightarrow P_1 \quad P_1 \rightsquigarrow P_1' \quad P_1' \Rightarrow P'}{P \rightsquigarrow P'}\ STRUCT$$

FIG. 9.5 - **Relation de transition entre processus de** $SCPL_0$

3. I est une inférence de type RES.
   Le résultat peut être facilement obtenu par application des règles RES et $STRUCT$ du Π-calcul.

4. I est une inférence de type $STRUCT$.
   Il est facile de montrer que si $[[P_1]] \Rightarrow [[P_2]]$ est prouvable sans l'aide de la règle $TERM1_S$, alors $P_1 \equiv P_2$ si l'on étend la relation de congruence avec la règle mentionnée dans le théorème. Le résultat en découle immédiatement.

□

Le théorème réciproque est faux car la relation de congruence structurelle est plus faible que la relation de réduction structurelle de $SCPL_0$. Cette dernière ne respecte pas les 9 règles définissant la congruence structurelle du Π-calcul [Milner, 1991]; elle n'est pas symétrique et pas compatible avec toutes les opérations sur les processus.

Conclusion

En conclusion, nous avons vu dans ce chapitre que nous pouvions représenter la communication synchrone dans CPL par une forme particulière de réductions de processus et de preuves et que nous

pouvions *restreindre le système d'inférence de CPL* pour l'adapter à ces preuves. Ainsi, nous avons pu obtenir un *calcul synchrone*, SCPL.
Le système déductif de SCPL n'est *pas complet* par rapport à la logique linéaire : un séquent de SCPL prouvable en logique linéaire n'est pas toujours prouvable dans le système d'inférence de SCPL. Nous avons montré une autre voie de représentation de la communication synchrone qui conserve la complétude par rapport à la logique linéaire. Elle consiste à restreindre la syntaxe des processus mais elle est très lourde.
Par restriction de la sémantique opérationnelle et de la sémantique dénotationnelle de CPL, nous avons obtenu celles de SCPL. La notion d'équivalence entre processus de SCPL ainsi obtenue, est plus proche de la notion de bissimulation dans la mesure où toutes les deux s'appuient sur la communication synchrone mais les liens entre les deux ne sont guère plus simples que dans CPL.
A notre connaissance, les seuls autres travaux utilisant la logique linéaire pour bâtir un calcul de processus synchrone, sont ceux de [Kobayashi and Yonezawa, 1994a]. Mais comme nous l'avons montré dans la section 3, l'équivalence entre CCS et un fragment de ACL n'est pas vraie : la démontrabilité en logique linéaire d'un séquent $\vdash P$ où P est une formule représentant un processus de CCS traduit dans ACL, n'implique pas la terminaison de P dans CCS (au sens où c'est défini dans [Kobayashi and Yonezawa, 1994a]).

Chapitre 10

CPL :un cadre commun pour l'étude sémantique de programmes

Introduction

Comme tout modèle de représentation de la réalité, CPL peut être utilisé comme *outil d'analyse dans le domaine de la communication et du parallélisme.* D'autres outils existent déjà [Hoare, 1985; Hennessy, 1988; Milner, 1989] mais CPL permet peut-être de capter certains aspects de la réalité que d'autres formalismes ont du mal à saisir. Ainsi par exemple, il permet d'exprimer l'interaction d'un processus avec l'extérieur dans sa globalité comme ne peut pas le faire la notion de bissimulation. Il présente par ailleurs l'intérêt d'être un modèle logique.
Il peut aussi servir de base à la *conception de langages de programmation parallèles fondés sur CPL.* Dans le chapitre 7, nous avons eu une idée du pouvoir d'expression du langage de CPL : il permet de représenter des mécanismes de communication très divers, de définir les processus de façon récursive, de construire des programmes modulaires, des programmes génériques. Le fait que la réduction de processus peut prendre dans CPL la forme de la construction de preuves, permet naturellement d'en faire un *langage de programmation logique* dont l'interprète serait un démonstrateur de théorèmes dans un fragment particulier d'ILL.
Une autre application consiste à *utiliser CPL comme cadre sémantique commun pour étudier la correction de programmes écrits dans des langages les plus divers.* CPL offre autour de la notion d'interface, la possibilité d'étudier la correction de ces programmes par rapport à une spécification donnée, à les comparer entre eux ou à vérifier que certaines transformations de programmes préservent leur sémantique.
Dans ce chapitre, nous nous proposons de montrer comment il est possible de traduire naturellement dans CPL des programmes écrits dans des langages les plus divers pour pouvoir ensuite les comparer. Nous n'effectuerons pas le travail de comparaison lui-même car c'est un vaste chantier qui nous mènerait beaucoup trop loin. Nous nous contenterons d'en poser les prémisses. Et plutôt que d'en rester à des généralités, nous avons préféré nous servir d'un exemple, celui du tri d'une liste d'entiers.

1 Spécification du problème en Prolog et traduction dans CPL

Il s'agit d'exprimer le problème de façon relativement naturelle sans préjuger de la manière dont il va être résolu. Un langage proche de celui de la logique classique peut convenir tel que Prolog.
Afin d'avoir un programme purement logique, nous représenterons les entiers sous forme symbolique à

l'aide des constructeurs 0 et s. Nous obtenons alors un programme p_1 qui se présente ainsi :

$$
\begin{array}{l}
sort(L_1, L_2) : -permut(L_1, L_2),\ ord(L_2). \\
permut([\,], [\,]). \\
permut([X|L], L') : -permut(L, L_1),\ insert(X, L_1, L'). \\
insert(X, L, [X|L]). \\
insert(X, [Y|L], [Y|L']) : -insert(X, L, L'). \\
ord([\,]). \\
ord([X]) : -int(X). \\
ord([X, Y|L]) : -inf(X, Y),\ ord([Y|L]). \\
int(0). \\
int(s(X)) : -int(X). \\
inf(0, X) : -int(X). \\
inf(s(X), s(Y)) : -inf(X, Y).
\end{array}
$$

Ce programme n'est pas efficace du tout et il va falloir en rechercher un autre plus efficace mais équivalent à p_1. Si l'on écrit les autres programmes dans Prolog, cela facilitera la comparaison. Mais Prolog n'est peut-être pas le meilleur langage pour écrire des programmes efficaces. Si nous choisissons un langage très éloigné de Prolog, la comparaison risque d'être plus délicate.
L'idée alors est de traduire p_1 dans un processus P_1 de CPL. Ensuite, pour vérifier la correction de tout autre programme p écrit dans un langage quelconque, nous allons chercher à le traduire par un processus P de CPL que nous pourrons comparer à P_1.
Voyons comment représenter le programme p_1 par un processus P_1 (d'autres possibilités existent [Zlatuška, 1993]). Comme p_1 est constitué de 12 clauses, P_1 va être obtenu par composition parallèle de 12 processus, chacun représentant une clause. Les clauses étant utilisées un nombre indéterminé de fois au cours de la satisfaction d'une requête, ces 12 processus seront tous récurrents.
La tête d'une clause sera traduite par l'envoi d'un message alors que les formules atomiques constituant le corps seront représentés par des réceptions de messages précédant l'envoi.
Ainsi la règle $sort(L_1, L_2) : -permut(L_1, L_2),\ ord(L_2)$ se traduira par le processus :

$$! \forall l_1\ l_2\ permut(l_1, l_2).ord(l_2).\underline{sort(l_1, l_2)}.1.$$

Et le fait $permut([\,], [\,])$ se traduira par le processus :

$$\underline{permut([\,], [\,])}.1.$$

Il existe aussi une traduction duale qui consiste à représenter les têtes de clauses par des envois de messages suivis de réceptions correspondant aux corps de clauses.
Voyons maintenant comment se traduit le mécanisme d'exécution de Prolog dans CPL. Les buts g soumis au programme p_1 seront représentés par des co-processus G placés en parallèle avec P_1. L'exécution de la requête $< p_1, g >$ se traduira alors par la réduction totale de $P_1 \otimes G$ c'est-à-dire la construction d'une preuve de $P_1 \otimes G \vdash 1$. La preuve ainsi construite fournira la réponse à la requête. Prenons un exemple. Soit le but $sort([s(0),\ 0,\ s(s(0))],\ X)$ soumis à p_1. Il est alors traduit dans CPL par le co-processus $\forall x\ sort([s(0),\ 0,\ s(s(0))],\ x).1$. La réponse à la requête sera fournie par une preuve du séquent $P_1 \otimes (\forall x\ sort([s(0),\ 0,\ s(s(0))],\ x).1) \vdash 1$. Dans cette preuve, nous retrouverons en particulier la valeur de x, $[0,\ s(0),\ s(s(0))]$, qui est la réponse de l'interprète Prolog.

2 Un programme Prolog plus efficace

Nous nous proposons maintenant tout en restant dans le cadre de Prolog d'écrire un programme p_2 plus efficace que p_1 mais qui permet de réaliser la même spécification. Nous avons choisi de traduire le

tri rapide à l'aide de p_2 d'oú le programme suivant :

$$
\begin{array}{l}
sort(L_1, L_2) : -quicksort(L_1, L_2). \\
quicksort([\,], [\,]). \\
quicksort([X|L], L') : -partition(X, L, L_1, L_2), \\
\qquad\qquad\qquad\qquad quicksort(L_1, L_1'), \\
\qquad\qquad\qquad\qquad quicksort(L_2, L_2'), \\
\qquad\qquad\qquad\qquad append(L_1', [X|L_2'], L'). \\
partition(\_, [\,], [\,], [\,]). \\
partition(X, [Y|L], L_1, L_2) : -inf(X, Y, R),\ cont-partition(R, X, Y, L, L_1, L_2). \\
append([\,], L, L). \\
append([X|L_1], L_2, [X|L]) : -append(L_1, L_2, L). \\
inf(0, X, true) : -int(X). \\
inf(s(X), 0, false) : -int(X). \\
inf(s(X), s(Y), R) : -inf(X, Y, R). \\
cont-partition(true, X, Y, L, L_1, [Y|L_2]) : -partition(X, L, L_1, L_2). \\
cont-partition(false, X, Y, L, [Y|L_1], L_2) : -partition(X, L, L_1, L_2). \\
int(0). \\
int(s(X)) : -int(X).
\end{array}
$$

Comme p_1, nous pouvons traduire p_2 par un processus P_2 de CPL ce qui va permettre de comparer indirectement les deux programmes Prolog. Comme ils sont écrits dans le même langage, nous aurions pu les comparer directement mais le passage par l'intermédiaire de CPL ne complique pas les choses. Il va falloir s'assurer s'ils vérifient la même spécification. Dans CPL, cela revient à comparer les interfaces de P_1 et de P_2. Si l'on considère les interfaces absolues, il est évident qu'elles ne sont pas équivalentes : par exemple le programme p_2 permet de concaténer deux listes, ce que ne peut pas faire p_1.
Nous allons donc comparer les interfaces de P_1 et de P_2 relativement à un environnement qui correspond au problème que nous cherchons à résoudre. Ici, l'environnement choisi sera l'ensemble $\mathcal{TRI}$ des co-processus de la forme $sort(t_1, t_2).1$ où t_1 et t_2 sont des termes quelconques d'une algèbre bâtie en utilisant au minimum les constructeurs d'entiers 0 et s.
Il faut ensuite montrer que $P_1 \overset{\mathcal{TRI}}{\sim} P_2$. La tâche se ramène donc à une démonstration de théorèmes en logique linéaire intuitionniste. Cette démonstration n'est pas forcément simple ; c'est pourquoi nous ne la ferons pas ici mais ce qu'il faut retenir c'est que CPL fournit un cadre adapté pour étudier la correction de programmes, les programmes et les spécifications y étant représentés dans un même formalisme logique. Et comme nous allons le voir maintenant, cela ne se limite pas à des programmes Prolog.

3 Un programme Pascal

Nous savons que Prolog n'est pas forcément le langage le mieux adapté à l'écriture de programmes efficaces. Donc il est souvent nécessaire d'avoir recours à un autre langage, un langage de programmation impérative par exemple. Nous avons choisi ici Pascal parce qu'il est très connu mais nous aurions très bien pu prendre le langage C.
Nous avons choisi un autre principe algorithmique consistant à ordonner les éléments d'une liste par permutation successive de tous les couples de voisins mal ordonnés.
Nous obtenons alors le programme décrit par la figure 10.1 Nous allons montrer comment représenter ce programme p_3 par un processus P_3 de CPL. Pour cela, nous allons traduire chaque instruction i de p_3 par un processus P_i de CPL en allant des plus élémentaires vers les plus complexes.
Un trait caractéristique des programmes Pascal est la séquentialité des instructions exprimée par l'opérateur " ;". Pour l'exprimer dans CPL, nous indiquerons qu'une instruction i a fini d'être exécutée par l'envoi sur le canal fin du message i, ce qui permettra de déclencher l'instruction suivante.
Il serait un peu fastidieux d'exposer ici la traduction complète du programme p_3. Nous nous contenterons d'en traduire la partie la plus significative : l'instruction *while*. Commençons par ses composantes élémentaires.

Ainsi, l'instruction i_1 $termin := not(modif)$ peut être représentée dans CPL par le processus P_{i_1} suivant :

$$\begin{array}{l}
\underline{(eval\ in :\ not(modif))}. \\
\forall v\ t\ w\ (eval\ out :\ not(modif)\ (v,t)). \\
\quad (termin :\ (w,t)). \\
\quad \underline{(termin :\ (v,t)}. \\
\quad \underline{(fin :\ i_1)}. \\
\quad 1
\end{array}$$

Commentons cette définition. Elle suppose tout d'abord l'existence d'un processus récurrent P_{eval} qui évalue des expressions arithmétiques, booléennes ou autres : sur son canal d'entrée *eval in*, il reçoit l'expression à évaluer et sur son canal de sortie *eval out*, il retourne un couple formé de la valeur de l'expression et de son type.
On commence par envoyer l'expression à évaluer au processus P_{eval}. La réponse v est retournée ensuite avec son type t. La variable *termin* du programme est représentée par un canal du même nom où la valeur stockée est w. Celle-ci est remplacée par v et la fin de l'exécution de l'instruction i_1 est indiquée par l'envoi du message i_1 sur le canal *fin*.
De la même manière, les instructions i_2 et i_3 qui suivent immédiatement i_1, sont représentées par les processus P_{i_2} et P_{i_3}. Alors, i_4 qui est définie comme **begin** i_1; i_2; i_3 **end**, est représentée par le processus P_{i_4} défini ainsi :

$$P_{i_1} \otimes (fin :\ i_1).P_{i_2} \otimes (fin :\ i_2).P_{i_3} \otimes (fin :\ i_3).\underline{(fin :\ i_4)}.1$$

L' instruction i_6 qui constitue le corps de l'instruction "while", est de type "if-then-else" et définie ainsi : **if** $\Gamma.sv = nil$ **then** i_4 **else** i_5. Elle peut donc être traduite par le processus P_{i_6} de CPL suivant :

$$\begin{array}{l}
\underline{(eval :\ \Gamma.sv = nil)}. \\
\&(eval :\ \Gamma.sv = nil\ (true, bool)). \\
\quad \otimes P_{i_4} \\
\quad \otimes (fin :\ i_4). \\
\quad \underline{(fin :\ i_6)}. \\
\quad 1 \\
\&(eval :\ \Gamma.sv = nil\ (false, bool)). \\
\quad \otimes P_{i_5} \\
\quad \otimes (fin :\ i_5). \\
\quad \underline{(fin :\ i_6)}. \\
\quad 1
\end{array}$$

Enfin, l'instruction i_7 qui est définie comme **while** $not(termin)$ **do** i_6, peut être représentée par le processus :

$$\begin{array}{l}
\underline{(debut :\ i_7)}. \\
!\,(debut :\ i_7). \\
\quad \underline{(eval\ in :\ not(termin))}. \\
\quad \&\ (eval\ out :\ not(termin)\ (true, bool)). \\
\quad\quad \otimes P_{i_6} \\
\quad\quad \otimes (fin :\ i_6). \\
\quad\quad \underline{(debut :\ i_7)}. \\
\quad\quad 1 \\
\quad \&\ (eval\ out :\ not(termin)\ (false, bool)). \\
\quad\quad \underline{(fin :\ i_7)}. \\
\quad\quad 1
\end{array}$$

Nous pouvons traduire de cette façon tout le programme p_3 en un processus P_3 de CPL. L'exécution du programme p_3 exige l'entrée d'une liste d'entiers l_0 et va se terminer par la sortie d'une liste l. Dans CPL, ceci peut être modélisé par un co-processus fonctionnel $\underline{(sort\ in :\ l_0)}.\forall l\ (sort\ out :\ l_0\ l).1$ Ensuite, il s'agit de savoir si p_3 est correct, ce qui revient à comparer l'interface de P_3 avec celle de P_1 choisi comme

étalon.
Mais il y a un problème dans la mesure où la première est fonctionnelle et où la deuxième est une interface Prolog. Il faut donc au préalable standardiser les interfaces. Cela peut au moyen d'un processus auxiliaire qui va jouer le rôle d'un adaptateur : il va permettre de transformer une interface Prolog en une interface fonctionnelle. Il peut être défini ainsi :

$$P_{adapt} \equiv \forall l_1 l_2\ \underline{sort(l_1, l_2)}.\underline{(sort\ in:\ l_1)}.(sort\ out:\ l_2).1$$

Alors, il va falloir comparer l'interface de $P_3 \otimes P_{adapt}$ avec celle de P_1 relativement à l'environnement $\mathcal{TRI}$ et montrer qu'elles sont identiques.

