Thèse de Doctorat de l'Université Denis Diderot, Paris 7
Logique et Fondements de l'Informatique

Logique non-commutative et Programmation concurrente par contraintes

Paul Ruet

Directeur de thèse:
François Fages
Président du jury:
Jean-Yves Girard
Rapporteurs:
Yves Lafont
Vijay Saraswat

Soutenue le 23 octobre 1997

Jury:
Michele Abrusci
Pierre-Louis Curien
François Fages
Jean-Yves Girard
Jean-Louis Krivine
Yves Lafont
François Lamarche
Vijay Saraswat

à mes parents

Remerciements.

Je remercie François Fages d'avoir accepté de diriger cette thèse. Je lui suis reconnaissant pour ses encouragements et sa disponibilité, et pour les nombreuses discussions qui m'ont aidé tout au long de ce travail.

Je remercie Yves Lafont et Vijay Saraswat d'avoir lu cette thèse et contribué à la perfectionner par leurs remarques.

Je remercie Michele Abrusci, Pierre-Louis Curien, Jean-Yves Girard, Jean-Louis Krivine et François Lamarche d'avoir bien voulu participer au jury. Je remercie particulièrement vivement Michele Abrusci pour son invitation à travailler avec lui cet été à l'Université de Rome, Jean-Yves Girard pour ses critiques très pertinentes lors de visites à l'Institut de Mathématiques de Luminy, et François Lamarche pour avoir prété une oreille attentive à mes travaux lors de visites à l'équipe Calligramme de Nancy. Tous trois m'ont fait l'honneur de partager certaines de leurs recherches personnelles, et l'interaction a été pour moi très stimulante.

Je remercie Albert Burroni, François Métayer et Jacques Penon pour m'avoir communiquer leur goût pour les catégories dans le cadre de leur groupe de travail à l'Université Paris 7, Arnaud Fleury, Philippe de Groote, Christian Retoré et Jacques van de Wiele pour des échanges fructueux (au moins en ce qui me concerne) sur la logique linéaire non-commutative. Je remercie Sylvain Soliman pour son travail de DEA, complémentaire de cette thèse, sur les preuves de propriétés de programmes CC avec la sémantique des phases de la logique linéaire, Akim Demaille et Jean Leneutre pour leur lecture de cette thèse et leurs remarques qui m'ont aidé à l'améliorer, je leur souhaite une bonne continuation.

Je remercie le Laboratoire d'Informatique de l'Ecole Normale Supérieure, et tout particulièrement l'équipe Langages Logiques et Contraintes, pour m'avoir accordé un cadre de travail agréable.

Résumé.

Nous étudions des connections entre la logique et la programmation concurrente par contraintes (`cc`) dans le paradigme *programme = formule* et *calcul = recherche de preuve* qui est celui de la programmation logique, et nous montrons que la logique intuitionniste et la logique linéaire permettent de caractériser certaines observations intéressantes du comportement opérationnel des programmes (stores, succès), mais ne rendent pas bien compte des phénomènes de synchronisation.

Cela nous amène à introduire une version "mixte" de la logique linéaire, combinant des connecteurs commutatifs et des connecteurs non-commutatifs. Nous la présentons par un calcul des séquents classique (qui étend d'une part la logique linéaire commutative et d'autre part la logique linéaire non-commutative cyclique) et par les éléments de base d'une théorie de la démonstration: une sémantique des phases et l'élimination des coupures, des réseaux de preuves et un critère de séquentialisation.

La logique linéaire mixte permet de caractériser des observations fines: les suspensions d'un agent `cc`.

Abstract.

We study connections between logic and concurrent constraint programming (`cc`) in the *formulas as programs* and *proof search as computation* paradigm of logic programming, and we show that intuitionnistic logic and linear logic enable the characterization of some interesting observations on the computational behaviour of programs (stores, successes), but fail to give a precise account of synchronization phenomena.

This leads us to introduce a new "mixed" version of linear logic, which combines both commutative and non-commutative connectives. This logic is presented in terms of a classical sequent calculus (which extends both commutative linear logic and cyclic non-commutative linear logic) and the basic features of a proof theory: a phase semantics and cut elimination, proof nets and a sequentialization criterion.

Mixed linear logic enables the characterization of fine observations: the suspensions of a `cc` agent.

Chapter 1

Introduction

Programmation concurrente par contraintes.

En programmation logique par contraintes[1] (PLC), le comportement opérationnel des programmes est étroitement lié à la théorie de la preuve, à travers le paradigme *programme = formule* et *calcul = recherche de preuve*.

Le lien précis entre les programmes PLC et la logique est donné par des théorèmes de correction et de complétude de la résolution [22, 33, 54, 14]: par exemple, les "contraintes succès" d'un programme (aspect opérationnel) sont exactement les contraintes qui satisfont (en logique classique) la requête initiale.

De tels résultats facilitent l'écriture et la compréhension des programmes et fournissent des outils puissants, tant théoriques que pratiques, d'analyse des programmes.

Le cadre de la programmation concurrente par contraintes (`cc`), introduit par Vijay Saraswat [49], est une extension de la programmation logique par contraintes avec un mécanisme de synchronisation qui permet de voir les programmes comme des agents concurrents.

Les langages `cc` peuvent être présentés dans le style des algèbres de processus (CCS, π-calcul,...) par un ensemble d'opérateurs pour la composition parallèle $||$, le choix non-déterministe $+$, la localisation de variable $\exists$, la synchronisation $\rightarrow$, et par un système de transitions entre agents. Toutefois l'opérateur de synchronisation donne naissance à un mécanisme de suspension qui change la nature des succès et fait perdre leur caractérisation logique.

Dès lors il est naturel de chercher à comprendre dans quelle mesure le paradigme de la programmation logique peut être étendu aux langages `cc`, et dans quelle logique. Il s'agit donc d'interpréter les opérateurs de composition d'agents comme des connecteurs logiques et les règles de transitions comme des inférences de formules.

[1]Pour une introduction, voir par exemple [13], ainsi que [45, 27].

Partant des premiers résultats obtenus par Patrick Lincoln et Vijay Saraswat [31, 50], nous montrons dans cette thèse que la logique intuitionniste et la logique linéaire de Jean-Yves Girard [17] permettent de caractériser certaines observations intéressantes du comportement opérationnel des programmes (*stores*, *succès*), mais ne rendent pas bien compte des phénomènes de synchronisation. Plus précisément, les *suspensions* d'un programme ne sont pas caractérisables dans ces logiques, un obstacle en logique linéaire étant la règle $A \otimes (c \multimap B) \vdash c \multimap (A \otimes B)$.

Il manque un connecteur "séquentiel", c'est-à-dire non-commmutatif.

Logique non-commutative.

Les versions existantes de la logique linéaire non-commutative ne fournissent pas une solution immédiate: d'une part le connecteur *précède* $<$ de Christian Retoré [44] satisfait la même propriété que l'implication linéaire: $A \otimes (B < C) \vdash B < (A \otimes C)$, et d'autre part une logique purement non-commutative (comme le calcul de Lambek [30], sa version classique due à Michele Abrusci [1, 2], ou la logique circulaire de Girard et Yetter [18, 58]) ne permettrait pas d'interpréter l'opérateur de composition parallèle (qui est évidemment commutatif).

Cette difficulté nous conduit à introduire une nouvelle version "mixte" de la logique linéaire, où cohabitent des connecteurs multiplicatifs commutatifs ($\otimes$ et $\parr$) et des connecteurs (multiplicatifs) non-commutatifs ($\odot$ et $\lhd$).

Cette logique mixte a des connecteurs additifs, des exponentiels et des quantificateurs, mais la partie vraiment originale est évidemment le fragment multiplicatif, qui étend d'une part la LLM commutative ($\otimes$ et $\parr$) et d'autre part la LLM circulaire ($\odot$ et $\lhd$). Les connecteurs commutatifs et non-commutatifs ne sont pas indépendants: $A \otimes B \vdash A \odot B$ et duallement $A \lhd B \vdash A \parr B$.

Partant de deux travaux antérieurs (une version intuitionniste multiplicative de Philippe de Groote, et un calcul des séquents préliminaire classique avec des modalités [11, 47]) nous définissons un calcul des séquents classique où l'information sur la manière dont les formules doivent être combinées à l'intérieur d'un séquent ($\parr$ ou $\lhd$) est représentée par un ordre série-parallèle, ce qui nous permet d'adapter la sémantique des phases intuitionniste de [11].

Ce calcul fait appaître des règles structurelles réversibles explicites, et nous montrons que dans la version correspondante avec les règles structurelles réversibles implicites, les séquents ne sont pas des ordres (série-parallèles) comme dans le cas intuitionniste [11], mais des "ordres cycliques" portant sur des ensembles de formules. Ces ordres cycliques sont des relations ternaires qui généralisent les graphes orientés cycliques: un ordre cyclique R sur un ensemble E est cyclique ($R(x, y, z) \Rightarrow R(y, z, x) \Rightarrow R(z, x, y)$), et a la propriété que pour tout x dans E, la relation binaire $\langle R, x\rangle$ définie sur $E \setminus \{x\}$ par $\langle R, x\rangle(y, z)$ ssi $R(x, y, z)$ est un ordre, mais pas nécessairement total; R exprime donc une sorte d'"orientation" sur E.

Cette structure d'ordre cyclique se retrouve dans les voyages le long des réseaux de preuve (à la manière de [17]): nous définissons un critère de correction basé sur l'idée, due à Michele Abrusci, d'une troisième position d'interrupteur pour les liens *pars* non-commutatifs, et nous montrons un théorème de séquentialisation.

La version intuitionniste de cette logique linéaire mixte permet de rendre compte finement de la concurrence dans les langages `cc`: ainsi les suspensions d'un agent `cc` peuvent être caractérisées dans la logique.

Positionnement de la thèse par rapport aux autres connections entre programmation concurrente et logique linéaire.