4 L'utilisation de CPL comme langage de programmation parallèle

Le défaut d'un langage comme Pascal est qu'il impose une séquentialisation des instructions qui n'a pas toujours de signification logique et qui nuit à la modularité des programmes. L'utilisation de CPL comme langage de programmation montre qu'il peut capter la logique interne d'un programme : la dépendance entre actions sera exprimée par la communication et l'indépendance par le parallélisme.
Nous avons choisi de programmer le tri rapide dans CPL. Le programme général sera représenté par un processus P_4 qui va être la composition parallèle de processus récurrents, chacun étant spécialisé dans l'accomplissement d'une tâche particulière. Dans notre cas, nous aurons :

$$P_4 = !P_{sort} \otimes !P_{quicksort} \otimes !P_{partition} \otimes !P_{append} \otimes !P_{inf} \otimes !P_{int}$$

Définissons ensuite chacun des composants de P_4.
Le processus principal P_{sort} peut s'écrire ainsi :

$$\begin{array}{l}
P_{sort} = \forall\ l\ (\text{sort in}: l). \\
\qquad \underline{(\text{quicksort in}: l)}. \\
\qquad \forall\ l'(\text{quicksort out}: l\ l'). \\
\qquad\quad \underline{(\text{sort out}: l\ l')}. \\
\qquad\quad 1
\end{array}$$

P_{sort} fait appel à $P_{quicksort}$ qui est défini ainsi :

$$\begin{array}{l}
P_{quicksort} = \&(\text{quicksort in}: [\ ]). \\
\qquad \underline{(\text{quicksort out}: [\ ]\ [\ ])}. \\
\qquad 1 \\
\qquad \&\forall\ x\ l(\text{quicksort in}: [x|l]). \\
\qquad\quad \underline{(\text{partition in}: (x, l))}. \\
\qquad\quad \forall\ l_1\ l_2(\text{partition out}: (x, l)\ (l_1, l_2)). \\
\qquad\qquad \otimes\underline{(\text{quicksort in}: l_1)}. \\
\qquad\qquad\quad 1 \\
\qquad\qquad \otimes\underline{(\text{quicksort in}: l_2)}. \\
\qquad\qquad\quad 1 \\
\qquad\qquad \otimes\forall\ l'_1\ l'_2(\text{quicksort out}: l_1\ l'_1). \\
\qquad\qquad\qquad (\text{quicksort out}: l_2\ l'_2). \\
\qquad\qquad\qquad \underline{(\text{append in}: (l'_1, [x|l'_2]))}. \\
\qquad\qquad\qquad \forall\ l'(\text{append out}: (l'_1, [x|l'_2])\ l'). \\
\qquad\qquad\qquad\quad \underline{(\text{quicksort out}: [x|l]\ l')}. \\
\qquad\qquad\qquad\quad 1
\end{array}$$

Le processus $P_{partition}$ qui découpe une liste l en deux sous-listes l_1 et l_2 selon la valeur discriminante x est défini ainsi :

$$P_{partition} = \&\ \forall\ x(\text{partition in}: (x, [\ ])).$$

$$\underline{(\text{partition out}: (x, [\,]) \ ([\,],[\,]))}.$$
$$1$$
$$\& \ \forall x \ y \ l(\text{partition in}: (x, [y|l])).$$
$$\otimes\underline{(\text{inf in}: (x, y))}.$$
$$1$$
$$\otimes\underline{(\text{partition in}: (x, l))}.$$
$$1$$
$$\otimes\forall l_1 \ l_2(\text{partition out}: (x, l) \ (l_1,l_2)).$$
$$\&(\text{inf out}: (x, y) \ \text{true}).$$
$$\underline{(\text{partition out}: (x, [y|l]) \ (l_1,[y|l_2]))}.$$
$$1$$
$$\&(\text{inf out}: (x, y) \ \text{false}).$$
$$\underline{(\text{partition out}: (x, [y|l]) \ ([y|l_1],l_2))}.$$
$$1$$

Le processus P_{append} qui concatène deux listes l_1 et l_2 en une seule l peut s'écrire ainsi :

$$P_{append} = \&\forall \ l(\text{append in}: ([\,],l)).$$
$$\underline{(\text{append out}: ([\,],l) \ l)}.$$
$$1$$
$$\&\forall \ x \ l_1 \ l_2 \ (\text{append in}: ([x|l_1],l_2)).$$
$$\underline{(\text{append in}: (l_1, \ l_2))}.$$
$$\forall \ l \ (\text{append out}: (l_1,l_2) \ l).$$
$$\underline{(\text{append out}: ([x|l_1],l_2) \ [x|l])}.$$
$$1$$

Maintenant nous passons à la définition du processus P_{inf} qui compare deux entiers.

$$P_{inf} = \& \ \forall \ x(\text{inf in}: (0, x)).$$
$$\underline{(\text{int in}: x)}.$$
$$(\text{int}: \text{out x}).$$
$$\underline{(\text{inf out}: (0, x) \ \text{true})}.$$
$$1$$
$$\& \ \forall \ x \ (\text{inf in}: (s(x), 0)).$$
$$\underline{(\text{int in}: x)}.$$
$$(\text{int}: \text{out } x).$$
$$\underline{(\text{inf out}: (s(x), 0) \ \text{false})}.$$
$$1$$
$$\& \ \forall \ x \ y \ (\text{inf in}: (s(x), s(y))).$$
$$\underline{(\text{inf in}: (x, y))}.$$
$$\forall \ r \ (\text{inf out}: (x, y) \ r).$$
$$\underline{(\text{inf out}: (s(x), s(y)) \ r)}.$$
$$1$$

Nous terminons par la définition de P_{int} qui teste si un terme est un entier.

$$P_{int} = \& \ (\text{int in}: 0).$$
$$\underline{(\text{int out}: 0)}.$$
$$1$$
$$\& \ \forall \ x(\text{int in}: s(x)).$$
$$\underline{(\text{int in}: x)}.$$
$$(\text{int}: \text{out } x).$$
$$\underline{(\text{int out}: s(x))}.$$
$$1$$

Pour vérifier la correction de P_4, il faut lui adjoindre le processus P_{adapt} pour le transformer en un processus qui ait une interface Prolog au lieu d'une interface fonctionnelle. Il s'agira ensuite de comparer les interfaces de $P_4 \otimes P_{adapt}$ et de P_1 relativement à l'environnement $\mathcal{TRI}$.

Conclusion

Ce qu'on peut retenir de ce chapitre, c'est que CPL peut fournir un cadre commun pour étudier la correction de programmes écrits dans des styles les plus éloignés. Nous avons vu qu'on pouvait sans peine traduire dans CPL des programmes impératifs comme des programmes logiques. Nous aurions tout aussi bien le faire pour des programmes orientés objets. Les programmes fonctionnels semblent poser davantage de problèmes. Il semble difficile de se passer du second ordre pour les représenter. En effet, lorsque l'on réduit $(\lambda x.E_1)E_2$ on peut considérer $\lambda x.E_1$ comme un processus qui reçoit comme message un autre processus E_2 ce qui implique que x puisse être instanciée par un processus. C'est pourquoi, nous avons laissés de côté pour l'instant les langages fonctionnels.
Dans le cadre de CPL, on peut représenter un programme par un processus et une spécification par une interface. Ainsi les notions de programme et de spécification sont traduites dans un même cadre logique où les concepts de déduction, de réduction et de relativisation d'une interface, de réalisation d'un processus par un autre peuvent être des outils utiles pour l'étude de la correction des programmes et des transformations de programmes.

```
program sort
   type  intlist = ^intlistr ;
         intlistr = record
                    v : int ;
                    sv :^intlist ;
                    end ;

   var   termin, modif: bool ;
         x : int ;
         l0, l :intlist ;

   procedure input(var l1 : intlist) ;
         ⋮

   procedure output(l2 : intlist) ;
         ⋮

   begin
         input(l0) ;
         termin := false ;
         modif := false ;
         l := l0 ;
         while not(termin) do
                    if l^.sv= nil then
                      begin
                          termin := not(modif) ;
                          modif := false ;
                          l := l0 ;
                      end
                    else
                      begin
                          x := l^.v ;
                          if x >l^.sv^. v then
                          begin
                            modif := true ;
                            l^.v := l^.sv^.v ;
                            l^.sv^.v := x ;
                          end ;
                          l := l^.sv ;
                      end ;
         output(l) ;
   end ;
```

FIG. 10.1 - Programme Pascal de tri d'une liste d'entiers

Conclusion

Dans la première partie de cette thèse, nous avons montré la pertinence qu'il y avait à se saisir de la notion de permutabilité d'inférences comme point de départ d'une étude de la construction de preuves en logique linéaire. Dans le cadre du calcul des séquents linéaire, nous avons montré que cette notion constituait le fondement de la normalisation de preuves et de l'élaboration de stratégies de construction. Nous avons pu ainsi mettre à jour une dualité entre principes de construction des preuves en chaînage avant et principes de construction des preuves en chaînage arrière. Nous avons mis en évidence les deux facteurs essentiels qui déterminent normalisation des preuves et stratégies de recherche : le fragment logique dans lequel les preuves se situent et le sens de leur construction qui est envisagé. Nous avons pu ainsi déboucher sur une méthode qui permet de normaliser et d'élaborer des stratégies de construction de preuves dans n'importe quel fragment de la logique linéaire. Une application systématique de cette méthode dans CLL pour la construction de preuves en chaînage avant, nous a permis d'élaborer des stratégies qu'il sera intéressant de comparer précisément avec les stratégies basées sur la résolution.
Ces travaux peuvent trouver une première application dans la conception de démonstrateurs de théorèmes automatiques ou semi-automatiques en logique linéaire. Une autre perspective est d'étudier dans quelle mesure ces résultats ont une portée qui dépasse le cadre de la logique linéaire et peuvent être étendus à d'autre logiques exprimées à l'aide d'un calcul de séquents.
Dans la deuxième partie de cette thèse, nous avons proposé un modèle du parallélisme sous forme d'un calcul de processus, CPL, reposant sur la construction de preuves en logique linéaire. L'intérêt d'un tel modèle est qu'il amalgame la théorie du parallélisme avec la programmation logique. L'approche basée sur la notion d'interface autour de laquelle s'articule la sémantique dénotationnelle, demande à être comparée plus précisément avec les approches classiques fondées sur la notion de bissimulation. Ce sera un de nos axes de travail futurs.
Une première application de CPL est de fournir la base d'un langage de programmation logique parallèle. Mais il peut être utilisé aussi en tant que cadre logique d'analyse dans le domaine du parallélisme. Enfin, il peut offrir un cadre commun et des outils théoriques pour une étude sémantique des programmes et des transformations de programmes écrits dans des langages les plus divers.

Bibliographie

[Abramsky and Jagadeesan, 1992] S. Abramsky and R. Jagadeesan. Games and full completeness for multiplicative linear logic. Technical Report DOC 92/24, Imperial College, London, September,25 1992.

[Abrusci, 1990] V. M. Abrusci. Noncommutative intuitionistic linear propositional logic. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 36 :297–318, 1990.

[Abrusci, 1991] V. M. Abrusci. Phase semantics and sequent calculus for pure noncommutative classical linear propositional logic. *Journal of Symbolic Logic*, 56(4) :1403–1451, December 1991.

[Agha, 1986] G. Agha. *Actors : a Model of Concurrent Computation in Distributed Systems.* MIT Press, 1986.

[Alexiev, 1993] V. Alexiev. Applications of linear logic to computation : an overview. Technical Report TR93-18, Department of Computing Science, 615 GSB University of Alberta, Edmonton, Alberta T6G 2H1 Canada, December 1993.

[Allwein and Dunn, 1992] G. Allwein and J. M. Dunn. Kripke models for linear logic. *Journal of Symbolic Logic*, 58(2) :514–545, 1992.

[Andreoli and Pareschi, 1990a] J.-M. Andreoli and R. Pareschi. Linear objects : Logical processes with built-in inheritance. In *proceedings of 7th international Conference of Logic Programming, Jerusalem,june 1990*, 1990.

[Andreoli and Pareschi, 1990b] J.-M. Andreoli and R. Pareschi. Logic programming with sequent systems ; a linear logic approach. In Schroeder-Heister P., editor, *Proceedings of an International Workshop on Extensions of Logic Programming, Tübingen, December 8-10, 1989*, volume 475 of *Lecture Notes in Computer Science*, pages 1–30. Springer Verlag, 1990.

[Andreoli and Pareschi, 1991a] J.-M. Andreoli and R. Pareschi. Communication as fair distribution of knowledge. In *Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'91), Phoenix, U.S.A., nov. 1991*, volume 26(11), pages 212–219. ACM SIGPLAN Notices, 1991.

[Andreoli and Pareschi, 1991b] J.-M. Andreoli and R. Pareschi. Linear objects : Logical processes with built-in inheritance. *New Generation Computing*, 9(3-4) :445–473, 1991. earlier version [Andreoli and Pareschi, 1990a].

[Andreoli *et al.*, 1991] J.-M. Andreoli, R. Pareschi, and M. Bourgois. Dynamic programming as multiagent programming. In M. Tokoro, O. Nierstrasz, and P. Wegner, editors, *Object-based Concurrent Computing, ECOOP'91, Geneva, Switzerland, july 1991*, volume 612 of *Lecture Notes in Computer Science*, pages 163–176. Springer Verlag, 1991.

[Andreoli, 1992] J. M. Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3) :297–347, 1992.

[Asperti, 1987] A. Asperti. A Logic for Concurrency. Manuscript, November 1987.

[Back, 1979] R. J. R. Back. On correct refinement of programs. *Journal of Computer and System Sciences*, 23(1) :49–68, 1979.

[Back, 1980] R. J. R. Back. Correctness preserving programs refinements : proof theory and applications. Mathematical Centre Tracts 131, Mathematical Centre, Amsterdam, 1980.

[Back, 1988] R. J. R. Back. A calculus of refinements for program derivations. *Artificial Intelligence*, 25 :593–624, 1988.

[Banatre, 1991] J.-P. Banatre. *La programmation parallèle : outils, méthodes et éléments de mise en œuvre*. Eyrolles, Paris, 1991.

[Barr, 1991] M. Barr. $\star$-Autonomous categories and linear logic. *Mathemathical Structures in Computer Science*, 1(2) :159–178, 1991.

[Bellin and Ketonen, 1992] G. Bellin and J. Ketonen. A decision procedure revisited : Notes on direct logic, linear logic and its implementation. *Theoretical Computer Science*, 95 :115–142, 1992.

[Bellin, 1991] G. Bellin. Proof nets for multiplicative and additive linear logic. Technical Report LFCS-91-169, University of Edimburg, May 1991.

[Berry and Boudol, 1990] G. Berry and G. Boudol. The chemical abstract machine. In *17th ACM Symposium on Principles of Programming Languages (POPL'90), San Francisco, USA, jan. 1990*, pages 81–94. ACM, 1990.

[Blass, 1992] A. Blass. A game semantics for linear logic. *Annals of Pure and Applied Logic*, 56 :183–220, 1992. Volume dedicated to the memory of John Myhill.

[Brower, 1975] L.E.J. Brower. *Collected Works*, volume 1. Nort Holland, Amsterdam, 1975.

[Chandy and Misra, 1988] K. M. Chandy and J. Misra. *Parallel Program Design A Foundation*. Addison-Wesley Publishing Company, 1988. ISBN 0-201-05866-9.

[Chirimar and Lipton, 1991] J. Chirimar and J. Lipton. Provability in TBLL : a decision procedure. In *Int. Workshop CSL'91, Bern, October 1991*, volume 626 of *Lecture Notes in Computer Science*, pages 53–65, 1991.

[Constable, 1991] R.L. Constable. Type theory as a foundation of computer science. In *International Conference TACS '91, Sendai, Japan, September 1991*, volume 526 of *Lecture Notes in Computer Science*, pages 226–243, 1991.

[Coquand and Huet, 1988] T. Coquand and G. Huet. The Calculus of Constructions. *Information and Computation*, 76 :95–120, 1988.

[Coquand *et al.*, 1994] T. Coquand, B. Nordström, J. Smith, and B. von Sydow. Type Theory and Programming. Report 81, Programming Methodology Group, Chalmers University of Technology, June 1994.