Notre travail sur les `cc` se situe évidemment dans la continuité des travaux visant, dans d'autres paradigmes et avec d'autres approches, à établir des connections entre la logique linéaire et la programmation concurrente.

D'une part le caractère local de l'élimination des coupures dans les réseaux de preuve a inspiré de nouveaux langages de programmation parallèle: les réseaux d'interaction d'Yves Lafont [28].

D'autre part le degré de finesse des connecteurs linéaires permet de voir la logique linéaire comme une logique des actions, et a suggéré des correspondances dans le paradigme de la programmation logique, avec des approches différentes de la nôtre: ainsi des travaux d'Andreoli et Pareschi (qui soulignent que l'analogie 'calcul = recherche de preuve' pour la logique linéaire correspond à un langage réactif [3]), de Miller [37] (qui décrit une connection entre la logique linéaire et une version asynchrone, due à Boudol [7], du π-calcul de Milner [38]) de Kobayashi et Yonezawa [25, 26] et de Perrier [42].

On peut rattacher assez naturellement à cette dernière famille les travaux de Martí-Oliet et Meseguer [34], et Engberg et Winskel [12] sur la logique linéaire et les réseaux de Petri.

Chapter 2

Préliminaires: programmation concurrente par contraintes

La programmation concurrente par contraintes (`cc`, Saraswat [49]) est un modèle de programmation concurrente, où des *agents* (ou *processus*) communiquent par l'intermédiaire d'un *store* commun, c'est-à-dire un ensemble de contraintes sur les valeurs des variables en jeu, exprimant de l'information partielle sur les valeurs possibles de ces variables.

Les *contraintes* sont des formules du premier ordre, le langage des contraintes est supposé clos par conjonction et quantification existentielle.

Dans un programme `cc`, les agents sont supposés idéalement distribués (comme des molécules dans une solution chimique suffisamment bien agitée), de sorte que cela ait un sens de parler de *parallélisme* du calcul.

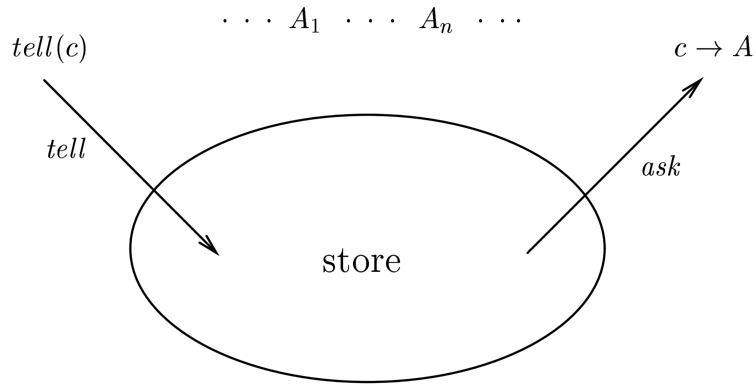
Les opérations élémentaires des agents sont:

- l'ajout d'une contrainte c dans le store (agent $tell(c)$),
- la suspension sur une certaine contrainte c: si A est un agent, l'agent $ask(c) \to A$ (ou plus simplement $c \to A$) attend, pour exécuter A, que le store contienne suffisamment d'information pour impliquer c.

Une étape élémentaire de calcul ne change pas la valeur des variables, mais peut interdire certaines valeurs qui étaient possibles auparavant ($tell(c)$), raffinant ainsi l'information partielle sur les variables libres.

C'est ce jeu de *tell* et *ask* qui confère à ce type de langages son caractère concurrent: pour que l'exécution de l'agent $d \to A$ soit possible, il faut qu'il y ait, via le store, communication avec d'autres agents présents dans la "solution" (par exemple un agent c tel que c implique d).

Notons que la communication est à sens unique: seul $d \to A$ reçoit de l'information, il n'y a pas vraiment d'échange. Autrement dit, les agents émetteurs (tell) ne sont pas bloquants (il n'y a pas d'opérateur explicite pour la séquentialité). De plus les agents peuvent rester inactifs. Ce sont deux caractéristiques de la communication *asynchrone* (voir par exemple [38, 7]).



Hérités de la tradition des *algèbres de processus*, les langages cc peuvent être présentés très simplement par un ensemble d'opérateurs pour la composition parallèle $\|$, le choix non-déterministe $+$, la localisation de variable $\exists$, la suspension $\to$, et par un système de transitions exprimant la sémantique opérationnelle des agents, dans le style SOS [51] ou dans le style de la machine chimique abstraite CHAM [5].

La programmation concurrente par contraintes peut aussi être présentée comme une extension de la *programmation logique par contraintes* [22, 33] avec un mécanisme de suspension $c \to A$, et la sémantique opérationnelle de cc est la même que celle de la programmation logique par contraintes, sauf pour $c \to A$ qui bloque jusqu'à ce que le store contienne assez d'information pour impliquer c, auquel cas $c \to A$ devient A.

Notons que par rapport aux langages de programmation logique, ce mécanisme répond naturellement au manque de structure de contrôle explicite, il est d'ailleurs présent, avec certaines restrictions, dans les premiers systèmes de PLC tels que CHIP [57, 21] et il est maintenant essentiel dans les applications de la programmation par contraintes à la modélisation de systèmes complexes et aux problèmes d'optimisation combinatoire [32].

Dans quelle mesure le paradigme de la programmation logique (programme = formule, exécution = preuve) peut être étendu aux cc, et dans quelle logique, tel est l'objet des chapitres 4 et 6.

2.1 Programmes cc monotones

2.1.1 Syntaxe

Définition 2.1.1 (Système de contraintes intuitionniste) *Un* système de contraintes *est un couple* $(\mathcal{C}, \Vdash_{\mathcal{C}})$*, où:*

- $\mathcal{C}$ *est un ensemble de formules (les* contraintes*) construites à partir d'un ensemble* V *de variables, d'un ensemble* Σ *de symboles de fonctions et de relations, et d'opérateurs logiques:* 1 *(vrai), la conjonction* $\wedge$ *et le quantificateur existentiel* $\exists$*;*

- $\Vdash$ *est une partie de* $\mathcal{C} \times \mathcal{C}$.

Au lieu de $(c,d) \in \Vdash_{\mathcal{C}}$, on écrira $c \Vdash_{\mathcal{C}} d$. $\Vdash_{\mathcal{C}}$ représente la "théorie" du système de contraintes, et exprime les relations de déduction entre contraintes.

Dans ce qui suit, $c, d, e \dots$ dénotent des contraintes.

$\vdash_{\mathcal{C}}$ est la plus petite relation $\subseteq \mathcal{C}^\star \times \mathcal{C}$ contenant $\Vdash_{\mathcal{C}}$ et close par les règles suivantes (vl(A) dénote l'ensemble des variables libres de A):

$$\Gamma, c \vdash c \qquad \frac{\Gamma, c \vdash d \qquad \Gamma \vdash c}{\Gamma \vdash d} \qquad \vdash 1 \qquad \frac{\Gamma \vdash c}{\Gamma, 1 \vdash c}$$

$$\frac{\Gamma, d, d \vdash c}{\Gamma, d \vdash c} \qquad \frac{\Gamma \vdash c}{\Gamma, d \vdash c} \qquad \frac{\Gamma \vdash c[t/x]}{\Gamma \vdash \exists x c} \qquad \frac{\Gamma, c \vdash d}{\Gamma, \exists x c \vdash d}\, x \notin vl(\Gamma, d)$$

$$\frac{\Gamma \vdash c_1 \qquad \Gamma \vdash c_2}{\Gamma \vdash c_1 \wedge c_2} \qquad \frac{\Gamma, c_1 \vdash c}{\Gamma, c_1 \wedge c_2 \vdash c} \qquad \frac{\Gamma, c_2 \vdash c}{\Gamma, c_1 \wedge c_2 \vdash c}$$

Ce sont les règles de la logique intuitionniste (LI) pour 1, $\wedge$ et $\exists$.
Toutefois le cadre de la logique intuitionniste (plutôt que celui de la logique classique) n'est pas essentiel, il est juste suffisant dans la mesure où les contraintes ne contiennent que des conjonctions et des quantifications existentielles.

Définition 2.1.2 (Agents) *La syntaxe des* agents `cc` *est donnée par la grammaire suivante:*

$$A ::= 1 \mid p(\vec{x}) \mid tell(c) \mid (A \parallel A) \mid A + A \mid \exists x A \mid c \rightarrow A$$

où 1 *représente l'inaction,* $\parallel$ *est l'opérateur de composition parallèle,* $+$ *l'opérateur de choix non-déterministe,* $\exists$ *l'opérateur de localisation de variable et* $\rightarrow$ *l'opérateur de suspension. Les agents atomiques* $p(\vec{x})\dots$ *sont appelés* noms de processus *ou* noms de procédure.

La récursion est obtenue par des déclarations:

Définition 2.1.3 (Dclarations) *La syntaxe des* déclarations *est donnée par la grammaire suivante:*

$$D ::= \epsilon \mid p(\vec{x}) = A \mid D, D$$

On fait l'hypothèse, très naturelle, que dans une déclaration $p(\vec{x}) = A$, toutes les variables libres de A ont une occurrence libre dans p, et que dans une relation $c_1 \dots c_n \Vdash_{\mathcal{C}} c$, toutes les variables libres de c ont une occurrence libre dans $c_1 \dots c_n$. Notons que c'est bien le sens qu'on donne aux clauses de Horn dans les langages de programmation logique, par exemple: les variables qui sont libres dans le corps mais pas dans la tête sont considérées (implicitement dans la syntaxe, explicitement dans la sémantique) comme quantifiées existentiellement dans le corps (car universellement dans la clause).

La sémantique opérationnelle que nous allons définir porte sur des configurations o l'on distingue explicitement le store des agents:

Définition 2.1.4 (Configurations) *Une* configuration *est un triplet* $(\vec{x}; c; \Gamma)$, *o* $\vec{x}$ *est un ensemble de variables,* c *est une contrainte, et* Γ *un multi-ensemble d'agents.*

Dans une configuration $(\vec{x}; c; \Gamma)$, $\vec{x}$ est l'ensemble des variables locales créées lors d'une exécution et c est le store.

On note $\mathcal{A}$ (resp. $\mathcal{B}$) l'ensemble des agents (resp. configurations).

La sémantique opérationnelle est définie par un système de transitions qui fait abstraction de stratégies d'évaluation spécifiques. Le système de transitions est donné dans le style de la CHAM [5] (voir aussi [43]). Cette présentation a l'avantage de garder toujours explicites les quantifications de variables. On distingue une relation de congruence (entre configurations) de la relation de transitions (entre configurations) proprement dite:

Définition 2.1.5 *La relation de* congruence structurelle $\equiv$ *est la plus petite congruence telle que:*

Composition parallèle	$(\vec{x}; c; A \parallel B, \Gamma) \equiv (\vec{x}; c; A, B, \Gamma)$	
Inaction	$(\vec{x}; c; 1, \Gamma) \equiv (\vec{x}; c; \Gamma)$	
α***-Conversion***	$\dfrac{y \notin vl(c, \Gamma)}{(\vec{x}; c; \exists y A, \Gamma) \equiv (\vec{x}, y; c; A, \Gamma)}$	
Variables locales	$\dfrac{y \notin vl(c, \Gamma)}{(\vec{x}; c; \exists y A, \Gamma) \equiv (\vec{x}, y; c; A, \Gamma)}$	$\dfrac{y \notin vl(c, \Gamma)}{(\vec{x}, y; c; \Gamma) \equiv (\vec{x}; c; \Gamma)}$

La relation de transition $\longrightarrow$ *est la plus petite relation transitive telle que:*

Tell	$(\vec{x}; c; tell(d), \Gamma) \longrightarrow (\vec{x}; c \wedge d; \Gamma)$
Ask	$\dfrac{c \vdash_{\mathcal{C}} d}{(\vec{x}; c; d \rightarrow A, \Gamma) \longrightarrow (\vec{x}; c; A, \Gamma)}$
Déclarations	$\dfrac{(p(\vec{y}) = A) \in P}{(\vec{x}; c; p(\vec{y}), \Gamma) \longrightarrow (\vec{x}; c; A, \Gamma)}$
$\equiv$	$\dfrac{(\vec{x}; c; \Gamma) \equiv (\vec{y}; d; \Delta)}{(\vec{x}; c; \Gamma) \longrightarrow (\vec{y}; d; \Delta)}$
Non-déterminisme (choix aveugle)	$(\vec{x}; c; A + B, \Gamma) \longrightarrow (\vec{x}; c; A, \Gamma)$ $(\vec{x}; c; A + B, \Gamma) \longrightarrow (\vec{x}; c; B, \Gamma)$

Comme d'habitude, la sémantique opérationnelle précise dépend du choix des observables. Nous allons considérer les stores, les succès et les suspensions:

Définition 2.1.6 (Observables) *Le* store *d'une configuration* $(\vec{x}; c; \Gamma)$ *est la contrainte* $\exists \vec{x} c$. *On dit que* $\exists \vec{x} c$ *est un store accessible partir de l'agent* A *s'il existe un multi-ensemble de formules* Γ *tel que* $(\emptyset; 1; A) \longrightarrow (\vec{x}; c; \Gamma)$.

Un succès *pour un agent* A *est un contrainte* c *telle que* $(\emptyset; 1; A) \longrightarrow (\emptyset; c; 1)$*; un succès est donc un store seul, sans agent actif, aucune transition n'est plus possible.*

Une suspension *pour* A *est une configuration* $(\vec{x}; c; d_1 \rightarrow A_1, \ldots, d_n \rightarrow A_n)$ *telle que* $(\emptyset; 1; A) \longrightarrow (\vec{x}; c; d_1 \rightarrow A_1, \ldots, d_n \rightarrow A_n)$ *et pour aucun* i, $c \vdash_{\mathcal{C}} d_i$.

Remarques:

▶ Dans une configuration $(\vec{x}; c; \Gamma)$, Γ est un multi-ensemble. Donc la règle pour la composition parallèle implique la commutativité et l'associativité de l'opérateur $||$.

De mme, $\vec{x}$ étant un ensemble de variables, on a par exemple $(\vec{x}; c; \exists y \exists z A) \equiv (\vec{x}; c; \exists z \exists y A)$.

▶ Une propriété de ce calcul est que l'exécution est *monotone*: les informations contenues dans le store ne sont pas consommées par la règle de communication (*ask*), autrement dit:

Proposition 2.1.7 (Monotonie [52]) – *Si* $(\vec{x}; c; \Gamma) \longrightarrow (\vec{y}; d; \Delta)$ *alors* $\exists \vec{y} d \vdash_{\mathcal{C}} \exists \vec{x} c$.

– *Si* $(\vec{x}; c; \Gamma) \longrightarrow (\vec{y}; d; \Delta)$, *alors pour tout multi-ensemble d'agents* Σ *et toute contrainte* e, $(\vec{x}; c \wedge e; \Gamma, \Sigma) \longrightarrow (\vec{y}; d \wedge e; \Delta, \Sigma)$.

Ces proprits permettent de donner à `cc` une sémantique dénotationnelle, où les agents sont vus comme des opérateurs de clôture sur le semi-treillis des contraintes [52, 23].

Des variantes non-monotones des `cc`, où les contraintes sont consommables, ont été introduites par Saraswat et Lincoln [50], puis étudiées dans [6, 55]. Nous prsenterons un tel calcul non-monotone en Section 2.2.

▶ La relation $\longrightarrow$ est réflexive (car $\equiv$ l'est et on utilise la règle de transition $\equiv$): cela reflète le caractère asynchrone du calcul: un agent peut rester silencieux, indpendemment des autres.

▶ L'agent non-déterminisme $A + B$, peut se comporter comme A ou comme B. L'opérateur de choix non-déterminisme $+$, appelé *choix aveugle*, est donc différent du *choix gardé "one-step"* défini par

$$\frac{A \longrightarrow A'}{A + B \longrightarrow A'} \quad \text{et} \quad \frac{B \longrightarrow B'}{A + B \longrightarrow B'} \quad .$$

Comme il est remarqué dans [23, 16], la différence entre le non-déterminisme (angélique, avec backtracking) et l'indéterminisme ("committed-choice") vient de la façon dont sont définies les observations, plutôt que de la manière dont les transitions sont définies, mais le non-déterminisme angélique fait en général référence à la règle pour le choix aveugle. Pour des raisons de simplicité nous n'avons pas considéré explicitement ici les deux formes de non-déterminisme. Nous reviendrons brièvement sur cette distinction et sur la sémantique logique de l'indéterminisme dans le Chapitre 6. Une étude plus approfondie dépasse le cadre de cette thèse.

▶ Les agents et déclarations ne contenant pas $+$ sont dits *déterministes*. Cette appellation est justifie par la proposition suivante:

Proposition 2.1.8 (Confluence [52]) *Pour toute configuration déterministe κ (en l'absence des rgles non-dterministes), si $\kappa \longrightarrow \kappa_1$ et $\kappa \longrightarrow \kappa_2$, alors il existe une configuration déterministe κ' telle que $\kappa_1 \longrightarrow \kappa'$ et $\kappa_2 \longrightarrow \kappa'$.*

▶ Les programmes `cc` sont paramétrés par un système de contraintes, qu'en général nous ne mentionnerons pas explicitement dans le système de transitions, ni même dans la relation de déduction entre contraintes, sauf si le contexte peut prêter à confusion.

2.1.2 Exemple

Dans le problème classique des n reines (placement de n reines sans prise sur un échiquier $n \times n$), la concurrence apparaît dans l'utilisation d'un test d'implication de contraintes, qui s'écrit naturellement avec l'opérateur de suspension des langages `cc`.

Les contraintes que l'on considère sont les formules de l'arithmétique formées avec $\wedge$ et $\exists$, mais pour des questions de décidabilité, on se limite à des variables prenant leurs valeurs dans des intervalles bornés.

```
reines(N,L) = ∃Colonne
      liste(N,Colonne) ||
      domain(Colonne,[1,N]) ||
      sure(Colonne) ||
      labeling(Colonne).

sure(M) = ∃X ∃L
      M=[]
       +
      (M=[X|L] ||
      non_attaque(X,L) ||
      sure(L)).

non_attaque(X,L) =
      non_attaque(X,L,1).

non_attaque(X,L,N) = ∃Y ∃L' ∃N'
      L=[]
       +
      (L=[Y|L'] ||
      tell(X≠Y) ||
      tell(X≠Y+N) ||
      tell(X≠Y-N) ||
      tell(N'=N+1) ||
      non_attaque(X,L',N')).
```

La concurrence apparaît à un autre niveau. En effet il est intéressant de modéliser ce problème de manière redondante (une liste des positions des reines par ligne et une liste des positions des reines par colonne): l'utilisation concurrente des deux modèles permet en effet de détecter certains échecs plus tôt, et d'améliorer la stabilité et la robustesse des performances [24].

```
mm-reines(N,L) = ∃Ligne ∃Colonne
      liste(N,Colonne) ||
      domain(Colonne,[1,N]) ||
      sure(Colonne) ||
```

```
        liste(N,Ligne) ||
        domain(Ligne,[1,N]) ||
        sure(Ligne) ||
        liaison(Ligne,Colonne) ||
        append(Ligne,Colonne,L) ||
        labeling(L).

liaison(L,C) = liaison(L,1,C).

liaison(L,I,C) = ∃X ∃L' ∃I'
        L=[]
         +
        (L=[X|L'] ||
        equivalence(X,I,C,1) ||
        tell(I'=I+1) ||
        liaison(L',I',C)).

equivalence(X,I,C,J) = ∃Y ∃L ∃J'
        C=[]
         +
        (C=[Y|L] ||
        X=J → tell(Y=I) ||
        X≠J → tell(Y≠I) ||
        Y=I → tell(X=J) ||
        Y≠I → tell(X≠J) ||
        tell(J'=J+1) ||
        equivalence(X,I,L,J')).
```

2.2 Programmes `cc` non-monotones

Des variantes non-monotones des `cc`, où les contraintes sont consommables, ont été introduites par Saraswat et Lincoln [50], puis étudiées dans [6, 55, 8].