[de Paiva, 1989] V. de Paiva. A Dialectica-like model of linear logic. In D. H. Pitt, D. E. Rydeheard, P. Dybjer, A. M. Pitts, and A. Poigné, editors, *Category Theory and Computer Science, Manchester, september 1989*, volume 389 of *Lecture Notes in Computer Science*, pages 313–340. Springer Verlag, 1989.

[Dunn, 1986] J. Dunn. Relevence Logic and Entailment. In D. Gabbay and F. Gunther, editors, *Handbook of Philosophical Logic*. 1986.

[Engberg and Winskel, 1994] U. Engberg and G. Winskel. Linear logic on petri nets. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *A decade of concurrency - Reflections and Perspectives, REX School/Symposium, Noordwijkerhout, The Netherlands, june 1993*, volume 803 of *Lecture Notes in Computer Science*, pages 176–229. Springer Verlag, 1994.

[Gallier, 1986] J.-H. Gallier. *Logic for Computer Science.* Harper and Row, New-York, 1986.

[Gallier, 1992a] J. Gallier. Constructive logics. part i : A tutorial on proof systems and typed λ-calculi. Technical Report?, Computer Science Department, University of Pennsylvania, 1992. Revised version of Digital PRL Tech. Report No. 8 .

[Gallier, 1992b] J. Gallier. Constructive logics. part ii : Linear logic and proof nets. Technical Report?, Computer Science Department, University of Pennsylvania, 1992. Revised version of Digital PRL Tech. Report No. 9.

[Galmiche and Perrier, 1994a] D. Galmiche and G. Perrier. Foundations of proof search strategies design in linear logic. In A. Nerode and Yu.V. Matiyasevich, editors, *Proceedings of Logical Foundations of Computer Science, St Petersburg, Russia, July 1994*, volume 813 of *Lecture Notes in Computer Science*, pages 101–113. Springer Verlag, 1994.

[Galmiche and Perrier, 1994b] D. Galmiche and G. Perrier. On proof normalisation in linear logic. *Theoretical Computer Science*, 135(1) :67–110, December 1994.

[Galmiche, 1992] D. Galmiche. Program development in constructive type theory. *Theoretical Computer Science*, 94(2) :237–259, 1992.

[Gelernter, 1985] D. Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1) :80–112, 1985.

[Gentzen, 1935] G. Gentzen. Untersuchungen über das logische scliessen. *Mathematische Zeitschrift*, 39 :176–210, 405–431, 1935. Translation in Szabo M.E., editor, The collected papers of Gerhard Gentzen, Chapter 3, 68-131, North-Holland, Amsterdam, 1969.

[Girard, 1986] J.Y. Girard. The system F of variables types, fifteen years later. *Theoretical Computer Science*, 45 :159–192, 1986.

[Girard, 1987] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1) :102, 1987.

[Girard, 1989] J.-Y. Girard. Towards a geometry of interaction. In J. W. Gray and A. Scedrov, editors, *Proceedings of the AMS-IMS-SIAM Joint Summer Research Conference on Categories in Computer Science and Logic, Boulder, Colorado, june 14-20, 1987*, volume 92 of *Contemporary Mathematics*, pages 69–108. American Mathematical Society, 1989.

[Gonthier *et al.*, 1992] G. Gonthier, M. Abadi, and J.-J. Lévy. Linear logic without boxes. In *Proceedings of the Seventh Annual Symposium on Logic in Computer Science, Santa Cruz, June 22-25,1992*, pages 223–234. IEEE Computer Society Press, 1992.

[Gunter and V., 1989] C. Gunter and Gehlot V. A Proof-Theoritical Operational Semantics for True Concurrency. Preliminary Report, 1989.

[Harland and Pym, 1990] J. Harland and D. Pym. The uniform proof-theoretic foundation of linear logic programming. Technical Report ECS-LFCS-90-124, University of Edinburgh, November 1990.

[Harland and Pym, 1991] J. Harland and D. Pym. The uniform proof-theoretic foundation of linear logic programming (extended abstract). In V. Saraswat and K. Ueda, editors, *Intl. Symposium on Logic Programming (SLP'91)*, pages 304–318, 1991. For the full paper see [Harland and Pym, 1990].

[Harland and Pym, 1992] J. Harland and D. Pym. On resolution in fragments of classical linear logic. In *LPAR'92, International Conference on Logic Programming and Automated Reasoning, St. Petersburg, Russia, july 1992*, volume 624 of *Lecture Notes in Computer Science*, pages 30–41, 1992.

[Hennessy and de Nicola, 1983] M. Hennessy and R. de Nicola. Testing Equivalence for Processes. *Theoretical Computer Science*, 34 :83–133, 1983.

[Hennessy, 1988] M. Hennessy. *Algebraic Theory of Processes.* MIT Press Series in the Foundations of Computing. The MIT Press, 1988.

[Heyting, 1966] A. Heyting. *Intuitionism.* Springer Verlag, Berlin, second revised edition, 1966.

[Hoare, 1985] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall International Series in Computer Science. Prentice-Hall, 1985.

[Hodas and Miller, 1991] J. S. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science, Amsterdam, July 15-18, 1991*, pages 32–42. IEEE Computer Society Press, 1991.

[Hodas and Miller, 1994] J. S. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110(2):327–365, May 1994. Earlier version [Hodas and Miller, 1991].

[Howard, 1980] W. Howard. The formulas-as-types notion of construction. In *To H.B. Curry : Essays on Combinatory Logic, Lambda-Calculus and Formalism*, pages 479–490. Academic Press, 1980.

[Kanovich, 1991] M.I. Kanovich. The multiplicative fragment of linear logic is NP- complete. ITLI Prepublication Series X-91-13, University of Amsterdam, 1991.

[Ketonen and Weyhrauch, 1984] J. Ketonen and R. Weyhrauch. A decidable fragment of predicate calculus. *Theoretical Computer Science*, 32:297–307, 1984.

[Kleene, 1952] S.-C. Kleene. Permutability of inferences in Gentzen's calculi LK and LJ. *Memoirs of the AMS*, 10:1–26, 1952.

[Kleene, 1968] S.-C. Kleene. *Mathematical Logic.* John Wiley & Sons, 1968.

[Kobayashi and Yonezawa, 1992] N. Kobayashi and A Yonezawa. Asynchronous communication model based on linear logic. In *proceedings of Joint Intl. Conf. and Symp. on Logic Programming(JICSLP'92), Workshop on Linear Logic and Logic Programming, Washington, november 92*, 1992.

[Kobayashi and Yonezawa, 1993] N. Kobayashi and A. Yonezawa. Logical, testing, and observation equivalence for processes in a linear logic programming. Technical Report 93-4, Department of Information Science, the University of Tokyo, 1993. presented at Linear Logic Workshop, Cornell University.

[Kobayashi and Yonezawa, 1994a] N. Kobayashi and A. Yonezawa. Asynchronous communication model based on linear logic. *Formal Aspects of Computing*, 3:1–37, 1994. Earlier version [Kobayashi and Yonezawa, 1992].

[Kobayashi and Yonezawa, 1994b] N. Kobayashi and A. Yonezawa. Typed higher-order concurrent linear logic programming. Technical Report 94-12, Department of Information Science, the University of Tokyo, July 1994.

[Kobayashi and Yonezawa, 1994c] N. Kobayashi and N. Yonezawa. Type-theoritic foundations for concurrent object-oriented programming. In *ACM SIGPLAN Conference on Object-Oriented Programming Systems,Languages and Applications (OOPSLA'94).* ACM, 1994.

[Lafont and Streicher, 1991] Y. Lafont and T. Streicher. Games semantics for linear logic. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science, Amsterdam, July 15-18, 1991*, pages 43–50. IEEE Computer Society Press, 1991.

[Lambek, 1958] J. Lambek. The mathematics of sentence structure. *Amer. Math. Monthly*, 65:154–169, 1958.

[Lifschitz, 1989] V. Lifschitz. What is the inverse method? *Journal of Automated Reasoning*, 5(1):1–23, March 1989.

[Lincoln and Saraswat, 1992] P. Lincoln and V. Saraswat. Higher-order, linear, concurrent constraint programming, July 1992. Draft paper.

[Lincoln and Scedrov, 1992] P. Lincoln and A. Scedrov. First Order Linear Logic Without Modalities is NEXPTIME-Hard. Manuscript, August 1992. Accepted for publication in Theoretical Computer Science.

[Lincoln and Shankar, 1994] P. Lincoln and N. Shankar. First-order Linear Logic and other cut-free sequent calculi. In *Proceedings of the Ninth Annual Symposium on Logic in Computer Science, Paris,France, jul. 1994*. IEEE Computer Society Press, 1994. Also appears in SRI Technical Report CSL-93-11.

[Lincoln and Winkler, 1992] P. Lincoln and T. Winkler. Constant-only multiplicative linear logic is NP-complete. Manuscript, August 1992.

[Lincoln *et al.*, 1990] P. Lincoln, J. Mitchell, A. Scedrov, and N. Shankar. Decision problems for propositional linear logic. In *Proceedings of 31th Annual Symposium on Foundations of Computer Science, Volume II, St. Louis, Missouri, October 22-24, 1990*, pages 662–671. IEEE Computer Society Press, 1990.

[Lincoln, 1992] P. Lincoln. *Computational aspects of linear logic.* PhD thesis, Stanford University, 1992.

[LINDA, 1990] Scientific Computing Associates inc, 246 Church Street, Suite 307 New Haven, CT 06510 USA. *Original LINDA C-Linda Reference manual*, 1990.

[Martí-Oliet and Meseguer, 1989] N. Martí-Oliet and J. Meseguer. From Petri nets to linear logic. In *proceedings of Category Theory and Computer Science, Manchester, sept. 1989*, volume 389 of *Lecture Notes in Computer Science*, pages 313–340. Springer Verlag, 1989.

[Martin-Löf, 1982] P. Martin-Löf. Constructive Mathematics and Computer Programming. In *6th Congress for Logic, Methodology and Philosophy of Science*, volume IV, pages 153–175. North-Holland, Amsterdam, 1982.

[Masseron *et al.*, 1993] M. Masseron, C. Tollu, and J. Vauzeilles. Generating plans in linear logic : I. Actions as proofs. *Theoretical Computer Science*, 113 :349–370, june 1993.

[Masseron, 1993] M. Masseron. Generating plans in linear logic : II. A geometry of conjunctive actions. *Theoretical Computer Science*, 113 :371–375, june 1993.

[Meyer, 1988] B. Meyer. *Object-oriented Software Construction.* International series in Computer Science. Prentice Hall, 1988.

[Miller and Nadathur, 1988] D. Miller and G. Nadathur. An overview of λprolog. In K. Bowen and R. Kowalski, editors, *Fifth International Conference and Symposium on Logic Programming.* MIT Press, 1988.

[Miller *et al.*, 1987] D. Miller, G. Nadathur, and A. Scedrov. Hereditary harrop formulas and uniform proof systems. In *Proceedings of the Annual Symposium on Logic in Computer Science, Ithaca, New York, jun. 1987*, pages 98–105. IEEE Computer Society Press, 1987.

[Miller *et al.*, 1991] D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51 :125–157, 1991.

[Miller, 1992] D. Miller. The π-calculus as a theory in linear logic : Preliminary results. Technical Report MS-CIS-92-48, Computer Science Department, University of Pennsylvania, June 1992.

[Milner *et al.*, 1992] R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes, I. and II. *Information and Computation*, 100(1) :1–77, September 1992.

[Milner, 1980] R. Milner. A Calculus of Communicating Systems. In *LNCS*, volume 92, 1980.

[Milner, 1983] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25 :267–310, 1983.

[Milner, 1989] R. Milner. *Communication and Concurrency.* International series in Computer Science. Prentice Hall, 1989.

[Milner, 1991] R. Milner. The Polyadic π-Calculus: a Tutorial. Technical Report ECS-LFCS-91-180, University of Edingburg, October 1991.

[Milner, 1992] R. Milner. Functions as processes. *Mathemathical Structures in Computer Science*, 2(1) :119–141, June 1992.

[Mints, 1990] G. Mints. Gentzen-type systems and resolution rules. Part I. Propositional logic. In P. Martin-Löf and G. Mints, editors, *COLOG-88, Tallin, Estonia*, volume 417 of *Lecture Notes in Computer Science*, pages 198–231. Springer Verlag, 1990.

[Mints, 1993] G. Mints. Resolution Calculus for the First-Order Linear-Logic. *Journal of Logic Language and Information*, 2 :59–83, 1993.

[Nordström *et al.*, 1990] B. Nordström, K. Petersson, and J. Smith. *Programming in Martin-Löf's Type Theory, An Introduction*, volume 7 of *Monographs on Computer Science.* Oxford Press, 1990.

[Park, 1980] D.M.R. Park. Concurrency and Automota on Infinite Sequences. In *LNCS*, volume 104. Springer Verlag, 1980.

[Plotkin, 1981] G.D. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI-FN-19, Computer Science Dept,Århus University, Denmark, 1981.

[Pym and Harland, 1994] D. Pym and J. Harland. A uniform proof-theoretic investigation of linear logic programming. *Journal of Logic and Computation*, 4(2) :175–207, 1994.

[Robinson, 1965] J.A. Robinson. A machine oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12 :23–41, 1965.

[Seely, 1989] R. A. G. Seely. Linear logic, $*$-autonomous categories and cofree coalgebras. In J. W.. Gray and A. Scedrov, editors, *Proceedings of the AMS-IMS-SIAM Joint Summer Research Conference on Categories in Computer Science and Logic, Boulder, Colorado, june 14-20, 1987*, volume 92 of *Contemporary Mathematics*, pages 371–382. American Mathematical Society, 1989.

[Shankar, 1992] N. Shankar. Proof Search in the Intuitionistic Sequent Calculus. In D. Kapur, editor, *11th International Conference on Automated Deduction*, pages 522–536, Saratoga Springs,NY,USA, June 1992. Springer Verlag.

[Tammet, 1993] T. Tammet. Proof search strategies in linear logic. Programming Methodology Group Report 70, Chalmers University Group, University of Göteborg, 1993.

[Troelstra, 1992] A. S. Troelstra. *Lectures on Linear Logic.* CSLI Lecture Notes 29. Center for the Study of Language and Information, Stanford, 1992.

[Urquhart, 1984] A. Urquhart. The undecidability of entailment and relevant implication. *Journal of Symbolic Logic*, 49 :1059–1073, 1984.

[Volpe, 1994] P. Volpe. Concurrent Logic Programming as Uniform Linear Proofs. In G. Levi and M.R. Artalejo, editors, *Algebraic and Logic Programming , ALP94', Madrid, Spain, sep. 1994*, volume 850 of *Lecture Notes in Computer Science*, pages 133–149. Springer Verlag, 1994.

[Voronkov, 1992] A. Voronkov. Theorem proving in non-standard logics based on the inverse method. In D. Kapur, editor, *11th International Conference on Automated Deduction , Saratoga Springs NY, USA, jun. 1992*, volume 607 of *Lecture Notes in Computer Science*, pages 648–662. Springer Verlag, 1992.

[Wallen, 1990] L.A. Wallen. *Automated proof search in non-classical logics.* Series in Artificial Intelligence. MIT press, 1990.

[Yetter, 1990] D. N. Yetter. Quantales and (noncommutative) linear logic. *Journal of Symbolic Logic*, 55(1) :41–64, March 1990.

[Zlatuška, 1993] J. Zlatuška. Commited-choice Concurrent Logic Programming in Linear Logic. In *K. G. Colloquium*, volume 713 of *Lecture Notes in Computer Science*, pages 337–348. Springer Verlag, 1993.

Annexe A

Permutabilité d'inférences dans CLL

Théorème 2.3.1

Soit t_1 et t_2 deux types du tableau ci-dessous.

1. *la case (t_1, t_2) du tableau est vide si et seulement si toute inférence de type t_1 en position de permutation avec une inférence de type t_2 dans une déduction, est permutable avec celle-ci ;*
2. *la case (t_1, t_2) du tableau contient np ou $-$ si et seulement si il existe une inférence de type t_1 en position de permutation avec une inférence de type t_2 dans une déduction mais qui ne soit pas permutable avec cette dernière (le tiret $-$ indique que la non permutabilité est toute relative et qu'elle peut être contournée comme nous le verrons dans la prochaine section) ;*
3. *la case (t_1, t_2) du tableau contient une croix $\times$ si et seulement si il n'existe pas de déduction où une inférence de type t_1 soit en position de permutation avec une inférence de type t_2.*

$t_2 \backslash t_1$	$\otimes$	$\wp$	$\perp$	$\&$	$\oplus$	$?$	$w?$	$c?$	$!$	$\forall$	$\exists$	cut
$\otimes$									np			
$\wp$	np								np			np
$\perp$									np			
$\&$	np	$-$	$-$	$-$	np	np	np	$-$	np	$-$	np	np
$\oplus$									np			
$?$												
$w?$												
$c?$	np											np
$!$	$\times$	$\times$	$\times$	$\times$	$\times$	np			$\times$	$\times$	$\times$	np
$\forall$									np		np	np
$\exists$									np			
cut									np			

Preuve

Considérons une preuve de CLL et deux inférences de cette preuve I_1 et I_2 telles que I_1 soit en position de permutation avec I_2. Soit t_1 et t_2 les types respectifs de I_1 et I_2. Nous allons passer en revue toutes les valeurs possibles de t_1 et t_2 en recherchant celles pour lesquelles I_1 et I_2 sont toujours permutables. Ce qui complique cette tâche, ce sont les règles d'inférence sensibles au contexte : $\&$, $!$ et $\forall$.
Nous allons faire une analyse de cas qui tienne compte de cela.

1. **t_2 est égal à $\&$.**
   Pour que I_1 soit permutable avec I_2, il faut que I_2 soit précédée par une autre inférence de type t_1

qui puisse dans la permutation, fusionner avec I_1. Il est facile de produire pour chaque valeur de t_1 un contre-exemple qui montre que la permutabilité n'est pas toujours vraie. Nous n'irons pas plus loin ici, réservant pour la prochaine section l'explication sur la différence entre cases marquées "*np*" et cases marquées "−" dans le tableau.

2. **t_2 est égal à !.**
Le contexte initial de I_2 est de la forme $?\Delta$ et comme I_1 est en postion de permutation avec elle, ce contexte inclut la partie principale de I_1. Il y a alors deux possibilités :
   - soit cette partie principale est vide ce qui signifie que t_1 est égal à *cut* ;
   - soit I_1 a une formule principale de la forme $?F$ ce qui signifie que t_1 est égal à ?, w? ou c?. Les autres cas ne sont pas possibles, ce qui justifie l'existence des croix dans le tableau accompagnant l'énoncé du théorème.

Examinons maintenant ces quatre possibilités.

(a) t_1 **est égal à** *cut*
I_1 et I_2 forment alors la déduction suivante :

$$\dfrac{\dfrac{\vdash F,\ G,\ ?\Delta_1 \quad \vdash F^{\perp},\ ?\Delta_2}{\vdash G,\ ?\Delta_1,\ ?\Delta_2}\,\mathrm{cut}}{\vdash\ !G,\ ?\Delta_1,\ ?\Delta_2}\,!$$

En essayant de permuter les deux inférences, nous obtenons comme résultat :

$$\dfrac{\dfrac{\vdash F,\ G,\ ?\Delta_1}{\vdash F,\ !G,\ ?\Delta_1}\,! \quad \vdash F^{\perp},\ ?\Delta_2}{\vdash\ !G,\ ?\Delta_1,\ ?\Delta_2}\,\mathrm{cut}$$

Malheureusement, cette déduction n'est valide que si F est de la forme $?F'$, ce qui n'est pas toujours le cas donc I_1 n'est pas toujours permutable avec I_2.

(b) t_1 **est égal à ?.**
Les deux inférences constituent donc la déduction :

$$\dfrac{\dfrac{\vdash F,\ G,\ ?\Delta}{\vdash ?F,\ G,\ ?\Delta}\,?}{\vdash ?F,\ !G,\ ?\Delta}\,!$$

En permutant, nous devrions obtenir la déduction :

$$\dfrac{\dfrac{\vdash F,\ G,\ ?\Delta}{\vdash F,\ !G,\ ?\Delta}\,!}{\vdash ?F,\ !G,\ ?\Delta}\,?$$

Mais comme dans le cas précédent, la validité de la déduction dépend de la forme de F qui doit être de la forme?F'.

(c) t_1 **est égal à w?.**
Les deux inférences constituent donc la déduction :

$$\dfrac{\dfrac{\vdash G,\ ?\Delta}{\vdash ?F,\ G,\ ?\Delta}\,\mathrm{w?}}{\vdash ?F,\ !G,\ ?\Delta}\,!$$

En permutant, nous devrions obtenir la déduction :

$$\dfrac{\dfrac{\vdash G,\ ?\Delta}{\vdash\ !G,\ ?\Delta}\,!}{\vdash ?F,\ !G,\ ?\Delta}\,\mathrm{w}?$$

Cette déduction est correcte et elle a même hypothèse et conclusion que la précédente donc I_1 est permutable avec I_2.

(d) t_1 **est égal à** `c?`.
I_1 est I_2 forment alors la déduction :

$$\dfrac{\dfrac{\vdash ?F,\ ?F,\ G,\ ?\Delta}{\vdash ?F,\ G,\ ?\Delta}\,\mathrm{c}?}{\vdash ?F,\ !G,\ ?\Delta}\,!$$

En permutant, nous devrions obtenir la déduction :

$$\dfrac{\dfrac{\vdash ?F,\ ?F,\ G,\ ?\Delta}{\vdash ?F,\ ?F,\ !G,\ ?\Delta}\,!}{\vdash ?F,\ !G,\ ?\Delta}\,\mathrm{c}?$$

Cette déduction est correcte et elle a même hypothèse et conclusion que la précédente donc I_1 est permutable avec I_2.

3. t_1 **est égal à** ! **est** t_2 **est différent de** ! **et de** &.
Quel que soit le nombre de prémisses de I_2, les deux inférences constituent une déduction qui a la forme suivante :

$$\dfrac{\dfrac{\vdash F,\ ?\Delta_a,\ ?\Delta}{\vdash\ !F,\ ?\Delta_a,\ ?\Delta}\,! \quad \vdots}{\vdash\ !F,\ \Delta_p,\ ?\Delta,\ \Delta'}\,t_2$$

$?\Delta_a$ est la partie active du prémisse de I_2 confondu avec la conclusion de I_1 et Δ_p est la partie principale de I_2.
Si l'on veut que la permutation soit possible, cela implique que Δ_p soit vide ou constitué par une formule de la forme?F pour que la condition sur le contexte de la règle ! soit respectée. Cela veut dire que t_2 doit être égal à *cut*, ?, *w*? ou *c*?.
Dans les trois derniers cas, il est évident qu'il y aura permutablité. Voyons précisément ce qui se passe lorsque t_2 est égal à *cut*. I_1 et I_2 forment alors la déduction :

$$\dfrac{\dfrac{\vdash F,\ ?G,\ ?\Delta}{\vdash\ !F,\ ?G,\ ?\Delta}\,! \quad \vdash\ !(G^{\perp}),\ \Delta'}{\vdash\ !F,\ ?\Delta,\ \Delta'}\,\mathrm{cut}$$

Mais comme Δ' n'est pas toujours de la forme $?\Delta''$, I_1 n'est donc pas toujours permutable avec I_2.

4. t_2 **est égal à** $\forall$ **et** t_1 **est différent de** !.
Comme I_1 est en position de permutation avec I_2, la partie principale de I_1 est donc incluse dans le contexte de I_2 donc la variable y sur laquelle porte la quantification effectuée par I_2 n'est pas libre dans cette partie principale.

Si I_1 n'est pas de type *cut*, $\forall$ ou $\exists$, y ne sera pas libre non plus dans les parties actives de ses prémisses. Donc ,si I_1 résulte de l'application d'une règle insensible au contexte, elle sera permutable avec I_2.
Il reste donc à traiter 4 cas : *cut*, &, $\forall$ et $\exists$.

(a) t_1 **est égal à** *cut*.
I_1 et I_2 forment alors une déduction de la forme :

$$\dfrac{\dfrac{\vdash F,\ G[y/x],\ \Delta_1 \quad \vdash F^{\perp},\ \Delta_2}{\vdash G[y/x],\ \Delta_1,\ \Delta_2}\,\mathrm{cut}}{\vdash \forall x\ G,\ \Delta_1,\ \Delta_2}\,\forall$$

Par permutation, on devrait obtenir la déduction :

$$\dfrac{\dfrac{\vdash F,\ G[y/x],\ \Delta_1}{\vdash F,\ \forall x G,\ \Delta_1}\,\forall \qquad \vdash F^{\perp},\ \Delta_2}{\vdash \forall x G,\ \Delta_1,\ \Delta_2}\,\mathrm{cut}$$

Or, cette déduction n'est correcte que si y n'est pas libre dans F, ce qui est loin d'être toujours le cas. Donc I_1 n'est pas toujours permutable avec I_2.

(b) t_1 **est égal à** &.
Les deux inférences constituent alors la déduction :

$$\dfrac{\dfrac{\vdash F_1,\ G[y/x],\ \Delta \quad \vdash F_2,\ G[y/x],\ \Delta}{\vdash F_1 \& F_2,\ G[y/x],\ \Delta}\,\&}{\vdash F_1 \& F_2,\ \forall x G,\ \Delta}\,\forall$$

Par une permutation, on obtient la déduction :

$$\dfrac{\dfrac{\vdash F_1,\ G[y/x],\ \Delta}{\vdash F_1,\ \forall x G,\ \Delta}\,\forall \qquad \dfrac{\vdash F_2,\ G[y/x],\ \Delta}{\vdash F_2,\ \forall x G,\ \Delta}\,\forall}{\vdash F_1 \& F_2,\ \forall x G,\ \Delta}\,\&$$

La déduction étant correcte, les inférences sont permutables.

(c) t_1 **est égal à** $\forall$.
I_1 et I_2 forment alors la déduction :

$$\dfrac{\dfrac{\vdash F[u/x],\ G[v/y],\ \Delta}{\vdash \forall x F,\ G[v/y],\ \Delta}\,\forall}{\vdash \forall x F,\ \forall y G,\ \Delta}\,\forall$$

Le résultat de la permutation des deux inférences doit être la déduction :

$$\dfrac{\dfrac{\vdash F[u/x],\ G[v/y],\ \Delta}{\vdash F[u/x],\ \forall y G,\ \Delta}\,\forall}{\vdash \forall x\ F,\ \forall y G,\ \Delta}\,\forall$$

Vérifions si les deux inférences I_2' et I_1' (se suivant dans cet ordre dans la déduction ci-dessus) sont correctes, c'est-à-dire si elles satisfont la condition relative à la règle $\forall$.
Comme I_1 vérifie cette condition, u n'est pas libre dans $\forall x\ F,\ G[v/y],\ \Delta$ donc elle ne l'est pas plus dans $\forall x\ F,\ \forall y\ G,\ \Delta$ ce qui assure la correction de I_1'.

I_2 vérifiant la condition relative à la règle $\forall$, v n'est pas libre dans $\forall x F$, $\forall y G$, Δ. La seule possiblité pour v d'être libre dans $F[u/x]$, $\forall y G$, Δ c'est d'être identique à u. Dans ce cas, G[v/y] est égal à G[u/y]. Or, u n'est pas libre dans cette formule ce qui implique que y n'est pas libre dans G. On peut donc, pour assurer la correction de I_2' choisir à la place de v n'importe quelle variable qui n'est pas libre dans $F[u/x]$, $\forall y\ G$, Δ.
On a donc montré que I_1 est permutable avec I_2.

(d) t_1 **est égal à** $\exists$.
I_1 et I_2 forment alors la déduction :

$$\dfrac{\dfrac{\vdash\ F[u/x],\ G[v/y],\ \Delta}{\vdash\ \exists x F,\ G[v/y],\ \Delta}\ \exists}{\vdash\ \exists x F,\ \forall y G,\ \Delta}\ \forall$$

Le résultat de la permutation des deux inférences devrait alors se présenter ainsi :

$$\dfrac{\dfrac{\vdash\ F[u/x],\ G[v/y],\ \Delta}{\vdash\ F[u/x],\ \forall y G,\ \Delta}\ \forall}{\vdash\ \exists x F,\ \forall y G,\ \Delta}\ \exists$$

Mais pour que cette déduction soit correcte, il faudrait que v ne soit pas libre dans $F[u/x]$, $\forall y G$, Δ ; si c'est vrai pour $\forall y G$, Δ, c'est loin d'être le cas pour F[u/x]. Donc I_1 n'est pas toujours permutable avec I_2.

5. t_1 **est égal à** & **et** t_2 **correspond à une règle qui est insensible au contexte.**
Selon le nombre de prémisses de I_2, on a deux configurations possibles :

(a) I_2 **a un seul prémisse.**
I_1 et I_2 forment alors une déduction de la forme :

$$\dfrac{\dfrac{\vdash\ F_1,\ \Delta_a,\ \Delta \quad \vdash\ F_2,\ \Delta_a,\ \Delta}{\vdash\ F_1 \& F_2,\ \Delta_a,\ \Delta}\ \&}{\vdash\ F_1 \& F_2,\ \Delta_p,\ \Delta}\ t_2$$

Dans cette déduction, Δ_a et Δ_p représentent les parties active et principale de I_2.
Par permutation, on doit obtenir la déduction :

$$\dfrac{\dfrac{\vdash\ F_1,\ \Delta_a,\ \Delta}{\vdash\ F_1,\ \Delta_p,\ \Delta}\ t_2 \quad \dfrac{\vdash\ F_2,\ \Delta_a,\ \Delta}{\vdash\ F_2,\ \Delta_p,\ \Delta}\ t_2}{\vdash\ F_1 \& F_2,\ \Delta_p,\ \Delta}\ \&$$

Comme I_2 ne dépend pas du contexte, cette déduction est correcte donc I_1 est toujours permutable avec I_2.

(b) I_2 **a deux prémisses.**
I_2 est alors une inférence de type $\otimes$ ou cut et elle forme avec I_1 une déduction de la forme :

$$\dfrac{\dfrac{\vdash\ F_1,\ \Delta_a^1,\ \Delta_1 \quad \vdash\ F_2,\ \Delta_a^1,\ \Delta_1}{\vdash\ F_1 \& F_2,\ \Delta_a^1,\ \Delta_1}\ \& \qquad \vdash\ \Delta_a^2,\ \Delta_2}{\vdash\ F_1 \& F_2,\ \Delta_p,\ \Delta_1,\ \Delta_2}\ t_2$$

Dans cette déduction, Δ_a^1 et Δ_a^2 représentent les parties actives gauche et droite de I_2 et Δ_p sa partie principale.
Par permutation, nous devons obtenir le résultat suivant :

$$\dfrac{\dfrac{\vdash F_1,\ \Delta_a^1,\ \Delta_1 \quad \vdash \Delta_a^2,\ \Delta_2}{\vdash F_1,\ \Delta_p,\ \Delta_1,\ \Delta_2}\,t_2 \qquad \dfrac{\vdash F_2,\ \Delta_a^1,\ \Delta_1 \quad \vdash \Delta_a^2,\ \Delta_2}{\vdash F_2,\ \Delta_p,\ \Delta_1,\ \Delta_2}\,t_2}{\vdash F_1 \& F_2,\ \Delta_p,\ \Delta_1,\ \Delta_2}\,\&$$

Cette déduction est correcte donc I_1 est toujours permutable avec I_2.

6. **t_1 est égal à $\forall$ et t_2 ne correspond pas à une règle sensible au contexte.**
Selon le nombre de prémisses de I_2, on distingue deux cas :

(a) I_2 **a un seul prémisse.**
L'objet de la permutation a alors la forme :

$$\dfrac{\dfrac{\vdash F[y/x],\ \Delta_a,\ \Delta}{\vdash \forall x F,\ \Delta_a,\ \Delta}\,\forall}{\vdash \forall x F,\ \Delta_p,\ \Delta}\,t_2$$

Le résultat de la permutation doit se présenter ainsi :

$$\dfrac{\dfrac{\vdash F[y/x],\ \Delta_a,\ \Delta}{\vdash F[y/x],\ \Delta_p,\ \Delta}\,t_2}{\vdash \forall x F,\ \Delta_p,\ \Delta}\,\forall$$

Mais pour que cette déduction soit correcte, il faut que y ne soit pas libre dans $\forall x F$, Δ_p, Δ. Comme I_1 est correcte, on sait déja que y n'est pas libre dans $\forall x F$, Δ_a, Δ. Mais de Δ_a à Δ_p, peuvent être apparues de nouvelles variables libres si t_2 est égal à $\oplus$ ou w?. Si parmi celles-ci, on trouve y, on conserve une preuve correcte en renommant y. Donc dans tous les cas, la permutabilité est assurée.