Ces versions non-monotones sont naturellement plus proches des calculs de processus comme CCS ou le π-calcul de Milner.

Dans cette section nous présentons une version non-monotone, `lcc`, et nous donnons une traduction des `cc` monotones dans `lcc` qui respecte le système de transition, de sorte que `lcc` est un raffinement de `cc`, et les caractérisations logiques que nous ferons du comportement opérationnel de `lcc` vaudront aussi pour `cc`.

2.2.1 Syntaxe

Comme pour les `cc` monotones, nous définissons le système de contraintes, les agents, les configurations et le système de transition.

Définition 2.2.1 (Système de contraintes linéaire) *Un* système de contraintes linaire *est un couple* $(\mathcal{C}, \Vdash_{\mathcal{C}})$*, où:*

- $\mathcal{C}$ *est un ensemble de formules (les* contraintes linaires*) construites à partir d'un ensemble* V *de variables, d'un ensemble* Σ *de symboles de fonctions et de relations, et d'opérateurs logiques:* 1*, la conjonction multiplicative* $\otimes$*, le quantificateur existentiel* $\exists$ *et le connecteur exponentiel* $!$*;*
- $\Vdash_{\mathcal{C}} \subseteq \mathcal{C} \times \mathcal{C}$*.* 1 *est l'élément neutre de* $\otimes$*.*

Au lieu de $((c,d) \in \Vdash_{\mathcal{C}}$, on écrira $c \Vdash_{\mathcal{C}} d$.

$\vdash_{\mathcal{C}}$ est la plus petite relation $\subseteq \mathcal{C}^\star \times \mathcal{C}$ contenant $\Vdash_{\mathcal{C}}$ et close par les règles suivantes (vl(A) dénote l'ensemble des variables libres de A):

$$c \vdash c \qquad \frac{\Gamma, c \vdash d \qquad \Delta \vdash c}{\Gamma, \Delta \vdash d} \qquad \vdash 1 \qquad \frac{\Gamma \vdash c}{\Gamma, 1 \vdash c}$$

$$\frac{\Gamma, c_1, c_2 \vdash c}{\Gamma, c_1 \otimes c_2 \vdash c} \qquad \frac{\Gamma \vdash c_1 \qquad \Delta \vdash c_2}{\Gamma, \Delta \vdash c_1 \otimes c_2} \qquad \frac{\Gamma \vdash c[t/x]}{\Gamma \vdash \exists x c} \qquad \frac{\Gamma, c \vdash d}{\Gamma, \exists x c \vdash d}\ x \notin vl(\Gamma, d)$$

$$\frac{\Gamma, c \vdash d}{\Gamma, !c \vdash d} \qquad \frac{!\Gamma \vdash d}{!\Gamma \vdash !d} \qquad \frac{\Gamma \vdash d}{\Gamma, !c \vdash d} \qquad \frac{\Gamma, !c, !c \vdash d}{\Gamma, !c \vdash d}$$

Ce sont les règles de la logique linéaire intuitionniste (LLI) pour 1, $\otimes$, $\exists$ et $!$ (pour une introduction à la logique linéaire, voir le Chapitre 3).

Définition 2.2.2 (Agents) *La syntaxe des* agents `lcc` *est la même que dans le cas monotone:*

$$A ::= 1 \mid p(\vec{x}) \mid tell(c) \mid (A \parallel A) \mid A + A \mid \exists x A \mid c \to A$$

La récursion est obtenue par des déclarations:

Définition 2.2.3 (Dclarations) *La syntaxe des* déclarations *est donnée par la grammaire suivante:*

$$D ::= \epsilon \mid p(\vec{x}) = A \mid D, D$$

On fait encore l'hypothèse, très naturelle, que dans une déclaration $p(\vec{x}) = A$, toutes les variables libres de A ont une occurrence libre dans p, et que dans une relation $c_1 \ldots c_n \Vdash_{\mathcal{C}} c$, toutes les variables libres de c ont une occurrence libre dans $c_1 \ldots c_n$.

Définition 2.2.4 (Configurations) *Une* configuration *est un triplet* $(\vec{x}; c; \Gamma)$, *o* $\vec{x}$ *est un ensemble de variables,* c *est une contrainte, et* Γ *un multi-ensemble d'agents.*

Dans une configuration $(\vec{x}; c; \Gamma)$, $\vec{x}$ est l'ensemble des variables locales créées lors d'une exécution et c est le store.

On note $\mathcal{A}$ (resp. $\mathcal{B}$) l'ensemble des agents (resp. configurations).

Définition 2.2.5 *La relation de* congruence structurelle $\equiv$ *est la plus petite congruence telle que:*

Composition parallèle	$(\vec{x}; c; A \parallel B, \Gamma) \equiv (\vec{x}; c; A, B, \Gamma)$	
Inaction	$(\vec{x}; c; 1, \Gamma) \equiv (\vec{x}; c; \Gamma)$	
α***-Conversion***	$\dfrac{z \notin vl(c, \Gamma)}{(\vec{x}, y; c; \Gamma) \equiv (\vec{x}, z; c[z/y]; \Gamma[z/y])}$	
Variables locales	$\dfrac{y \notin vl(c, \Gamma)}{(\vec{x}; c; \exists y A, \Gamma) \equiv (\vec{x}, y; c; A, \Gamma)}$	$\dfrac{y \notin vl(c, \Gamma)}{(\vec{x}, y; c; \Gamma) \equiv (\vec{x}; c; \Gamma)}$

La relation de transition $\longrightarrow$ *est la plus petite relation transitive telle que:*

Tell	$(\vec{x}; c; tell(d), \Gamma) \longrightarrow (\vec{x}; c \otimes d; \Gamma)$
Ask	$\dfrac{c \vdash_{\mathcal{C}} d \otimes e}{(\vec{x}; c; e \to A, \Gamma) \longrightarrow (\vec{x}; d; A, \Gamma)}$
Déclarations	$\dfrac{(p(\vec{y}) = A) \in P}{(\vec{x}; c; p(\vec{y}), \Gamma) \longrightarrow (\vec{x}; c; A, \Gamma)}$
$\equiv$	$\dfrac{(\vec{x}; c; \Gamma) \equiv (\vec{y}; d; \Delta)}{(\vec{x}; c; \Gamma) \longrightarrow (\vec{y}; d; \Delta)}$
Non-déterminisme (choix aveugle)	$(\vec{x}; c; A + B, \Gamma) \longrightarrow (\vec{x}; c; A, \Gamma)$ $(\vec{x}; c; A + B, \Gamma) \longrightarrow (\vec{x}; c; B, \Gamma)$

Définition 2.2.6 (Observables) *Le* stores *et les* succès *sont dfinis comme dans le cas monotone.*

Du fait que les contraintes sont des formules linéaires, on doit modifier légèrement la définition des suspensions: on dira qu'une suspension *pour A est une configuration $(\vec{x}; c; d_1 \to A_1, \ldots, d_n \to A_n)$ telle que $(\emptyset; 1; A) \longrightarrow (\vec{x}; c; d_1 \to A_1, \ldots, d_n \to A_n)$ et pour aucun i, $c > d_i$, où la relation entre contraintes "$c > d$" est la plus petite relation contenant $\vdash_{\mathcal{C}}$ et close par la règle $\forall e \in \mathcal{C}(c > d \ \Rightarrow \ (c \otimes e) > d)$.*

On dira qu'un agent A suspend avec le store c sur les contraintes $d_1, ..., d_n$, *s'il existe une suspension pour A de la forme $(\vec{x}; c; d_1 \to A_1, \ldots, d_n \to A_n)$.*

Remarque:

▶ Les seules diffrences par rapport aux `cc` monotones sont donc le fait que les contraintes sont des formules de logique linaire et la rgle de communication *ask* qui consomme de l'information. Le calcul est donc intrinsquement non-dterministe, mme en l'absence de l'opérateur de choix +, puisque plusieurs contraintes peuvent satisfaire la condition de la règle.

2.2.2 Traduction de `cc` dans `lcc`

Les langages `lcc` sont un raffinement des `cc` habituels, et en effet le caractère monotone des `cc` peut être retrouvé simplement en utilisant le connecteur exponentiel ! de la logique linéaire, qui autorise la duplication d'hypothèses et permet donc d'éviter la consommation des contraintes lors d'une application de la règle *ask*:

Définition 2.2.7 *Soit $(\mathcal{C}, \Vdash_{\mathcal{C}})$ un système de contraintes. On définit la* traduction *de $(\mathcal{C}, \Vdash_{\mathcal{C}})$, qui est un systme de contraintes linéaires $(\mathcal{C}, \Vdash_{\mathcal{C}})^\circ$ de la manière suivante, en même temps que la traduction des agents `cc` en agents `lcc`:*

$$\begin{array}{rr}
\multicolumn{2}{c}{c^\circ = !c, \text{ si } c \text{ est une contrainte atomique}} \\
(c \wedge d)^\circ = c^\circ \otimes d^\circ & (\exists x c)^\circ = \exists x c^\circ \\
tell(c)^\circ = tell(c^\circ) & p(\vec{x})^\circ = p(\vec{x}) \\
(A \parallel B)^\circ = A^\circ \parallel B^\circ & (A + B)^\circ = A^\circ + B^\circ \\
(c \to A)^\circ = c^\circ \to A^\circ & (\exists x A)^\circ = \exists x A^\circ
\end{array}$$

La relation de déduction $\Vdash_{\mathcal{C}}^\circ$ est définie par: $c^\circ \Vdash_{\mathcal{C}}^\circ d^\circ$ ssi $c \Vdash_{\mathcal{C}} d$.