(b) I_2 **a deux prémisses.**
I_2 est alors de type $\otimes$ ou cut et elle forme avec I_1 une déduction qui se présente ainsi :

$$\dfrac{\dfrac{\vdash F[y/x],\ \Delta_a^1,\ \Delta_1}{\vdash \forall x F,\ \Delta_a^1,\ \Delta_1}\,\forall \qquad \vdash \Delta_a^2,\ \Delta_2}{\vdash \forall x F,\ \Delta_p,\ \Delta_1,\ \Delta_2}\,t_2$$

Le résultat de la permutation devrait avoir la forme suivante :

$$\dfrac{\dfrac{\vdash F[y/x],\ \Delta_a^1,\ \Delta_1 \quad \vdash \Delta_a^2,\ \Delta_2}{\vdash F[y/x],\ \Delta_p,\ \Delta_1,\ \Delta_2}\,t_2}{\vdash \forall x F,\ \Delta_p,\ \Delta_1,\ \Delta_2}\,\forall$$

Pour que cette déduction soit correcte, il faut que y ne soit pas libre dans $\forall x F$, Δ_p, Δ_1, Δ_2. Comme I_1 est correcte, on sait déja que y n'est pas libre dans $\forall x F$, Δ_a^1, Δ_1. Malheureusement, y peut être apparue comme variable libre dans Δ_2 et même dans Δ_p quand I_2 est de type $\otimes$. On conserve une preuve correcte par un renommage de y.

7. **t_1 et t_2 correspondent tous deux à des règles insensibles au contexte.**
Selon le nombre de prémisses des deux inférences, on peut distinguer les cas suivants :

(a) **I_1 et I_2 ont seulement un prémisse.**
L'objet de la permutation a alors la forme suivante :

$$\dfrac{\dfrac{\vdash \Delta_a,\ \Delta'_a,\ \Delta}{\vdash \Delta_p,\ \Delta'_a,\ \Delta}\ t_1}{\vdash \Delta_p,\ \Delta'_p,\ \Delta}\ t_2$$

Δ_a et Δ'_a constituent les parties actives respectives de I_1 et I_2 et Δ_p et Δ'_p leurs parties principales.
Le résultat de la permutation doit avoir la forme suivante :

$$\dfrac{\dfrac{\vdash \Delta_a,\ \Delta'_a,\ \Delta}{\vdash \Delta_a,\ \Delta'_p,\ \Delta}\ t_2}{\vdash \Delta_p,\ \Delta'_p,\ \Delta}\ t_1$$

Les inférences ne dépendant pas du contexte, cette déduction est correcte donc les inférences sont permutables.

(b) **I_1 a un prémisse et I_2 en a deux.**
I_1 et I_2 constituent alors une déduction de la forme (les notations ont une signification analogue à celle du cas précédent) :

$$\dfrac{\dfrac{\vdash \Delta_a,\ \Delta'^1_a,\ \Delta_1}{\vdash \Delta_a,\ \Delta'^1_a,\ \Delta_1}\ t_1 \qquad \vdash \Delta'^2_a,\ \Delta_2}{\vdash \Delta_p,\ \Delta'_p,\ \Delta_1,\ \Delta_2}\ t_2$$

Par permutation, on obtient alors la déduction :

$$\dfrac{\dfrac{\vdash \Delta_a,\ \Delta'^1_a,\ \Delta_1 \quad \vdash \Delta'^2_a,\ \Delta_2}{\vdash \Delta_a,\ \Delta'_p,\ \Delta_1,\ \Delta_2}\ t_2}{\vdash \Delta_p,\ \Delta'_p,\ \Delta_1,\ \Delta_2}\ t_1$$

Les inférences ne dépendant pas du contexte, la permutation est correcte.

(c) **I_1 et I_2 ont deux prémisses.**
Les deux inférences sont donc de type $\otimes$ ou cut et forment une déduction qui se présente ainsi :

$$\dfrac{\dfrac{\vdash \Delta^1_a,\ \Delta'^1_a,\ \Delta_1 \quad \vdash \Delta^2_a,\ \Delta_2}{\vdash \Delta_p,\ \Delta'^1_a,\ \Delta_1,\ \Delta_2}\ t_1 \qquad \vdash \Delta'^2_a,\ \Delta'}{\vdash \Delta_p,\ \Delta'_p,\ \Delta_1,\ \Delta_2,\ \Delta'}\ t_2$$

Le résultat de la permutation se présentera alors ainsi :

$$\dfrac{\dfrac{\vdash \Delta^1_a,\ \Delta'^1_a,\ \Delta_1 \quad \vdash \Delta'^2_a,\ \Delta'}{\vdash \Delta^1_a,\ \Delta'_p,\ \Delta_1,\ \Delta'}\ t_2 \qquad \vdash \Delta^2_a,\ \Delta_2}{\vdash \Delta_p,\ \Delta'_p,\ \Delta_1,\ \Delta_2,\ \Delta'}\ t_2$$

Comme I_1 et I_2 ne sont pas dépendantes du contexte, la permutation est correcte.

(d) I_1 **a deux prémisses et** I_2 **un seul.**
L'inférence I_1 est donc de type $\otimes$ ou cut et forme avec I_2 une déduction qui se présente ainsi :

$$\dfrac{\dfrac{\vdash \Delta_a^1,\ \Delta_a'^1,\ \Delta_1 \quad \vdash \Delta_a^2,\ \Delta_a'^2,\ \Delta_2}{\vdash \Delta_p,\ \Delta_a'^1,\ \Delta_a'^2,\ \Delta_1,\ \Delta_2}\,t_1}{\vdash \Delta_p,\ \Delta_p',\ \Delta_1,\ \Delta_2}\,t_2$$

Dans la déduction ci-dessus, la réunion de $\Delta_a'^1$ et $\Delta_a'^2$ constitue la partie active de I_2. Il est facile de voir que la permutation est possible si $\Delta_a'^1$ ou $\Delta_a'^2$ est vide. C'est en général le cas car I_2 a au plus une formule active sauf si elle est de type $\wp$ ou c?. Dans ce cas, elle a deux formules actives qui peuvent être réparties entre $\Delta_a'^1$ et $\Delta_a'^2$, ce qui rend la permutation impossible.
Si I_2 est d'un autre type que $\wp$ ou c?, sa partie active qui a au plus un élément, se confond avec $\Delta_a'^1$ ou $\Delta_a'^2$. Supposons par exemple que ce soit $\Delta_a'^1$ et que $\Delta_a'^2$ soit vide. La permutation est alors possible et le résultat est le suivant :

$$\dfrac{\dfrac{\vdash \Delta_a^1,\ \Delta_a'^1,\ \Delta_1}{\vdash \Delta_a^1,\ \Delta_p',\ \Delta_1}\,t_2 \qquad \vdash \Delta_a^2,\ \Delta_2}{\vdash \Delta_p,\ \Delta_p',\ \Delta_1,\ \Delta_2}\,t_1$$

Annexe B

Elimination des coupures dans CLL

Théorème 2.5.2

Soit $\mathcal{P}_1$ une preuve de CLL qui comporte au moins une coupure. Alors il existe une preuve $\mathcal{P}_2$ qui a la même conclusion que $\mathcal{P}_1$ et une complexité des coupures strictement plus faible.

Preuve

Considérons I une coupure quelconque de $\mathcal{P}_1$ qui ne soit précédée par aucune autre. Démontrer le théorème revient à prouver qu'en montant I dans la preuve, on peut diminuer la complexité des coupures qui résultent de cette montée.
En utilisant le théorème 2.3.3, on peut remonter I dans $\mathcal{P}_1$ jusqu'aux inférences produisant ses formules actives. La coupure peut avoir été dupliquée au cours du mouvement mais quoi qu'il en soit, sa complexité n'a pas été modifiée.
Chacune des coupures I' résultant de cette montée, se trouve bloquée par les deux inférences I_1' et I_2' qui la précèdent. Mais on peut lever cet obstacle tout en diminuant la complexité de I'. Nous devons pour cela, considérer différents cas selon le type de I_1' et I_2' (le nombre de cas à considérer se trouve réduit de moitié à cause de la symétrie des coupures).

1. **$\top$ et un type quelconque.**
   Supposons par exemple que ce soit I_1' qui ait le type $\top$. Nous avons alors deux configurations possibles selon que la formule $\top$ de I_1' est active ou non dans I'.

   (a) **$\top$ est active dans I'.**
   Les trois inférences forment alors la configuration suivante :

$$\dfrac{\dfrac{}{\vdash \top, \Delta}\,\top \qquad \dfrac{}{\vdash 0, \top, \Delta'}\,\top}{\vdash \top, \Delta, \Delta'}\,\text{cut}$$

   Celle-ci peut être remplacée par le seul axiome : $\dfrac{}{\vdash \top, \Delta, \Delta'}\,\top$
   La complexité de la coupure a bien entendu diminué puisqu'il n'y a plus de coupure.

   (b) **$\top$ n'est pas active dans I'.**
   Les trois inférences avec celles qui les précèdent, forment alors la configuration suivante :

$$\dfrac{\dfrac{}{\vdash \top, F, \Delta}\,\top \qquad \begin{array}{c}\vdots\\ \vdash F^{\perp}, \top, \Delta'\end{array}}{\vdash \top, \Delta, \Delta'}\,\text{cut}$$

Celle-ci peut aussi être remplacée par le seul axiome : $\dfrac{}{\vdash \top,\ \Delta,\ \Delta'}\ \top$
De la même façon, la complexité de la coupure a été annulée.

2. **1 et $\perp$.**
On a alors la configuration suivante :

$$\dfrac{\dfrac{}{\vdash 1}\ 1 \qquad \dfrac{\left.\begin{array}{c}\vdots\\ \vdash \Delta\end{array}\right\}\mathcal{P}}{\vdash \perp,\ \Delta}\ \perp}{\vdash \Delta}\ \text{cut}$$

Elle peut être remplacée par la preuve $\mathcal{P}$ qui, étant donné nos hypothèses, est sans coupures donc là encore, la complexité de la coupure se trouve annulée.

3. **id et id.**
On a alors la configuration suivante :

$$\dfrac{\dfrac{}{\vdash A,\ A^{\perp}}\ \text{id} \qquad \dfrac{}{\vdash A,\ A^{\perp}}\ \text{id}}{\vdash A,\ A^{\perp}}\ \text{cut}$$

Elle peut être remplacée par le seul axiome $\dfrac{}{\vdash A,\ A^{\perp}}\ \text{id}$
La coupure a été éliminée.

4. **$\otimes$ et $\wp$.**
L'inférence I' avec celles qui la précèdent, forment la configuration :

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \vdash F_1,\ \Delta_1\end{array} \quad \begin{array}{c}\vdots\\ \vdash F_2,\ \Delta_2\end{array}}{\vdash F_1 \otimes F_2,\ \Delta_1,\ \Delta_2}\ \otimes \qquad \dfrac{\begin{array}{c}\vdots\\ \vdash F_1^{\perp},\ F_2^{\perp},\ \Delta'\end{array}}{\vdash F_1^{\perp} \wp F_2^{\perp},\ \Delta'}\ \wp}{\vdash \Delta_1,\ \Delta_2,\ \Delta'}\ \text{cut}$$

Elle peut alors être remplacée par celle-ci :

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \vdash F_1,\ \Delta_1\end{array} \quad \begin{array}{c}\vdots\\ \vdash F_1^{\perp},\ F_2^{\perp},\ \Delta'\end{array}}{\vdash F_2^{\perp},\ \Delta_1,\ \Delta'}\ \text{cut} \qquad \begin{array}{c}\vdots\\ \vdash F_2,\ \Delta_2\end{array}}{\vdash \Delta_1,\ \Delta_2,\ \Delta'}\ \text{cut}$$

Pour ce qui est de la complexité des coupures de l'ensemble, seule compte la coupure la plus haute dont la complexité est strictement inférieure à celle de I'.

5. **& et ⊕.**
L'inférence I' avec celles qui la précèdent, forment la configuration (pour ⊕, nous choisissons par exemple $\oplus_1$) :

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \vdash F_1,\ \Delta\end{array}\quad\begin{array}{c}\vdots\\ \vdash F_2,\ \Delta\end{array}}{\vdash F_1 \& F_2,\ \Delta}\,\& \qquad \dfrac{\begin{array}{c}\vdots\\ \vdash F_1^{\perp},\ \Delta'\end{array}}{\vdash F_1^{\perp} \oplus F_2^{\perp},\ \Delta'}\,\oplus_1}{\vdash \Delta,\ \Delta'}\,\text{cut}$$

Elle peut alors être remplacée par celle-ci :

$$\dfrac{\begin{array}{c}\vdots\\ \vdash F_1,\ \Delta\end{array}\quad\begin{array}{c}\vdots\\ \vdash F_1^{\perp},\ \Delta'\end{array}}{\vdash \Delta,\ \Delta'}\,\text{cut}$$

La complexité de la coupure a bien entendu diminué.

6. **! et ?.**
L'inférence I' avec celles qui la précèdent, forment la configuration :

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \vdash F,\ ?\Delta\end{array}}{\vdash !F,\ ?\Delta}\,! \qquad \dfrac{\begin{array}{c}\vdots\\ \vdash F^{\perp},\ \Delta'\end{array}}{\vdash ?F^{\perp},\ \Delta'}\,?}{\vdash ?\Delta,\ \Delta'}\,\text{cut}$$

Elle peut alors être remplacée par celle-ci :

$$\dfrac{\begin{array}{c}\vdots\\ \vdash F,\ ?\Delta\end{array}\quad\begin{array}{c}\vdots\\ \vdash F^{\perp},\ \Delta'\end{array}}{\vdash ?\Delta,\ \Delta'}\,\text{cut}$$

La complexité de la coupure a évidemment diminué.

7. **! et w?.**
L'inférence I' avec celles qui la précèdent, forment la configuration :

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \vdash F,\ ?\Delta\end{array}}{\vdash !F,\ ?\Delta}\,! \qquad \dfrac{\left.\begin{array}{c}\vdots\\ \vdash \Delta'\end{array}\right\}\mathcal{P}}{\vdash ?(F^{\perp}),\ \Delta'}\,\text{w?}}{\vdash ?\Delta,\ \Delta'}\,\text{cut}$$

Elle peut alors être remplacée par celle-ci :

$$\dfrac{\dfrac{\left.\begin{array}{c}\vdots\\ \vdash\ \Delta'\end{array}\right\}\mathcal{P}}{\vdash\ \Delta',\ ?\Delta}\ \mathrm{w?}}{}\ \mathrm{w?}$$

La coupure a été supprimée.

8. **! et c?.**
C'est le cas qui est cause de toutes les difficultés puisque la remontée de la coupure provoque une duplication partielle de la preuve. I' et les inférences au-dessus forment alors la configuration :

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \vdash\ F,\ ?\Delta\end{array}}{\vdash\ !F,\ ?\Delta}\,! \qquad \dfrac{\begin{array}{c}\vdots\\ \vdash\ ?(F^{\perp}),\ ?(F^{\perp}),\ \Delta'\end{array}}{\vdash\ ?(F^{\perp}),\ \Delta'}\ \mathrm{c?}}{\vdash\ ?\Delta,\ \Delta'}\ \mathrm{cut}$$

Elle peut alors être remplacée par celle-ci :

$$\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\\ \vdash\ F,\ ?\Delta\end{array}}{\vdash\ !F,\ ?\Delta}\,! \qquad \dfrac{\dfrac{\begin{array}{c}\vdots\\ \vdash\ F,\ ?\Delta\end{array}}{\vdash\ !F,\ ?\Delta}\,! \qquad \begin{array}{c}\vdots\\ \vdash\ ?(F^{\perp}),\ ?(F^{\perp}),\ \Delta'\end{array}}{\vdash\ ?(F^{\perp}),\ ?\Delta,\ \Delta'}\ \mathrm{cut}}{\vdash\ ?\Delta,\ ?\Delta,\ \Delta'}\ \mathrm{cut}}{\begin{array}{c}\vdots\\ \vdash\ ?\Delta,\ \Delta'\end{array}}\ \mathrm{c?}}{\vdash\ ?\Delta,\ \Delta'}\ \mathrm{c?}$$

Du point de vue de la complexité globale des coupures, seule compte dans la nouvelle preuve, la coupure supérieure dont la complexité est moindre que celle de I'.

9. **∀ et ∃.**
Dans ce dernier cas, on a la configuration suivante :

$$\dfrac{\dfrac{\mathcal{P}\left\{\begin{array}{c}\vdots\\ \vdash\ F[y/x],\ \Delta\end{array}\right.}{\vdash\ \forall x\ F,\ \Delta}\ \forall \qquad \dfrac{\begin{array}{c}\vdots\\ \vdash\ F^{\perp}[t/x],\ \Delta'\end{array}}{\vdash\ \exists x F^{\perp}[t/x],\ \Delta'}\ \exists}{\vdash\ \Delta,\ \Delta'}\ \mathrm{cut}$$

En substituant t à y dans la preuve $\mathcal{P}$, on obtient une preuve $\mathcal{P}[t/y]$ en renommant éventuellement certaines variables pour éviter que la condition relative à la règle $\forall$ ne soit violée. On peut alors remplacer la configuration précédente par celle-ci :

$$\dfrac{\mathcal{P}[t/y]\left\{\begin{array}{ll} \vdots & \vdots \\ \vdash\ F[t/x],\ \Delta & \vdash\ F^{\perp}[t/x],\ \Delta' \end{array}\right.}{\vdash\ \Delta,\ \Delta'}\ \text{cut}$$

La complexité de la coupure a diminué dans cette transformation.