La relation ${\vdash_{\mathcal{C}}}^\circ$ est obtenue partir de ${\Vdash_{\mathcal{C}}}^\circ$ par les rgles de la logique linaire pour 1, !, $\otimes$ *et* $\exists$.

La traduction d'une configuration `cc` $(\vec{x}; c; \Gamma)$ est la configuration `lcc` $(\vec{x}; c^\circ; \Gamma^\circ)$.

La relation de transition $\longrightarrow^\circ$ est celle de `lcc`.

Remarque:
▶ Nous avons choisi de limiter l'usage de ! aux seules contraintes: en effet, habituellement, l'opérateur de réplication dans les calculs de processus (comme dans le π-calcul [38], où il est noté aussi !) n'a pas le même comportement que le connecteur exponentiel: il permet la duplication ($!A \longrightarrow (!A \,\|\, !A)$) mais pas l'effacement ($!A \not\twoheadrightarrow 1$).

Pour les contraintes, la traduction ci-dessus est une traduction bien connue de la logique intuitionniste en logique linéaire [17, p.81], d'où:

Proposition 2.2.8 *Soient c et d des contraintes: $c \vdash_{\mathcal{C}} d$ ssi $c^\circ \vdash_{\mathcal{C}^\circ} d^\circ$.*

Maintenant, on vérifie que les traductions de configurations ont bien le comportement monotone attendu:

Proposition 2.2.9 *Soient $(\vec{x}; c; \Gamma)$ et $(\vec{y}; d; \Delta)$ des configurations* `cc`*:*
(i) $(\vec{x}; c; \Gamma) \equiv (\vec{y}; d; \Delta)$ ssi $(\vec{x}; c^\circ; \Gamma^\circ) \equiv^\circ (\vec{y}; d^\circ; \Delta^\circ)$;
(ii) si $(\vec{x}; c; \Gamma) \longrightarrow (\vec{y}; d; \Delta)$ alors $(\vec{x}; c^\circ; \Gamma^\circ) \longrightarrow^\circ (\vec{y}; d^\circ; \Delta^\circ)$;
(iii) si $(\vec{x}; c^\circ; \Gamma^\circ) \longrightarrow^\circ (\vec{y}; d^\circ; \Delta^\circ)$ alors $(\vec{x}; c; \Gamma) \longrightarrow (\vec{y}; e; \Delta)$, avec $e \vdash d$.

Preuve. (i) est évident.

Pour (ii), on procde par induction sur $\longrightarrow$, le seul cas intéressant étant la règle *ask*: on suppose

$$(\vec{x}; c; d \to A, \Gamma) \longrightarrow (\vec{x}; c; A, \Gamma),$$

utilisant la relation $c \vdash_{\mathcal{C}} d$. On a donc $c \vdash_{\mathcal{C}} c \wedge d$, et donc d'après la Proposition 2.2.8, $c^\circ \vdash_{\mathcal{C}^\circ} (c \wedge d)^\circ = c^\circ \otimes d^\circ$. Par conséquent

$$(\vec{x}; c^\circ; d^\circ \to A^\circ, \Gamma^\circ) \longrightarrow^\circ (\vec{x}; c^\circ; A^\circ, \Gamma^\circ),$$

cqfd.

Pour (iii), on procde par induction sur $\longrightarrow^\circ$. De nouveau le seul cas intressant est la rgle *ask*: on suppose

$$(\vec{x}; c^\circ; d^\circ \to A^\circ, \Gamma^\circ) \longrightarrow^\circ (\vec{x}; e^\circ; A^\circ, \Gamma^\circ),$$

utilisant la relation $c^\circ \vdash_{\mathcal{C}^\circ} d^\circ \otimes e^\circ = (d \wedge e)^\circ$. Donc d'après la Proposition 2.2.8, $c \vdash_{\mathcal{C}} d \wedge e \vdash_{\mathcal{C}} d$, d'où

$$(\vec{x}; c; d \to A, \Gamma) \longrightarrow (\vec{x}; c; A, \Gamma),$$

et $c \vdash_{\mathcal{C}} e$, cqfd. ■

Donc la traduction ci-dessus respecte l'observation des stores et des succès d'une configuration `cc` (cas (i) et (ii) de la Proposition 2.2.9).

Le cas (iii) nous conduira, dans le Chapitre 6, à raffiner cette traduction, pour nous limiter à caractériser une classe particulière de suspensions (qui correspond à un certain choix de contraintes, naturel dans les langages de spécification de protocoles).

2.2.3 Exemple

Un problème classique d'expressivité des langages concurrents est celui du dîner des philosophes: n philosophes sont assis à une table ronde pour manger, chaque philosophe a une fourchette à sa droite (donc aussi une à sa gauche), et a besoin de deux fourchettes pour manger (la version avec baguettes est peut-être plus réaliste).

Comme le propose [6], ce problème a une solution distribuée extrêmement simple dans `lcc`.

Ici les contraintes considérées sont atomiques: soit $\mathtt{fourchette}_i$, soit $\mathtt{mange}_i$ $(1 \leq i \leq n)$, soit `ticket`.

$$
\begin{array}{l}
\mathtt{philosophe}_i\ \mathtt{=} \\
\quad\quad \mathtt{ticket} \rightarrow \mathtt{fourchette}_i \rightarrow \mathtt{fourchette}_{i+1 \bmod n} \rightarrow \\
\quad\quad\quad\quad \mathtt{(tell(mange}_i\mathtt{)} \parallel \\
\quad\quad\quad\quad \mathtt{mange}_i \rightarrow \\
\quad\quad\quad\quad\quad\quad \mathtt{(tell(fourchette}_i\mathtt{)} \parallel \mathtt{tell(fourchette}_{i+1 \bmod n}\mathtt{)} \parallel \\
\quad\quad\quad\quad\quad\quad \mathtt{tell(ticket)} \parallel \mathtt{philosophe}_i\mathtt{))}.
\end{array}
$$

$$
\begin{array}{l}
\mathtt{init\ =\ } \underbrace{\mathtt{tell(ticket)} \parallel \cdots \parallel \mathtt{tell(ticket)}}_{n-1\ \ fois} \parallel \\
\quad\quad \mathtt{tell(fourchette}_1\mathtt{)} \parallel \cdots \parallel \mathtt{tell(fourchette}_n\mathtt{)} \parallel \\
\quad\quad \mathtt{philosophe}_1 \parallel \cdots \parallel \mathtt{philosophe}_n.
\end{array}
$$

Ce programme est sûr et vivace: deux philosophes consécutifs ne peuvent pas manger en même temps, et au moins un philosophe peut manger.

On verra dans le Chapitre 4 (Section 4.2.2) une preuve de sûreté de ce programme utilisant la sémantique des phases de la logique linéaire.

Chapter 3

Préliminaires: logique linéaire

Ce chapitre est une courte introduction à la logique linéaire.

La logique linéaire, introduite par Girard [17] (voir aussi [20]), est un raffinement de la logique usuelle ayant le caractère constructif de la logique intuitionniste (normalisation forte de l'élimination des coupures) et la symétrie de la logique classique (négation involutive).

En logique classique ou intuitionniste, les vérités sont définitives: si vous avez A et $A \Rightarrow B$ alors vous pouvez déduire B, mais vous avez toujours A.

En logique linéaire, les formules ne sont pas systématiquement réutilisables: si on note $\multimap$ l'implication linéaire, alors de A et $A \multimap B$ vous pouvez déduire B, mais vous n'avez plus A. Autrement dit le processus de déduction "consomme" des hypothèses, comme des molécules sont consommées dans une réaction chimique.

En termes logiques, les règles de déduction responsables, en logique classique ou intuitionniste, de la pernnité des hypothèses sont les règles (dites *structurelles*) d'*affaiblissement* et de *contraction*:

$$\frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta} \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash A, \Delta}$$

$$\frac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta} \qquad \frac{\Gamma \vdash A, A, \Delta}{\Gamma \vdash A, \Delta}$$

qui permettent respectivement l'effacement et la duplication d'hypothèses (à gauche) et de conclusions (à droite).

En logique linéaire, ces règles ne sont plus systématiquement applicables. Cette nouveauté amène naturellement à faire la distinction entre deux types de connecteurs binaires:

- les **multiplicatifs**: conjonction $\otimes$ (*tenseur*) et disjonction $⅋$ (*par*),

- les **additifs**: conjonction & (*avec*) et disjonction $\oplus$ (*plus*).

Les deux conjonctions correspondent à deux manières de dire "et": une réaction chimique $2H_2 + O_2 \longrightarrow H_2O$, consomme deux H_2 "et" un O_2, et il n'y a pas de réaction $2H_2 + O_2 \longrightarrow O_2$ par exemple, qui "oublierait" les $H_2 \ldots$, et de même en logique linéaire, de $A \otimes B$ on ne peut pas déduire A; quant au &, il exprime la possibilité de deux actions, ainsi de $A\&B$ on peut déduire A "et" on peut déduire B, mais il faut choisir.

Le pouvoir expressif de la logique usuelle est retrouvé avec des nouveaux connecteurs unaires:

- les **exponentiels**: ! (*bien sûr*) et son dual ? (*pourquoi pas*),

qui font le "pont" entre les multiplicatifs et les additifs via les isomorphismes $!A \otimes !B \cong !(A\&B)$ et $?A \parr ?B \cong ?(A \oplus B)$.

3.1 Séquents

Les formules sont construites à partir de littéraux $p, q, \ldots p^\perp, q^\perp, \ldots$ et des constantes $1, \perp, \top, 0$ avec les connecteurs $\otimes, \parr, \&, \oplus$ (binaires), $!, ?$ (unaires) et les quantificateurs $\forall$ et $\exists$. La négation est définie par:

$$\begin{array}{rl} (p)^\perp = p^\perp & (p^\perp)^\perp = p \\ 1^\perp = \perp & \perp^\perp = 1 \\ \top^\perp = 0 & 0^\perp = \top \end{array}$$

et des règles de De Morgan qui expriment la dualité entre $\otimes$ et $\parr$, & et $\oplus$, ! et ?, $\forall$ et $\exists$.

La négation est donc une involution.

L'implication est un connecteur défini:

$$A \multimap B = A^\perp \parr B$$

Définition 3.1.1 *Un* séquent *est un jugement de la forme* $\vdash \Gamma$*, où* Γ *est un multi-ensemble de formules.*

Notation:

▶ Les symboles $\Gamma, \Delta \ldots$ désigneront des multi-ensembles de formules.

Le calcul des séquents est donné par les règles suivantes (les permutations de formules sont donc implicites):

Axiome / Coupure

$$\vdash A^\perp, A \qquad\qquad \frac{\vdash \Gamma, A \qquad \vdash A^\perp, \Delta}{\vdash \Gamma, \Delta}$$

Multiplicatifs

$$\vdash 1 \qquad\qquad \frac{\vdash \Gamma}{\vdash \Gamma, \perp}$$

$$\frac{\vdash \Gamma, A \qquad \vdash B, \Delta}{\vdash \Gamma, A \otimes B, \Delta} \qquad\qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}$$

Additifs

$$\vdash \Gamma, \top \qquad\qquad \text{(pas de règle pour 0)}$$

$$\frac{\vdash \Gamma, A \qquad \vdash \Gamma, B}{\vdash \Gamma, A \& B} \qquad\qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, A \oplus B} \qquad\qquad \frac{\vdash \Gamma, B}{\vdash \Gamma, A \oplus B}$$

Exponentiels

$$\frac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\text{ promotion} \qquad\qquad \frac{\vdash \Gamma}{\vdash \Gamma, ?A}\text{ affaiblissement}$$

$$\frac{\vdash \Gamma, A}{\vdash \Gamma, ?A}\text{ déréliction} \qquad\qquad \frac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A}\text{ contraction}$$

Quantificateurs

$$\frac{\vdash \Gamma, A}{\vdash \Gamma, \forall x A}\; x \notin vl(\Gamma) \qquad\qquad \frac{\vdash \Gamma, A[t/x]}{\vdash \Gamma, \exists x A}$$

Un séquent $\vdash A_1, \ldots, A_n$ s'interprète comme la formule $A_1 \parr \cdots \parr A_n$. Pour tout $i \in \{1 \ldots n\}$ on peut la lire $A_1^\perp \otimes \cdots A_{i-1}^\perp \otimes A_{i+1}^\perp \cdots \otimes A_n^\perp$ implique A_i.

Les exponentiels permettent de retrouver l'expressivité de la logique classique, c'est-à-dire qu'il existe des traductions de la logique classique en logique linéaire, correctes et complètes pour la prouvabilité. (Mais l'intéret des ces traductions est surtout qu'elles offrent une analyse fine de l'élimination des coupures dans le calcul des séquents classique, qui, comme on le sait, ne conflue pas et ne termine pas en général. Voir [19, 10]).

3.2 Sémantique

La logique linéaire a deux sémantiques:

1. une sémantique de la prouvabilité: la sémantique des *phases*,
2. une sémantique des preuves: la sémantique *cohérente*, qui est un raffinement des sémantiques de Scott pour la logique intuitionniste.

Nous nous limitons à une rapide présentation de la sémantique des phases, que nous adapterons à la logique mixte dans le Chapitre 5.

Définition 3.2.1 (Espace de phases) *Un* espace de phases *consiste en la donnée:*
– d'un monoïde $(P, *, 1)$, dont les éléments sont appelés phases,
– d'une partie $\bot_P$ de P, dont les éléments sont appelés les antiphases.

Un espace de phases enrichi *consiste en un espace de phases P et un sous-monoïde K de $J(P) = \{x \in \mathbf{1}, x \in \{x \star x\}^{\bot\bot}\}$.*

On notera aussi P l'espace de phase (enrichi).

Définition 3.2.2 *Si G est une partie de P, son* dual *est défini par:*

$$G^\bot = \{p \in P \mid \forall q \in G, p \star q \in \bot_P\}$$

Si G et H sont des parties de P, on définit:

$$G \star H = \{p \star q \mid p \in G, q \in H\}.$$

Définition 3.2.3 (Fait) *Un* fait *est une partie A de P telle que $A^{\bot\bot} = A$. A est* valide *quand $1 \in A$.*

Définition 3.2.4 *Définissons les opérations suivantes pour des faits A, B:*
– $A \otimes B = (A \star B)^{\bot\bot}$,
– $A \parr B = (B^\bot \star A^\bot)^\bot$,
– $A \& B = A \cap B$,
– $A \oplus B = (A \cup B)^{\bot\bot}$,
– $?A = (A^\bot \cap K)^\bot$,
– $!A = (A \cap K)^{\bot\bot}$.

Définition 3.2.5 (Structure de phase, Validité) *Une* structure de phase *(P, S) est la donnée d'un espace de phases enrichi P et d'une* valuation *qui associe un fait $S(p)$ à tout symbole de proposition p.*

*Etant donnée une structure de phase, on définit inductivement l'*interprétation *$S(A)$ d'une formule A de manière évidente. L'interprétation d'un séquent Γ est définie par induction: $S(()) = \bot$ et $S(\Gamma, \Delta) = S(\Gamma) \parr S(\Delta)$.*

Soit A une formule: A est dite valide dans S *quand $1 \in S(A)$. A est une* tautologie *si elle est valide dans toute structure de phase. Un séquent $\vdash \Gamma$ est valide si $1 \in S(\Gamma)$ pour toute structure de phase S.*

Théorème 3.2.6 (Correction et Complétude [17]) *Un séquent est prouvable dans le calcul des séquents ssi il est valide.*

Il existe une version intuitionniste de la logique linéaire, la sémantique des phases est modifiée en remplaçant l'ensemble $\perp$ des antiphases par la donnée pure et simple d'un ensemble $\mathcal{F}$ de parties de P, les *faits*, tel que:
– $P \in \mathcal{F}$,
– $\mathcal{F}$ est clos par intersection arbitraire,
– $\forall a \in P, \forall A \in \mathcal{F}, a \multimap A = \{x \in P \mid a \star x \in A\}$ est un fait,

le reste étant identique au cas classique. La double negation est remplacée par l'opération de clôture.

On a alors aussi des théorèmes de correction et complétude.

3.3 Réseaux

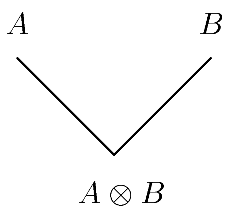
Voici une brève présentation d'une autre syntaxe pour les preuves en logique linéaire: les réseaux de preuve [17], qui sont une des grandes nouveautés de la logique linéaire.

Les réseaux de preuve sont la syntaxe naturelle des preuves dans le fragment multiplicatif de la logique linéaire.

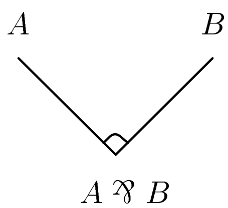
Définition 3.3.1 *Une* structure de preuve *est un graphe construit à partir de (petits) sous-graphes appelés* liens *pour lesquels on distingue des sommets* prémisses *et des sommets* conclusions*; les sommets sont étiqués par des formules; et les liens sont:*

axiome

coupure



tenseur

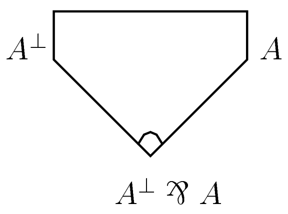


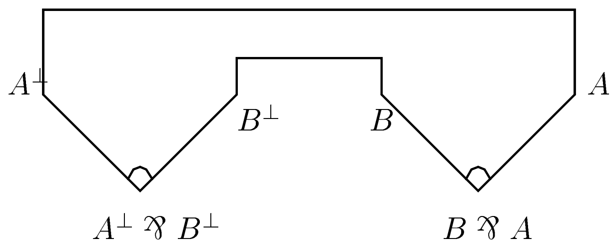
par

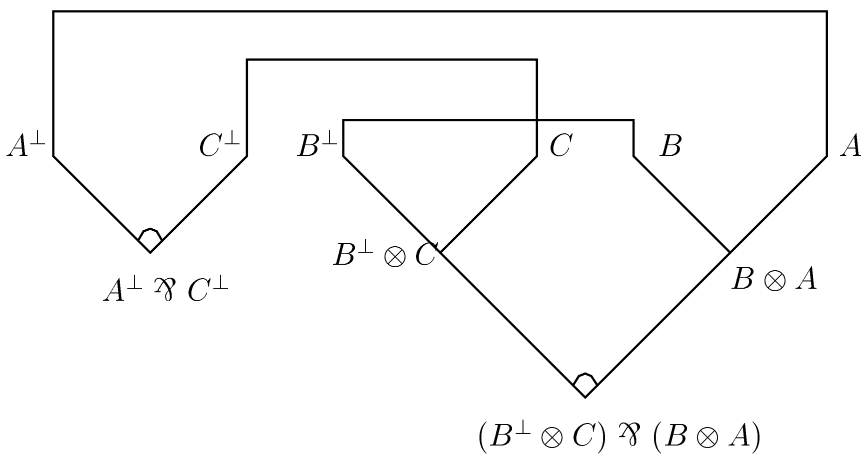
On demande en outre que toute occurrence de formule soit conclusion d'exactement un lien, et prémisse d'au plus un lien.