En conclusion, on a pu dans tous les cas diminuer la complexité de la coupure I'. Comme ceci est valable pour toutes les coupures de $\mathcal{P}_1$ non précédées par d'autres, on obtient donc bien une preuve $\mathcal{P}_2$ dont la complexité des coupures est strictement inférieure à celle de $\mathcal{P}_1$.

Annexe C

Descente des inférences produisant les A-formules dans CLL↓

Théorème 4.4.1

Soit F une formule de CLL ne contenant pas les opérateurs $\otimes$ et !. Dans CLL↓, pour toute preuve $\mathcal{P}$ d'un séquent $\overset{a}{\vdash} F, \Delta$ où F est une A-formule, il existe une preuve qui a les propriétés suivantes :

- elle a même conclusion que $\mathcal{P}$ et conserve pour celle-ci, la même partition entre A-formules et B-formules ;

-les inférences introduisant les sous-formules de F sont à la fin de la preuve (elles ne sont suivies par aucune autre).

Preuve

Considérons dans CLL↓ une preuve $\mathcal{P}$ d'un séquent $\overset{a}{\vdash} F, \Delta$ où F est une A-formule qui ne contient pas les opérateurs $\otimes$ et !. Pour démontrer le théorème, nous allons procéder par induction sur la structure de F. Nous allons commencer par montrer le lemme suivant :

Lemme C.0.1 *On peut remplacer la preuve $\mathcal{P}$ par une preuve $\mathcal{P}'$ qui a les caractéristiques suivantes : elle a même conclusion que $\mathcal{P}$ et dans cette conclusion, la partition entre B-formules et A-formules n'a pas changé et enfin la dernière inférence est celle qui introduit F.*

Preuve C.0.1 *Nous allons procéder par induction sur la structure de $\mathcal{P}$.*
Etant donné que cette deuxième induction est imbriquée dans la première, pour les distinguer nous la désignerons par $\mathcal{IND}_2$ et la première par $\mathcal{IND}_1$.
Si l'inférence introduisant F est la dernière de $\mathcal{P}$, il n'y a rien à faire. Sinon, nous appliquons l'hypothèse de l'induction $\mathcal{IND}_2$ à toutes les preuves extraites de $\mathcal{P}$ qui contiennent une inférence introduisant F et qui se termine par un prémisse de la dernière inférence I de $\mathcal{P}$.
Nous obtenons, à la place de $\mathcal{P}$, une preuve $\mathcal{P}'$ où les inférences introduisant F se situent toutes en avant dernière position. La partition entre B-formules et A-formules d'inférences dans les prémisses de I n'a pas changée. Ensuite, quand c'est possible, par une permutation avec I, nous obtenons la preuve cherchée. Etudions maintenant les cas où cette permutation est impossible. En tenant compte de la commutativité des opérateurs logiques binaires, il y en a cinq si l'on se réfère au tableau 4.1 de permutabilité des inférences dans CLL↓.

1. `I est de type` $\&$
   Ce cas peut se diviser en trois selon le type des deux inférences qui précèdent I :
   - *l'une est de type $\oplus_1$ et l'autre de type $\oplus_2$.*
   - *l'une est de type ? et l'autre de type w.*
   - *les deux sont de type $\exists$.*

Ces trois cas se traitent de façon analogue. C'est pourquoi nous avons choisi de ne présenter que le premier. La preuve $\mathcal{P}'$ se présente alors ainsi :

$$\dfrac{\dfrac{\mathcal{P}_1\left\{\begin{array}{c}\vdots\\ \vdash^{a_1} G_1,\ F_1,\ \Delta\end{array}\right.}{\vdash^{a_1} G_1,\ F_1 \oplus F_2,\ \Delta}\oplus_1 \qquad \dfrac{\mathcal{P}_2\left\{\begin{array}{c}\vdots\\ \vdash^{a_2} G_2,\ F_2,\ \Delta\end{array}\right.}{\vdash^{a_2} G_2,\ F_1 \oplus F_2,\ \Delta}\oplus_2}{\vdash^{a_1 \wedge a_2} G_1 \& G_2,\ F_1 \oplus F_2,\ \Delta}\ \& I$$

F_1 et F_2 sont des A-formules respectives de $\mathcal{P}_1$ et $\mathcal{P}_2$ donc nous pouvons appliquer à ces preuves l' hypothèse de l'induction $\mathcal{IND}_1$, c'est-à-dire les remplacer par deux nouvelles preuves $\mathcal{P}'_1$ et $\mathcal{P}'_2$ qui ont les propriétés décrites dans l'énoncé du théorème. Ces propriétés nous permettent de décomposer $\mathcal{P}'_1$ de la manière suivante :

- une déduction $\mathcal{D}_1$ d'hypothèse $\vdash^{a_1} G_1,\ \Delta$ et de conclusion $\vdash^{a_1} G_1,\ F_1,\ \Delta$ formée uniquement d'inférences introduisant les sous-formules de F_1 qui est une A-formule de $\mathcal{P}'_1$.

- pour chaque hypothèse $\vdash^{a_1} G_1,\ \Delta$ de $\mathcal{D}_1$, une preuve de celle-ci qui conserve la partition entre B-formules et A-formules par rapport à $\mathcal{P}_1$. Nous noterons $\mathcal{P}_1^k$ la preuve de l'hypothèse numéro k.

Bien entendu, nous avons les mêmes résultats pour $\mathcal{P}'_2$ avec une déduction $\mathcal{D}_2$ et des preuves $\mathcal{P}_2^k$. Considérons les preuves $\mathcal{P}_1^1$ et $\mathcal{P}_2^1$. A partir d'elles, nous pouvons construire la preuve $\mathcal{P}_0$ suivante :

$$\dfrac{\mathcal{P}_1^1\left\{\begin{array}{c}\vdots\\ \vdash^{a_1} G_1,\ \Delta\end{array}\right. \qquad \mathcal{P}_2^1\left\{\begin{array}{c}\vdots\\ \vdash^{a_2} G_2,\ \Delta\end{array}\right.}{\vdash^{a_1 \wedge a_2} G_1 \& G_2,\ \Delta}\ \&$$

Dans la preuve $\mathcal{P}'_1$, remplaçons toutes les preuves $\mathcal{P}_1^k$ par la preuve $\mathcal{P}_0$ et toutes les occurrences de G_1 par $G_1 \& G_2$. Si la condition relative à la règle se trouve violée, on effectue les renommages nécessaires. Complétons cette preuve par une inférence de type $\oplus_1$ introduisant $F_1 \oplus F_2$. Nous obtenons une preuve qui a même conclusion que $\mathcal{P}$; dans cette conclusion, la partition entre B-formules et A-formules est la même et la dernière inférence de la preuve introduit bien la formule F. On a bien la preuve cherchée.

2. **! est de type !**

Dans $\mathcal{P}'$, l'inférence introduisant F qui est juste avant I, est nécessairement de type ? car elle ne peut pas permuter avec I. La preuve $\mathcal{P}'$ a donc la forme suivante :

$$\dfrac{\dfrac{\mathcal{P}_0\left\{\begin{array}{c}\vdots\\ \vdash^{a} G,\ F',\ ?\Delta\end{array}\right.}{\vdash^{a} G,\ ?F',\ ?\Delta}\ ?}{\vdash^{0} !G,\ ?F',\ ?\Delta}\ !I$$

Dans cette preuve, la formule F a nécessairement la forme $?F'$. On peut appliquer l'hypothèse de l'induction $\mathcal{IND}_1$ à la preuve $\mathcal{P}_0$ donc il existe une preuve de $\vdash^{a} G,\ ?\Delta$ qui conserve la même partition entre B-formules et A-formules dans sa conclusion que dans $\mathcal{P}_0$. En prolongeant cette preuve par une inférence ! introduisant $!G$ puis par une inférence w introduisant $?F'$, on obtient la preuve cherchée.

3. I est de type $\forall$

Puisque dans $\mathcal{P}'$, l'inférence introduisant F qui est juste avant I, ne peut pas permuter avec I, elle est nécessairement de type $\exists$ donc $\mathcal{P}'$ se présente ainsi :

$$\dfrac{\dfrac{\mathcal{P}_0\left\{\begin{array}{c}\vdots\\ \vdash^{a} F'[t/x],\ G[z/y],\ \Delta\end{array}\right.}{\vdash^{a} \exists x F',\ G[z/y],\ \Delta}\ \exists}{\vdash^{a} \exists x F',\ \forall y G,\ \Delta}\ \forall\ I$$

Dans cette preuve, la formule F a la forme $\exists x F'$. $F'[t/x]$ est une A-formule de $\mathcal{P}_0$ donc nous pouvons appliquer à cette preuve l' hypothèse de l'induction $\mathcal{IND}_1$, c'est-à-dire la remplacer par une nouvelle preuve $\mathcal{P}'_0$ qui a les propriétés décrites dans l'énoncé du théorème. Ces propriétés nous permettent de décomposer $\mathcal{P}'_0$ de la manière suivante :

- une déduction $\mathcal{D}$ d'hypothèse $\vdash^{a} G[z/y],\ \Delta$ et de conclusion $\vdash^{a} F'[t/x],\ G[z/y],\ \Delta$ formée uniquement d'inférences introduisant les sous-formules de $F'[t/x]$ qui est une formule d'affaiblissement de $\mathcal{P}'_0$.

- pour chaque hypothèse $\vdash^{a} G[z/y],\ \Delta$ de $\mathcal{D}$, une preuve de celle-ci qui conserve la partition entre B-formules et A-formules par rapport à $\mathcal{P}_0$. Nous noterons $\mathcal{P}_0^k$ la preuve de l'hypothèse numéro k.

Modifions la preuve $\mathcal{P}'_0$ de la manière suivante :

- insérons après chaque conclusion d'une preuve $\mathcal{P}_0^k$, une inférence $\forall$ introduisant $\forall y G$; elle vérifie bien la condition que z n'est pas libre dans $\forall y G, \Delta$;

- remplaçons dans la déduction $\mathcal{D}$ chaque occurrence de $G[z/y]$ par $\forall y G$;

- prolongeons la preuve par une inférence introduisant $\exists x F$.

La preuve obtenue remplit bien les conditions recherchées.

□

On a donc montré que l'on pouvait remplacer la preuve initiale $\mathcal{P}$ par une autre qui a les caractéristiques suivantes : elle a même conclusion et dans cette conclusion, la répartition entre B-formules et A-formules n'a pas changé ; en outre la formule F est introduite par la dernière inférence I.

Il faut maintenant aller plus loin et faire en sorte que toutes les inférences produisant des sous-formules de F soient à la fin de la preuve. Selon le type de I, nous sommes amenés à distinguer trois cas.

1. `I est de type` $\wp$ `ou c?.`

Nous choisirons par exemple I de type $\wp$. La preuve $\mathcal{P}$ a alors la forme suivante :

$$\dfrac{\mathcal{P}_0\left\{\begin{array}{c}\vdots\\ \vdash^{a} F_1,\ F_2,\ \Delta\end{array}\right.}{\vdash^{a} F_1 \wp F_2,\ \Delta}\ \wp$$

La A-formule F est ici $F_1 \wp F_2$. F_1 est donc aussi une A-formule. Alors en appliquant l'hypothèse d'induction $\mathcal{IND}_1$ à $\mathcal{P}_0$, on peut remplacer cette preuve par une autre qui a la forme :

$$\dfrac{\dfrac{\mathcal{P}_1\left\{\begin{array}{c}\vdots\\ \vdash^{a} F_2,\ \Delta\end{array}\right.}{\vdots} \qquad \cdots \qquad \dfrac{\mathcal{P}_n\left\{\begin{array}{c}\vdots\\ \vdash^{a} F_2,\ \Delta\end{array}\right.}{\vdots}}{\vdash^{a} F_1,\ F_2,\ \Delta}$$

Dans cette preuve, les inférences introduisant des sous-formules de F_1 sont à la fin. F_2 est toujours une A-formule ce qui nous permet d'appliquer une nouvelle fois l'hypothèse d'induction aux preuves $\mathcal{P}_1, \cdots, \mathcal{P}_n$. On les remplace par des preuves de même conclusion, où la même partition entre B-formules et A-formules est conservée et dans ces preuves, les inférences introduisant les sous-formules de F_2 sont à la fin.
En prolongeant la preuve globale ainsi transformée par l'inférence I, on obtient la preuve cherchée.

2. `I est de type` $\&$.
   La preuve $\mathcal{P}$ a la forme suivante :

$$\dfrac{\mathcal{P}_1\left\{\begin{array}{c}\vdots\\ \vdash^{a_1} F_1,\ \Delta\end{array}\right. \qquad \mathcal{P}_2\left\{\begin{array}{c}\vdots\\ \vdash^{a_2} F_1,\ \Delta\end{array}\right.}{\vdash^{a_1 \wedge a_2} F_1 \& F_2,\ \Delta}\ \&$$

   Par hypothèse de l'induction $\mathcal{IND}_1$, nous pouvons remplacer les preuves $\mathcal{P}_1$ et $\mathcal{P}_2$ par des preuves vérifiant la propriété décrite dans l'énoncé du théorème.
   La preuve globale ainsi obtenue est la preuve cherchée.

3. `I a un seul prémisse et a au plus une formule active.`
   La preuve $\mathcal{P}$ a alors la forme suivante :

$$\dfrac{\mathcal{P}_0\left\{\begin{array}{c}\vdots\\ \vdash^{a} \Delta_a,\ \Delta\end{array}\right.}{\vdash^{a} F,\ \Delta}$$

   Dans cette preuve, Δ_a est la partie active de l'inférence I. Si elle est vide, cela signifie que I est de type w ; alors la preuve $\mathcal{P}$ elle-même remplit les conditions du théorème.
   Sinon Δ_a est formé d'une seule formule. On applique alors l'hypothèse de l'induction I_1 à la preuve $\mathcal{P}_0$ que l'on remplace par une preuve vérifiant les propriétés du théorème. La preuve globale ainsi obtenue est celle qui est cherchée.

□

Annexe D

Transposition des résultats de permutabilité de CLL dans LL

Plutôt que d'exposer une démonstration générale, nous avons choisi d'expliquer notre démarche par le biais d'un exemple. Considérons dans LL, la preuve suivante :

$$
\begin{array}{l}
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{a \vdash a}\,\mathrm{id}}{a \vdash a \oplus b}\,\oplus_R}{a,\ 1 \vdash a \oplus b}\,1_L}{a \otimes 1 \vdash a \oplus b}\,\otimes_L \qquad \dfrac{}{0 \vdash b}\,0_L}{a \multimap 0,\ a \otimes 1 \vdash b}\,\multimap_L}{(a \oplus b) \multimap 0 \vdash (a \otimes 1)^{\perp},\ b}\,{}^{\perp}_{R}}{(a \oplus b) \multimap 0 \vdash (a \otimes 1)^{\perp} \wp b}\,\wp_R
\end{array}
$$

Supposons que nous voulions déplacer l'inférence de type $\multimap_L$ dans le sens où c'est le plus facile.
Nous commençons par transposer la preuve dans CLL. Pour cela, dans chaque conclusion intermédiaire, toute formule de la partie gauche est remplacée par sa négation dans la partie droite. Chaque inférence qui n'a pas sa partie principale et ses formules actives toutes dans la partie droite des séquents, est remplacée par une inférence correspondante de CLL sauf les inférences introduisant des négations qui sont purement et simplement supprimées. Nous obtenons la preuve de CLL suivante :

$$
\begin{array}{l}
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash a,\ a^{\perp}}\,\mathrm{id}}{\vdash a \oplus b,\ a^{\perp}}\,\oplus}{\vdash a \oplus b,\ a^{\perp},\ \perp}\,\perp}{\vdash a \oplus b,\ a^{\perp} \wp \perp}\,\wp \qquad \dfrac{}{\vdash \top,\ b}\,\top}{\vdash (a \oplus b) \otimes \top,\ a^{\perp} \wp \perp,\ b}\,\otimes}{\vdash (a \oplus b) \otimes \top,\ (a^{\perp} \wp \perp) \wp b}\,\wp
\end{array}
$$

L'inférence que nous voulions déplacer, est maintenant devenue une inférence de type $\otimes$. D'après l'étude de permutabilité que nous avons effectuée dans CLL (voir le théorème 2.2.1), nous avons intérêt à monter

cette inférence dans la preuve. Nous obtenons la preuve suivante :

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash a,\ a^{\perp}}\,\mathrm{id}}{\vdash a \oplus b,\ a^{\perp}}\,\oplus \qquad \dfrac{}{\vdash \top,\ b}\,\top}{\vdash (a \oplus b) \otimes \top,\ a^{\perp},\ b}\,\otimes}{\vdash (a \oplus b) \otimes \top,\ a^{\perp},\ \perp,\ b}\,\perp}{\vdash (a \oplus b) \otimes \top,\ a^{\perp} \wp \perp,\ b}\,\wp}{\vdash (a \oplus b) \otimes \top,\ (a^{\perp} \wp \perp) \wp b}\,\wp
$$

Il ne reste plus qu'à transposer la preuve obtenue dans LL en effectuant l'opération opposée à celle effectuée initialement. La preuve finale est la suivante :

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{a \vdash a}\,\mathrm{id}}{a \vdash a \oplus b}\,\oplus_R \qquad \dfrac{}{0 \vdash b}\,0_L}{(a \oplus b) \multimap 0,\ a \vdash b}\,\multimap_L}{(a \oplus b) \multimap 0,\ a,\ 1 \vdash b}\,1_L}{(a \oplus b) \multimap 0,\ a \otimes 1 \vdash b}\,\otimes_L}{(a \oplus b) \multimap 0 \vdash (a \otimes 1)^{\perp},\ b}\,\perp_R}{(a \oplus b) \multimap 0 \vdash (a \otimes 1)^{\perp} \wp b}\,\wp_R
$$

On peut donc en conclure que dans LL, les inférences de type $\multimap_L$ auront la même facilité à monter que celles de type $\otimes$ dans CLL.