Une formule qui n'est prémisse d'aucun lien est dite terminale*, on dira aussi qu'elle est* conclusion *de la structure.*

Exemples:





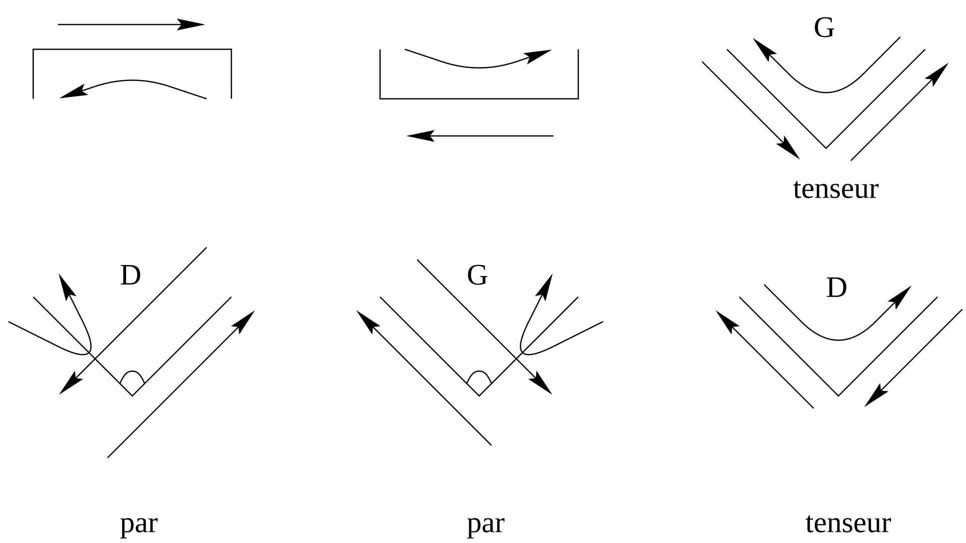


Définition 3.3.2 *A une preuve π (de conclusion $\vdash A_1, \ldots, A_n$) en calcul des séquents multiplicatif, on associe une structure de preuve π^- (ayant pour conclusions $A_1, \ldots, A_n$) de manière évidente.*

Deux preuves d'un même séquent ne différant que par l'ordre des règles seront traduites en la même structure de preuve. Donc la traduction quotiente le calcul des séquents par les commutations de règles.

Maintenant parmi toutes les structures de preuve, certaines ne sont pas la traduction d'une preuve en calcul des séquents, comme le deuxième exemple ci-dessus. On définit donc un critère de correction qui assure qu'une structure de preuve donnée est bien la traduction d'une preuve en calcul des séquents.

Définition 3.3.3 *Les* positions d'interrupteurs *associées aux liens sont:*



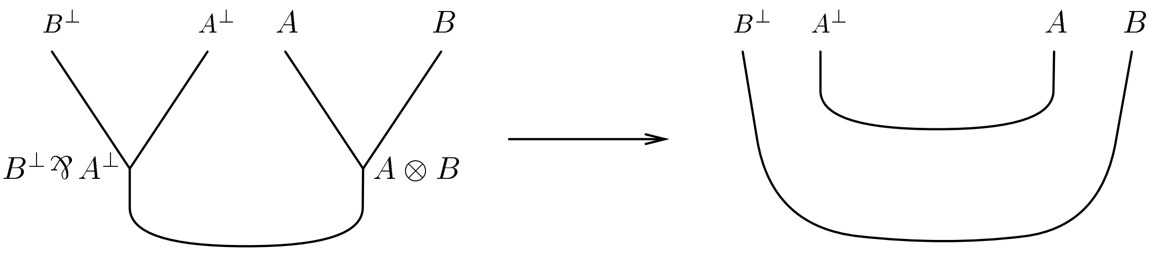
Pour faire un voyage*, on fixe une position d'interrupteur pour chaque lien de la structure de preuve, on part d'une formule et on parcourt le graphe comme prescrit par les positions d'interrupteurs. Un voyage est dit* court *si on revient au point de départ avant d'avoir parcouru toutes les arètes dans les deux sens (ou de manière équivalente: "avant d'avoir parcouru toute formule A dans les deux sens, $A^\wedge$ et $A_\vee$"). Il est dit* long *sinon.*

Définition 3.3.4 *On dit qu'une structure de preuve β est un* réseau de preuve *lorsque tous les voyages sont longs. On dit alors aussi: un réseau* correct.

Théorème 3.3.5 (Séquentialisation, [17]) *Une structure de preuve β est un réseau de preuve ssi elle est la traduction d'une preuve en calcul des séquents.*

3.4 Elimination des coupures

On définit localement l'élimination des coupures dans les structures de preuves:



Théorème 3.4.1 (Normalisation forte et confluence, [17]) *L'élimination des coupures préserve le fait d'être un réseau de preuve et a les propriétés de normalisation forte et de confluence.*

Pour traiter dans le cadre des structures de preuve tous les connecteurs de la logique linéaire (pas seulement les multiplicatifs), on étend la notion de structure de preuve à celle de structure de preuve avec boîtes, et les résultats de séquentialisation et normalisation forte s'adaptent au cadre général [17].

Chapter 4

Observation des stores et des succès

Du point de vue de la programmation logique, les aspects opérationnels de la programmation concurrente par contraintes devraient être liés étroitement à la théorie de la preuve, à travers le paradigme *calcul = recherche de preuve*. Ce paradigme, d'abord introduit pour les clauses de Horn, a été élégamment appliqué à la programmation logique par contraintes (avec ou sans négation par défaut, ou négation constructive), où la nature logique du système de contraintes s'étend aux buts et aux déclarations de programmes, et établit des connections fortes entre la sémantique opérationnelle et la déduction logique (voir [22, 33, 54, 14]). De telles connections facilitent l'écriture et la compréhension des programmes et fournissent des outils pour raisonner sur leur comportement opérationnel.

Maher [33] a été le premier à suggérer la possibilité d'une interprétation logique du mécanisme de synchronisation.

Dans [31] Lincoln et Saraswat donnent une connection intéressante entre l'observation des stores `cc` et la logique intuitionniste (LI). L'idée est d'exprimer les agents et les observations par des formules et de lire un séquent $A \vdash B$ comme "l'agent A satisfait le test B". Le résultat principal est que pour toute contrainte c et toute formule A associée à un agent, $A \vdash_{IL} c$ ssi c est un store accessible à partir de A.

L'observation des stores est importante, dans la mesure où ils représentent l'information véhiculée par les agents, qui est globalement accessible, et constituent la partie monotone du calcul.

Dans ce chapitre (basé sur [46]), nous nous intéressons aux observations caractérisables dans les logiques intuitionniste et linéaire, et nous montrons par des contre-exemples les obstacles à des caractérisations d'observations plus fines.

Dans le premier paragraphe nous montrons que si l'observation des stores est caractérisable en logique intuitionniste, en revanche ni les succès, ni les suspensions ne peuvent être caractérisés en logique intuitionniste, en raison de la règle d'*affaiblissement* (à gauche). Dans le paragraphe suivant nous montrons que l'observation des succès peut être caractérisée en logique linéaire, mais pas celle des suspensions.

4.1 Observations caractérisables en logique intuitionniste

4.1.1 L'observation des stores

Fixons un système de contraintes $(\mathcal{C}, \Vdash_\mathcal{C})$ et un ensemble de déclarations $\mathcal{D}$.

Définition 4.1.1 *On traduit les agents* cc *déterministes en formules intuitionnistes de la manière suivante:*

$$
\begin{array}{ll}
c^\dagger = c, \textit{ si } c \textit{ est une constrainte} & tell(c)^\dagger = c \\
p(\vec{x})^\dagger = p(\vec{x}) & (\exists x A)^\dagger = \exists x A^\dagger \\
(c \to A)^\dagger = c \Rightarrow A^\dagger & (A \parallel B)^\dagger = A^\dagger \wedge B^\dagger
\end{array}
$$

Si Γ *est le multi-ensemble d'agents* $(A_1 \dots A_n)$, *on définit* $\Gamma^\dagger = A_1^\dagger \wedge \dots \wedge A_n^\dagger$. *La traduction* $(\vec{x}; c; \Gamma)^\dagger)$ *d'une configuration* $(\vec{x}; c; \Gamma)$ *est la formule* $\exists \vec{x}(c \wedge \Gamma^\dagger)$.

On note LI($\mathcal{C}$,$\mathcal{D}$) le système de déduction obtenu en ajoutant à LI:

- l'axiome non-logique $c \vdash d$ pour tout $c \Vdash_\mathcal{C} d$ dans $\Vdash_\mathcal{C}$,
- l'axiome non-logique $p(\vec{x}) \vdash A^\dagger$ pour toute déclaration $p(\vec{x}) = A$ dans $\mathcal{D}$.

$\dashv\vdash$ dénote l'équivalence logique.

Théorème 4.1.2 (Correction) *Soient* $(\vec{x}; c; \Gamma)$ *et* $(\vec{y}; d; \Delta)$ *des configurations* cc *déterministes.*
Si $(\vec{x}; c; \Gamma) \equiv (\vec{y}; d; \Delta)$ *alors* $(\vec{x}; c; \Gamma)^\dagger \dashv\vdash_{LI(\mathcal{C},\mathcal{D})} (\vec{y}; d; \Delta)^\dagger$.
Si $(\vec{x}; c; \Gamma) \longrightarrow (\vec{y}; d; \Delta)$ *alors* $(\vec{x}; c; \Gamma)^\dagger \vdash_{LI(\mathcal{C},\mathcal{D})} (\vec{y}; d; \Delta)^\dagger$.

Preuve. Par induction sur $\equiv$ et $\longrightarrow$.
– Pour la *composition parallèle* et la α-*conversion*, c'est immédiat.
– Pour l'*inaction*, on a $A^\dagger \wedge 1 \dashv\vdash A^\dagger$.
– Pour les *variables locales*, on a $\exists x(A \wedge B) \dashv\vdash A \wedge \exists x B$ et $\exists x A \dashv\vdash A$ dès que $x \notin vl(A)$.
– Pour *Tell*, $\equiv$ et les *déclarations*, c'est immédiat.
– Pour *Ask*, on a bien $c \wedge (d \Rightarrow A) \vdash c \wedge A$ si $c \Vdash_\mathcal{C} d$, cqfd. ■

La réciproque est vraie pour l'observation des stores.