On peut étendre ce raisonnement à n'importe quel type d'inférence ce qui permet de transposer dans LL, le tableau du théorème 2.2.1 qui fait le bilan de la permutabilité d'inférences dans CLL. Nous obtenons ainsi le tableau D.1.

TAB. D.1 - *Permutabilité d'inférences dans les preuves de LL*

$t_2 \backslash t_1$	$\perp_L$	$\perp_R$	$\wp_L \multimap_L$ $\otimes_R$	$\otimes_L$ $\wp_R \multimap_R$	1_L $\perp_R$	$\oplus_L$ $\&_R$	$\&_L$ $\oplus_R$	$!_L$ $?_R$	$w!_L$ $w?_R$	$c!_L$ $c?_R$	$?_L$ $!_R$	$\exists_L$ $\forall_R$	$\forall_L$ $\exists_R$	*cut*
$\perp_L$											*np*			
$\perp_R$											*np*			
$\wp_L \multimap_L$ $\otimes_R$											*np*			
$\otimes_L$ $\wp_R \multimap_R$			*np*								*np*			
1_L $\perp_R$											*np*			
$\oplus_L$ $\&_R$	–	–	*np*	–	–	–	*np*	*np*	*np*	–	*np*	–	*np*	*np*
$\&_L$ $\oplus_R$											*np*			
$!_L$ $?_R$														
$w!_L$ $w?_R$														
$c!_L$ $c?_R$			*np*											*np*
$?_L$ $!_R$	×	×	×	×	×	×	×	*np*			×	×	×	*np*
$\exists_L$ $\forall_R$											*np*		*np*	*np*
$\forall_L$ $\exists_R$											*np*			
cut											*np*			

Annexe E

Réduction de l'interface relative d'un processus à une interface linéaire

Théorème 8.3.5

Pour tout processus P de CPL, il existe une interface linéaire qui est une réduction de $\mathcal{I}_{\mathcal{P}}(P)$ et qui a les mêmes variables libres que P.
En outre, si 0 n'est pas un sous-processus de P, alors les éléments de cette interface ne comprennent que des messages présents dans P avec éventuellement un renommage de variables liées.

Preuve

Définition E.0.1 *Pour éviter de nous répéter, nous dirons d'une interface qui est une réduction de l'interface relative $\mathcal{I}_{\mathcal{P}}(P)$ d'un processus P quelconque de CPL qu'elle est* normalisée *si elle a les mêmes variables libres que P et si, lorsque 0 n'est pas un sous-processus de P, les éléments de cette interface ne comprennent que des messages présents dans P avec éventuellement un renommage de variables liées.*

Nous allons procéder par induction sur la structure de P. Selon la forme de P, nous sommes amenés à distinguer les cas suivants :

1. P = M
   D'après le théorème 8.1.3, {M.1} est une réduction de l'interface absolue de M, à plus forte raison de sa interface relative aussi. Et il est immédiat de vérifier que cette réduction est normalisée.

2. P = 1
   Il est évident que { 1} est l'interface qui convient.

3. P = 0
   l'interface de 0 est $\mathcal{P}$ tout entier mais suivant le lemme 8.2.1, $\mathcal{L}$ est une réduction de cette interface. Pour normaliser $\mathcal{L}$, il reste à se débarrasser de ses éléments qui ont des variables libres. Considérons un processus linéaire quelconque L qui a comme variables libres $x_1, \cdots, x_n$. Le théorème 8.2.4 montre que le séquent $L \vdash \exists x_1 \cdots \exists x_n L$ est prouvable donc il est possible de réduire $\mathcal{L}$ à une interface linéaire normalisée.

4. P = $\underline{\text{M}}$.P'
   Par hypothèse d'induction, il existe une interface linéaire I' qui est une réduction normalisée de l'interface relative de P'.
   Soit l'ensemble I des processus linéaires de la forme M.L' où L' est un élément quelconque de I'. Montrons que I est une réduction normalisée de $\mathcal{I}_{\mathcal{P}}$(P).
   Tout d'abord, nous allons montrer que I est une partie de $\mathcal{I}_{\mathcal{P}}$(P). Soit M.L' un élément quelconque

de I. Puisque $L' \in \mathcal{I}_{\mathcal{L}}(P')$, alors : $P',\ L' \vdash 1$. Donc par application successive des règles RC_L et SD_L ,nous pouvons inférer :$P',\ M,\ M.L' \vdash 1$ et $\underline{M}.P',\ M.L' \vdash 1$.
Par conséquent : $M.L' \in \mathcal{I}_{\mathcal{P}}(P)$ ce qui implique : $I \subseteq \mathcal{I}_{\mathcal{P}}(P)$.
Maintenant nous allons montrer que I se déduit de $\mathcal{I}_{\mathcal{P}}$(P). Soit Q un co-processus quelconque de P qui soit un processus.
Par hypothèse : $\underline{M}.P',\ Q \vdash 1$. Comme nous pouvons descendre au maximum les inférences de type SD_L dans les preuves, nous en déduisons : $P',\ M,\ Q \vdash 1$.
Donc $\underline{M}.Q \in \mathcal{I}_{\mathcal{P}}(P')$. Il s'ensuit que nous pouvons trouver un élément L' de I' tel que $\underline{M}.Q \vdash L'$. D'où $M, Q \vdash L'$ et finalement $Q \vdash M.L'$ ce qu'il fallait démontrer.
Etant donné la forme de I et le fait que I' est une réduction normalisée, I l'est aussi.

5. P = M.P'
Par hypothèse d'induction, il existe une interface linéaire I' qui est une réduction normalisée de l'interface relative de P'.
Soit I l'ensemble des processus linéaires de la forme $\underline{M}.L'$ où L' est un élément quelconque de I'.
Montrons que I est une réduction normalisée de $\mathcal{I}_{\mathcal{P}}$(P).
Tout d'abord, nous allons montrer que I est une partie de $\mathcal{I}_{\mathcal{P}}$(P). Soit $\underline{M}.L'$ un élément quelconque de I. Puisque $L' \in \mathcal{I}_{\mathcal{P}}(P')$, alors : $P',\ L' \vdash 1$. Ensuite, par application des règles RC_L et SD_L, nous pouvons inférer successivement : $M.P',\ M,\ L' \vdash 1$ et $M.P',\ \underline{M}.L' \vdash 1$. Par conséquent : $\underline{M}.L' \in \mathcal{I}_{\mathcal{P}}(P)$ ce qui implique : $I \subseteq \mathcal{I}_{\mathcal{P}}(P)$.
Maintenant nous allons montrer que I se déduit de $\mathcal{I}_{\mathcal{P}}$(P). Soit Q un co-processus quelconque de P qui soit un processus. Par hypothèse : $M.P',\ Q \vdash 1$.
Considérons une preuve quelconque de ce séquent. Elle a la forme suivante :

$$\begin{array}{c} \vdots \\ \dfrac{P',\ P",\ \Gamma \vdash 1}{M.P',\ M,\ P",\ \Gamma \vdash 1}\ RC_L \\ \dfrac{}{M.P',\ \underline{M}.P",\ \Gamma \vdash 1}\ SD_L \\ \vdots \\ M.P',\ Q \vdash 1 \end{array}$$

Nous avons donc $\bigotimes(P",\Gamma) \in \mathcal{I}_{\mathcal{P}}(P')$. Donc nous pouvons trouver un élément L' de I' tel que $P",\ \Gamma \vdash L'$. Comme I' est normalisée, les variables libres de L' sont des variables libres de P'. Nous pouvons donc modifier la preuve ci-dessus de la façon suivante :

$$\begin{array}{c} \vdots \\ \dfrac{P",\ \Gamma \vdash L'}{M,\ P",\ \Gamma \vdash \underline{M}.L'}\ SD_R \\ \dfrac{}{\underline{M}.P",\ \Gamma \vdash \underline{M}.L'}\ SD_L \\ \vdots \\ Q \vdash \underline{M}.L' \end{array}$$

Nous venons donc de prouver que $Q \vdash \underline{M}.L'$.
Donc I est une réduction de $\mathcal{I}_{\mathcal{P}}$(P) et il est facile de vérifier qu'elle est normalisée.

6. $\mathsf{P} = \mathsf{P}_1 \otimes \mathsf{P}_2$
Par hypothèse d'induction, $\mathcal{I}_{\mathcal{P}}(P_1)$ et $\mathcal{I}_{\mathcal{P}}(P_2)$ ont des réductions normalisées respectives I_1 et I_2. Soit I l'interface linéaire constituée par tous les processus obtenus par entrelacement d'un élément quelconque de I_1 avec un élément quelconque de I_2. Montrons que I est une réduction normalisée de $\mathcal{I}_{\mathcal{P}}(\mathrm{P})$.
Nous allons commencer par montrer que I est une partie de $\mathcal{I}_{\mathcal{P}}(\mathrm{P})$. Soit L un élément quelconque de I. Il existe donc deux processus linéaires L_1 et L_2 de I_1 et I_2 tels que $L \in L_1 * L_2$. Le lemme 8.2.2 nous permet de déduire $L \in \mathcal{I}_{\mathcal{P}}(P)$ ce qui implique $I \subseteq \mathcal{I}_{\mathcal{P}}(P)$.
Maintenant, nous allons prouver que I se déduit de $\mathcal{I}_{\mathcal{P}}(\mathrm{P})$. Soit Q un élément quelconque de $\mathcal{I}_{\mathcal{P}}(\mathrm{P})$. Par hypothèse : $P_1 \otimes P_2,\ Q \vdash 1$. D'où : $P_1,\ P_2,\ Q \vdash 1$. Donc $\bigotimes(P_2, Q) \in \mathcal{I}_{\mathcal{P}}(P_1)$.
Par conséquent, nous pouvons trouver un élément L_1 de I_1 tel que $P_2,\ Q \vdash L_1$. Le théorème 8.2.2 nous permet ensuite d'en déduire $P_2,\ \overline{L_1},\ Q \vdash 1$. Donc $\bigotimes(\overline{L_1},\ Q) \in \mathcal{I}_{\mathcal{P}}(P_2)$. Par conséquent, nous pouvons trouver un élément L_2 de I_2 tel que $\overline{L_1},\ Q \vdash L_2$. D'où, d'après le théorème 8.2.2 : $\overline{L_1},\ \overline{L_2},\ Q \vdash 1$ et d'après le théorème 8.2.3, nous pouvons trouver un élément L de $L_1 * L_2$ tel que : $Q \vdash L$. On a donc prouvé que I se déduit de $\mathcal{I}_{\mathcal{P}}(\mathrm{P})$. Et l'on vérifie facilement que I est une réduction normalisée.

7. $\mathsf{P} = \mathsf{P}_1 \& \mathsf{P}_2$
Par hypothèse d'induction, $\mathcal{I}_{\mathcal{P}}(P_1)$ et $\mathcal{I}_{\mathcal{P}}(P_2)$ ont des réductions normalisées respectives I_1 et I_2. Soit I la réunion de I_1 et de I_2. Montrons que I est une réduction normalisée de $\mathcal{I}_{\mathcal{P}}(\mathrm{P})$.
Tout d'abord, nous allons montrer que I est une partie de $\mathcal{I}_{\mathcal{P}}(\mathrm{P})$. Soit L un élément quelconque de I. Cela signifie que $P_1,\ L \vdash 1$ ou $P_2,\ L \vdash 1$. Dans les deux cas, à l'aide de la règle ALT_L nous pouvons inférer : $P_1 \& P_2,\ L \vdash 1$ ce qui signifie : $L \in \mathcal{I}_{\mathcal{P}}(P)$. Nous avons donc montré : $I \subseteq \mathcal{I}_{\mathcal{P}}(P)$.
Maintenant, nous allons prouver que I se déduit de $\mathcal{I}_{\mathcal{P}}(\mathrm{P})$. Soit Q un élément quelconque de $\mathcal{I}_{\mathcal{P}}(\mathrm{P})$. Par hypothèse : $P_1 \& P_2,\ Q \vdash 1$. Comme nous pouvons descendre au maximum les inférences de type ALT_L dans les preuves, nous pouvons en déduire soit $P_1,\ Q \vdash 1$, soit $P_2,\ Q \vdash 1$. Dans les deux cas, il existe unélément L de I tel que $Q \vdash L$.
Nous concluons en vérifiant que I est bien une réduction normalisée.

8. $\mathsf{P} = \forall \mathsf{x} \mathsf{P}'$
Dans ce cas, il va nous falloir démontrer auparavant le lemme suivant.

Lemme E.0.2 *Si le séquent $\Gamma_1[t/x],\ \Gamma_2 \vdash 1$ est prouvable dans CPL et si x n'est pas libre dans Γ_2 et t, alors il existe un système de processus Γ_3 tel que $\Gamma_3[t/x] = \Gamma_2$ et le séquent $\Gamma_1,\ \Gamma_3 \vdash 1$ est prouvable.*

Preuve E.0.2 *Nous procédons par induction sur la structure de la preuve de $\Gamma_1[t/x],\ \Gamma_2 \vdash 1$. Selon le type de la dernière inférence I de la preuve, nous sommes amenés à distinguer les cas suivants :*

(a) I a le type SD_L, RC_L, $TERM_L$, PAR_L, ALT_L ou REC_L.
Par exemple, considérons le cas où I a le type RC_L. Ce cas est le plus complexe car I a deux formules principales (les autres cas sont plus simples mais se traitent selon le même principe). Selon la répartition des formules principales dans $\Gamma_1[t/x]$ et Γ_2, nous devons distinguer trois sous-cas.

i. **Les formules principales sont dans $\Gamma_1[t/x]$.**
Alors I a la forme suivante :

$$\frac{P[t/x],\ \Gamma_1'[t/x],\ \Gamma_2 \vdash 1}{M[t/x], M[t/x].P[t/x],\ \Gamma_1'[t/x],\ \Gamma_2 \vdash 1}\ RC_L$$

Par hypothèse d'induction, il existe un système de processus Γ_3 tel que $\Gamma_3[t/x] = \Gamma_2$ et le séquent $P,\ \Gamma_1',\ \Gamma_3 \vdash 1$ soit prouvable.
Grâce à la règle RC_L, nous pouvons inférer $M,\ M.P,\ \Gamma_1',\ \Gamma_3 \vdash 1$ qui est $\Gamma_1,\ \Gamma_3 \vdash 1$.

ii. **Les formules principales sont dans Γ_2.**
Alors I a la forme suivante :

$$\frac{P,\ \Gamma_1[t/x],\ \Gamma'_2\ \vdash\ 1}{M, M.P,\ \Gamma_1[t/x],\ \Gamma'_2\ \vdash\ 1}\ RC_L$$

Par hypothèse d'induction, il existe un système de processus Γ_3 tel que $\Gamma'_3[t/x] = \Gamma'_2$ et un processus P' tel que $P'[t/x] = P$ et que le séquent $P',\ \Gamma_1,\ \Gamma'_3\ \vdash\ 1$ soit prouvable.
A l'aide de la règle RC_L, nous pouvons inférer $M,\ M.P',\ \Gamma_1,\ \Gamma'_3\ \vdash\ 1$. Comme x n'est pas libre dans Γ_2, $M\ =\ M[t/x]$ et nous pouvons choisir $M,\ M.P,\ \Gamma'_3$ pour Γ_3 ; il vérifie les conditions requises.

iii. **Les formules principales sont réparties entre $\Gamma_1[t/x]$ et Γ_2.**
Par exemple, I a la forme suivante (le cas symétrique se traite de façon analogue) :

$$\frac{P,\ \Gamma'_1[t/x],\ \Gamma'_2\ \vdash\ 1}{M[t/x], M[t/x].P,\ \Gamma'_1[t/x],\ \Gamma'_2\ \vdash\ 1}\ RC_L$$

Par hypothèse d'induction, il existe un système de processus Γ_3 tel que $\Gamma'_3[t/x] = \Gamma'_2$ et un processus P' tel que $P'[t/x] = P$ et que le séquent $P',\ \Gamma'_1,\ \Gamma'_3\ \vdash\ 1$ soit prouvable. A l'aide de la règle RC_L, nous pouvons inférer $M,\ M.P',\ \Gamma'_1,\ \Gamma'_3\ \vdash\ 1$. Alors nous pouvons choisir $M[t/x],\ M[t/x].P,\ \Gamma'_3$ pour Γ_3 car il vérifie les conditions requises.

(b) I has the type RES_L.
Comme dans le cas précédent, nous sommes amenés à distinguer plusieurs sous-cas selon la localisation de la formule principale de I. Ici, les choses sont plus simples car la formule principale est unique.

- **La formule principale est dans $\Gamma_1[t/x]$.**
Alors I a la forme suivante :

$$\frac{P[t/x][y/z],\ \Gamma'_1[t/x],\ \Gamma_2\ \vdash\ 1}{\exists z(P[t/x]),\ \Gamma'_1[t/x],\ \Gamma_2\ \vdash\ 1}\ RES_L$$

Quand nous écrivons cette inférence, nous supposons que z est différent de x et n'est pas libre dans t de façon à avoir l'égalité : $\exists z(P[t/x])\ =\ (\exists zP)[t/x]$; si ce n'est pas le cas, nous renommons z.
Puisque y n'est pas libre dans $\Gamma'_1[t/x]$, Γ_2, nous pouvons la choisir différente de x. Alors nous avons l'égalité suivante : $P[t/x][y/z]\ =P[y/z][t/x]$.
Par hypothèse d'induction, il existe un système de processus Γ_3 tel que $\Gamma_3[t/x] = \Gamma_2$ et que le séquent $P[y/z],\ \Gamma'_1,\ \Gamma_3\ \vdash\ 1$ est prouvable.
A l'aide de la règle RES_L, nous pouvons inférer $\exists zP,\ \Gamma'_1,\ \Gamma_3\ \vdash\ 1$, c'est-à-dire $\Gamma_1,\ \Gamma_3\ \vdash 1$.

- **La formule principale est dans Γ_2.**
Alors I a la forme suivante :

$$\frac{P[y/z],\ \Gamma_1[t/x],\ \Gamma'_2\ \vdash\ 1}{\exists zP,\ \Gamma_1[t/x],\ \Gamma'_2\ \vdash\ 1}\ RES_L$$

Par hypothèse d'induction, il existe un système de processus Γ'_3 tel que $\Gamma'_3[t/x] = \Gamma'_2$ et un processus P' tel que $P'[t/x]\ =\ P[y/z]$ et que le séquent $P',\ \Gamma_1,\ \Gamma'_3\ \vdash\ 1$ est prouvable.
Soit le processus $P"$ égal à $P'[z/y]$. Comme nous pouvons supposer z différent de x, alors z n'est pas libre dans P' d'où l'égalité $P'\ =\ P"[y/z]$ .A l'aide de la règle RES_L, nous pouvons inférer $\exists zP",\ \Gamma_1,\ \Gamma'_3\ \vdash\ 1$.
Nous avons l'égalité : $P"[y/z][t/x]\ =\ P[y/z]$. Comme y n'est pas libre dans $\Gamma'_1[t/x]$, Γ_2, nous pouvons le choisir différent de x et nous pouvons supposer aussi que z n'est pas libre dans t. D'où, nous pouvons déduire :$P"[t/x][y/z]\ =\ P"[y/z][t/x]\ =\ P[y/z]$ et $P"[t/x]\ =\ P$. Alors nous pouvons choisir $\exists zP"$, Γ'_3 pour Γ_3 car il vérifie les conditions requises.

(c) I a le type GEN_L.
Alors nous sommes amenés à distinguer deux sous-cas en fonction de la localistation de la formule principale.

i. **La formule principale est dans $\Gamma_1[t/x]$.**
Alors I a la forme suivante :

$$\frac{P[t/x][t'/z],\ \Gamma_1'[t/x],\ \Gamma_2\ \vdash\ 1}{\forall z(P[t/x]),\ \Gamma_1'[t/x],\ \Gamma_2\ \vdash\ 1}\ GEN_L$$

Quand nous écrivons cette inférence, nous supposons que z est différent de x et n'est pas libre dans t de façon à avoir l'égalité : $\forall z(P[t/x]) = (\forall zP)[t/x]$ *; sinon nous renommons z. Puisque x n'est pas libre dans* Γ_2 *et dans t, nous pouvons supposer qu'il n'est pas libre dans t'. Alors, nous avons l'égalité :* $P[t/x][t'/z] = P[t'/z][t/x]$.
Par hypothèse d'induction, il existe un système de processus Γ_3 *tel que* $\Gamma_3[t/x] = \Gamma_2$ *et que le séquent* $P[t'/z],\ \Gamma_1',\ \Gamma_3\ \vdash\ 1$ *est prouvable. A l'aide de la règle* GEN_L*, nous pouvons inférer* $\forall zP,\ \Gamma_1',\ \Gamma_3\ \vdash\ 1$*, c'est-à-dire* $\Gamma_1,\ \Gamma_3\ \vdash 1$.

ii. **La formule principale est dans Γ_2.**
Alors I a la forme suivante :

$$\frac{P[t'/z],\ \Gamma_1[t/x],\ \Gamma_2'\ \vdash\ 1}{\forall zP,\ \Gamma_1[t/x],\ \Gamma_2'\ \vdash\ 1}\ GEN_L$$

Par hypothèse d'induction, il existe un système de processus Γ_3' *tel que* $\Gamma_3'[t/x] = \Gamma_2'$ *et un processus* P' *tel que* $P'[t/x] = P[t'/z]$ *et que le séquent* $P',\ \Gamma_1,\ \Gamma_3'\ \vdash\ 1$ *soit prouvable. Nous pouvons supposer z non libre dans t et comme x n'est pas libre dans* Γ_2*, nous pouvons aussi supposer que les variables de t' ne contient pas x. Nous pouvons donc trouver un processus P" tel que* $P"[t/x] = P'$ *et* $P"[t'/z] = P'$*. A l'aide de la règle* GEN_L*, nous pouvons inférer* $\forall zP",\ \Gamma_1,$
$\Gamma_3'\ \vdash\ 1$*. Alors nous pouvons choisir* $\forall zP",\ \Gamma_3'$ *pour* Γ_3 *car il vérifie les conditions requises.*□

Revenons maintenant à la démonstration du théorème proprement dit. Par hypothèse d'induction, il existe une interface linéaire I' qui est une réduction normalisée de l'interface relative de P'.
Soit l'ensemble I des processus linéaires de la forme $\exists xL'$ où L' est un élément quelconque de I'. Montrons que I est une réduction normalisée de $\mathcal{I}_{\mathcal{P}}$(P).
Tout d'abord, nous allons montrer que I est une partie de $\mathcal{I}_{\mathcal{P}}$(P). Soit $\exists xL'$ un élément quelconque de I. Puisque $L' \in \mathcal{I}_{\mathcal{L}}(P')$, nous avons : $P',\ L'\ \vdash\ 1$. A l'aide des règles GEN_L et RES_L, nous pouvons inférer successivement : $\forall xP',\ L'\ \vdash\ 1$ et $\forall xP',\ \exists xL'\ \vdash\ 1$ ce qui signifie :$L \in \mathcal{I}_{\mathcal{P}}(P)$. Nous avons donc prouvé : $I \subseteq \mathcal{I}_{\mathcal{P}}(P)$.
Maintenant, nous allons prouver que I se déduit de $\mathcal{I}_{\mathcal{P}}$(P). Soit Q un élément quelconque de $\mathcal{I}_{\mathcal{P}}$(P). Par hypothèse : $\forall xP',\ Q\ \vdash\ 1$. Considérons une preuve quelconque de ce séquent. Elle a la forme :

$$\begin{array}{c}\vdots\\ \dfrac{P'[t/x],\ \Gamma\ \vdash\ 1}{\forall xP',\ \Gamma\ \vdash\ 1}\ GEN_L\\ \vdots\\ \forall xP',\ Q\ \vdash\ 1\end{array}$$

Nous pouvons supposer que x n'est pas libre dans Γ et dans t. D'où, d'après le lemme E.0.2, nous pouvons trouver un système de processus Γ' tel que : $\Gamma\ =\ \Gamma'[t/x]$ et que le séquent $P',\ \Gamma'\ \vdash\ 1$

soit prouvable.
Donc il existe un élément L' de I' tel que : $\Gamma' \vdash L'$. D'où nous pouvons déduire : $\Gamma'[t/x] \vdash L'[t/x]$, c'est-à-dire $\Gamma \vdash L'[t/x]$. En outre, comme S' est une réduction normalisée, les variables libres de L' sont des variables libres de P', ce qui permet de modifier la preuve précédente de la façon suivante sans risque de voir la condition relative à RES_L violée :

$$\begin{array}{c} \vdots \\ \dfrac{\Gamma \vdash L'[t/x]}{\Gamma \vdash \exists x L'} REI_R \\ \vdots \\ Q \vdash \exists x L' \end{array}$$

Nous terminons l'analyse de cas en remarquant que I est une réduction normalisée.

9. P = ∃xP'
Par hypothèse d'induction, il existe une interface linéaire I' qui est une réduction normalisée de l'interface relative de P'.
Soit l'ensemble I des processus linéaires de la forme $\forall x L'$ où L' est un élément quelconque de I'.
Montrons que I est une réduction normalisée de $\mathcal{I}_{\mathcal{P}}$(P).
Tout d'abord, nous allons montrer que I est une partie de $\mathcal{I}_{\mathcal{P}}$(P). Soit $\forall x L'$ un élément quelconque de I. Puisque $L' \in \mathcal{I}_{\mathcal{L}}(P')$, nous avons : $P', L' \vdash 1$. A l'aide des règles GEN_L et RES_L, nous pouvons inférer successivement : $P', \forall x L' \vdash 1$ et $\exists x P', \forall x L' \vdash 1$ ce qui signifie : $L \in \mathcal{I}_{\mathcal{P}}(P)$. Nous avons donc prouvé : $I \subseteq \mathcal{I}_{\mathcal{P}}(P)$.
Maintenant, nous allons prouver que I se déduit de $\mathcal{I}_{\mathcal{P}}$(P). Soit Q un élément quelconque de $\mathcal{I}_{\mathcal{P}}$(P).
Par hypothèse : $\exists x P', Q \vdash 1$. Comme nous pouvons descendre au maximum les inférences de type RES_L dans les preuves, nous en déduisons : $P'[y/x], Q \vdash 1$ avec y non libre dans $\{\exists x P', Q\}$.
Donc il existe un élément L' de I' tel que : $Q \vdash L'[y/x]$. Ensuite, à l'aide de la règle GEN_R, nous pouvons inférer : $Q \vdash \forall x L'$ car y n'est pas libre dans $Q \vdash \forall x L'$.
Nous concluons en vérifiant que I est bien une réduction normalisée.

10. P =!P'
Par hypothèse d'induction, il existe une interface linéaire I' qui est une réduction normalisée de l'interface relative de P'.
A partir de I', nous pouvons construire inductivement une suite de interfaces linéaires $(I'_n)_{n \in \mathcal{N}}$ telle que pour tout $n > 0$, I'_n est une réduction linéaire normalisée de $\mathcal{I}_{\mathcal{P}}(\underbrace{P' \otimes \cdots \otimes P'}_{ntimes})$; si n = 0, $I'_0 = \{1\}$.
Soit I la réunion de tous les I'_n pour $n \in \mathbb{N}$. Montrons que I est une réduction linéaire normalisée de $\mathcal{I}_{\mathcal{P}}$(P) .
Tout d'abord, nous allons prouver que I est une partie de $\mathcal{I}_{\mathcal{P}}$(P). Soit L un élément un élément quelconque de I. Il existe donc un entier n tel que $L \in S'_n$. Donc : $P' \otimes \cdots \otimes P', L \vdash 1$. D'où : $P', \cdots, P', L \vdash 1$ et $!P', P', \cdots, P', L \vdash 1$.
Puis en utilisant n fois la règle REC_L, nous pouvons inférer le séquent suivant : $!P', L \vdash 1$, ce qui signifie : $L \in \mathcal{I}_{\mathcal{P}}(P)$. Nous avons donc prouvé : $I \subseteq \mathcal{I}_{\mathcal{P}}(P)$.
Maintenant il s'agit de montrer que I se déduit de $\mathcal{I}_{\mathcal{P}}(P)$. Soit Q un co-processus quelconque de !P' qui soit aussi un processus. Par hypothèse : $!P', Q \vdash 1$. Comme nous pouvons descendre au maximum les inférences de type REC_L dans les preuves qui n'utilisent pas les règles ALT_R et REC_R, nous en déduisons : $P', \cdots, P', Q \vdash 1$ et : $P' \otimes \cdots \otimes P', Q \vdash 1$. Nous pouvons donc trouver un entier n et un élément L de I'_n tels que $Q \vdash L$. Nous pouvons conclure que $L \in I$.
Enfin nous vérifions que I est une réduction normalisée. □

université HENRI POINCARÉ NANCY 1

Nom : PERRIER

Prénom : Guy

DOCTORAT de l'UNIVERSITE HENRI POINCARE, NANCY-I

en INFORMATIQUE

VU, APPROUVÉ ET PERMIS D'IMPRIMER

Nancy, le - 1 FEV. 1995 n°49

Le Président de l'Université

UNIVERSITÉ HENRI POINCARÉ, NANCY I
LE PRESIDENT

J.P. FINANCE

3C-F9

UNIVERSITÉ HENRI POINCARÉ - NANCY 1
24-30, rue Lionnois B.P.3069 54013 Nancy cedex
Tél : 83 32 81 81 Fax : 83 32 95 90
Adresse électronique : _____ @presn1.u-nancy.fr

Résumé

Cette thèse se propose d'étudier la construction de preuves en logique linéaire et sur la base de cette étude, de l'utiliser ensuite comme paradigme de calcul pour la programmation parallèle.
Dans le calcul des séquents linéaire, nous montrons que la propriété de permutabilité d'inférences joue un rôle essentiel dans la normalisation des preuves en vue de leur construction automatique. Cette normalisation est déterminée par deux facteurs : le fragment logique dans lequel nous nous plaçons et le sens de construction des preuves envisagé. Nous mettons ainsi à jour une dualité entre principes de construction des preuves en chaînage avant et principes de construction des preuves en chaînage arrière à partir de laquelle nous proposons une méthode pour normaliser les preuves et élaborer des stratégies de construction dans un fragment donné de la logique linéaire. Appliquée de façon systématique dans la logique linéaire toute entière, elle nous permet d'y proposer une stratégie originale de construction des preuves en chaînage avant.
Cette méthode va ensuite nous aider à choisir un fragment approprié de la logique linéaire intuitionniste pour y élaborer un calcul de processus asynchrone, CPL (Concurrent Programming Logic), basé sur la construction de preuves. Les formules y sont interprétées comme des processus et les preuves, avec certaines restrictions, comme des réductions de processus. Nous proposons une sémantique opérationnelle sous forme d'une relation de transition qui découle de cette interprétation.
Pour exprimer la capacité d'interaction des processus avec le monde extérieur, nous proposons une sémantique dénotationnelle, issue de la sémantique des phases de la logique linéaire, qui repose sur les notions de co-processus et d'interface. Un co-processus est un testeur qui caractérise un aspect de l'interaction possible du processus avec l'extérieur. Il est représenté par une formule qui vérifie une certaine forme de complémentarité avec la formule représentant le processus. Une interface est un ensemble de co-processus caractérisant de manière globale la capacité d'interaction d'un processus avec l'extérieur. Utilisant le cadre déductif, nous construisons un calcul sur les interfaces qui s'articule autour des concepts de réduction et de relativisation d'une interface.
La notion de communication synchrone est captée par une simple restriction sur la forme des preuves dans le système déductif de CPL. Cette restriction permet de définir un calcul synchrone, SCPL qui englobe notamment le π-calcul de Milner.
En tant que modèle logique du parallélisme, CPL peut être un outil d'analyse dans le domaine et peut servir de base à de nouvelles propositions de langages de programmation parallèle. On peut même d'emblée concevoir CPL comme un langage de programmation logique. Enfin, il peut constituer un cadre commun pour l'étude sémantique de programmes écrits dans des langages les plus divers.

Mots-clés: logique linéaire, déduction, programmation logique, parallélisme, communication, calcul de processus.