Notation:
▶ Soient $\kappa = (\vec{x}; c; \Gamma)$ une configuration $\mathtt{cc}$ déterministe, et ϕ une contrainte ou un nom de procédure. On écrit $\kappa \xrightarrow{\gg} \phi$ pour signifier:

- si ϕ est une contrainte: "il existe une configuration $(\vec{y}; d; \Delta)$, telle que $d \vdash_{\mathcal{C}} \phi$ et $\kappa \longrightarrow (\vec{y}; d; \Delta)$",
- si ϕ est un nom de procédure: "il existe une configuration $(\vec{y}; d; \phi, \Delta)$, telle que $vl(\phi) \cap \vec{y} = \emptyset$ et $\kappa \longrightarrow (\vec{y}; d; \phi, \Delta)$".

Lemme 4.1.3 *Soient κ et λ deux configurations $\mathtt{cc}$ déterministes telle que $\kappa^\dagger = \lambda^\dagger$, et ϕ une contrainte ou un nom de procédure. On a $\kappa \xrightarrow{\gg} \phi$ ssi $\lambda \xrightarrow{\gg} \phi$.*

Preuve. On montre par induction sur la formule $\kappa^\dagger = \lambda^\dagger$ que $\kappa \xrightarrow{\gg} \phi$ ssi $\lambda \xrightarrow{\gg} \phi$.

– Si $\kappa^\dagger = \lambda^\dagger$ est atomique, c'est clair.

– Si $\kappa^\dagger = \lambda^\dagger = c \Rightarrow A^\dagger$, avec c une contrainte et A un agent, alors κ et λ sont nécessairement toutes les deux égales à $(\emptyset; 1; c \to A)$.

– Si $\kappa^\dagger = \lambda^\dagger = \exists x A^\dagger$, alors les deux seules possibilités pour κ et λ sont $(x, \vec{y}; c; \Gamma)$ et $(\emptyset; 1; \exists x \exists \vec{y}(tell(c) \parallel \Gamma))$, et ces deux configurations sont structurellement congruentes, donc dans tous les cas $\kappa \xrightarrow{\gg} \phi$ ssi $\lambda \xrightarrow{\gg} \phi$.

– Si $\kappa^\dagger = \lambda^\dagger = A^\dagger \wedge B^\dagger$, alors les quatre possibilités pour κ et λ sont: $(\emptyset; 1; \Gamma, \Delta)$ (avec $\Gamma^\dagger = A^\dagger$ et $\Delta^\dagger = B^\dagger$), $(\emptyset; 1; A \parallel B)$, $(\emptyset; c; B)$ (si $A^\dagger = c$, une contrainte, c'est-à-dire $A = c$ ou $A = tell(c)$) et $(\emptyset; c \wedge d; 1)$ (si $A^\dagger = c$ et $B^\dagger = d$, des contraintes). L'induction sert uniquement dans le premier cas, et le résultat est évident. ■

Théorème 4.1.4 *Soient $\kappa = (\vec{x}; c; \Gamma)$ une configuration $\mathtt{cc}$ déterministe, et ϕ une contrainte ou un nom de procédure.*
Si $\kappa^\dagger \vdash_{LI(\mathcal{C},\mathcal{D})} \phi$, alors $\kappa \xrightarrow{\gg} \phi$.

Preuve. On prouve le résultat pour des multi-ensembles d'agents. On montre que si ${A_1}^\dagger, \ldots, {A_n}^\dagger \vdash_{LI(\mathcal{C},\mathcal{D})} \phi$, où les A_i sont des agents et ϕ est soit une contrainte soit un nom de procédure, alors $(\emptyset; 1; A_1, \ldots, A_n) \xrightarrow{\gg} \phi$.

Cela permet bien de conclure: soient en effet $(\vec{x}; c; \Gamma)$ une configuration $\mathtt{cc}$ déterministe, et ϕ une contrainte ou un nom de procédure. On note que $(\vec{x}; c; \Gamma)^\dagger = \exists \vec{x}(tell(c) \parallel \Gamma)^\dagger$. Donc si $(\vec{x}; c; \Gamma)^\dagger \vdash_{LI(\mathcal{C},\mathcal{D})} \phi$, alors $(\vec{x}; 1; tell(c), \Gamma)) \equiv (\emptyset; 1; \exists \vec{x}(tell(c) \parallel \Gamma)) \xrightarrow{\gg} \phi$. Mais $(\vec{x}; 1; tell(c), \Gamma))^\dagger = (\vec{x}; c; \Gamma))^\dagger$. Donc d'après le Lemme 4.1.3, $(\vec{x}; c; \Gamma)) \xrightarrow{\gg} \phi$, cqfd.

Procédons par induction sur une preuve π, en calcul des séquents, de ${A_1}^\dagger, \ldots, {A_n}^\dagger \vdash_{LI(\mathcal{C},\mathcal{D})} \phi$, où les A_i sont des agents et ϕ est soit une contrainte soit un nom de procédure.

Remarquons tout d'abord que cette induction a un sens. En effet, on peut supposer que les coupures portent uniquement sur des axiomes non-logiques, donc qu'elles sont de l'une des formes suivantes:

$$\frac{\Gamma \vdash p \qquad p \vdash \phi}{\Gamma \vdash \phi} \qquad\qquad \frac{\Gamma \vdash e \qquad e \vdash f}{\Gamma \vdash f}$$

$$\frac{p \vdash \psi \qquad \Gamma, \psi \vdash \phi}{\Gamma, p \vdash \phi} \qquad\qquad \frac{e \vdash f \qquad \Gamma, f \vdash \phi}{\Gamma, e \vdash \phi}$$

Donc, quand on remonte dans une telle preuve, la formule à droite du séquent reste soit une sous-formule d'une contrainte, donc elle-même une contrainte, soit un nom de procédure. D'autre part les formules à gauche restent des sous-formules de traductions d'agents ou des contraintes ou des noms de procédure, donc finalement des agents. (Remarquons ici que dans l'hypothèse d'induction, on a besoin du résultat non seulement pour les contraintes, mais aussi pour les noms de procédure.)

Chaque règle logique simule une règle de transition cc.

– π est un axiome: on utilise la réflexivité de $\longrightarrow$ dans le cas d'un axiome logique, la règle *déclarations* pour un axiome $p \vdash q$; le cas d'un axiome $d \vdash_{\mathcal{C}} e$ est trivial.

– π se termine par une coupure: les cas possibles sont énumérés ci-dessus. Considérons par exemple:

$$\frac{\Gamma^\dagger \vdash p \qquad p \vdash \phi}{\Gamma^\dagger \vdash \phi}$$

Par hypothèse d'induction, $(\emptyset; 1; \Gamma) \stackrel{\gg}{\longrightarrow} p$, c'est-à-dire qu'il existe une configuration $(\vec{y}; d; p, \Delta)$ telle que $vl(p) \cap \vec{y} = \emptyset$ et $(\emptyset; 1; \Gamma) \longrightarrow (\vec{y}; d; p, \Delta)$. Donc $(\emptyset; 1; \Gamma) \longrightarrow (\vec{y}; d; \phi, \Delta)$, avec $vl(\phi) \cap \vec{y} = \emptyset$ puisque $vl(\phi) \subset vl(p)$. Si $\phi^\dagger = \phi$ est un nom de procédure, c'est terminé. Si $\phi^\dagger$ est une contrainte c, alors $\phi = tell(c)$ et on a $(\emptyset; 1; \Gamma) \longrightarrow (\vec{y}; d \wedge c; \Delta)$, cqfd.
Les autres cas sont similaires.

– π se termine par une introduction à gauche de 1: immédiat par la règle *inaction*.

– π se termine par un affaiblissement:

$$\frac{\Gamma^\dagger \vdash \phi}{\Gamma^\dagger, A^\dagger \vdash \phi}$$

Par hypothèse d'induction, $(\emptyset; 1; \Gamma) \stackrel{\gg}{\longrightarrow} \phi$, et donc $(\emptyset; 1; A, \Gamma) \stackrel{\gg}{\longrightarrow} \phi$ grâce à la monotonie de $\longrightarrow$ (Proposition 2.1.7).

– π se termine par une contraction:

$$\frac{\Gamma^\dagger, A^\dagger, A^\dagger \vdash \phi}{\Gamma^\dagger, A^\dagger \vdash \phi}$$

Par hypothèse d'induction, $(\emptyset; 1; A, A, \Gamma) \xrightarrow{\gg} \phi$. Dans cette suite de transitions, certaines étapes activent l'une ou l'autre occurence de A, d'autres activent un sous-agent de Γ. Le point important est que pour l'agent déterministe A, la suite des transitions dans lesquelles il peut être actif (autrement dit la suite des actions qu'il peut effectuer) est déterminée (pas de $+$). Donc on peut supposer que dans l'exécution ci-dessus, à chaque fois qu'on active un sous-agent d'une occurrence de A, on effectue juste après la même transition avec l'autre occurrence, c'est bien sûr possible grâce à la monotonie du store (Proposition 2.1.8). Maintenant partant de la configuration $(\emptyset; 1; A, \Gamma)$, on simule l'exécution ci-dessus, en appliquant les mêmes transitions aux sous-agents de Γ et en "contractant" les paires de transitions pour les sous-agents de A: on n'applique la règle qu'à une des deux copies, toujours la même. On obtient bien alors une exécution $(\emptyset; 1; A, \Gamma) \xrightarrow{\gg} \phi$.

– π se termine par:

$$\frac{\Gamma^\dagger, A^\dagger, B^\dagger \vdash \phi}{\Gamma^\dagger, A^\dagger \wedge B^\dagger \vdash \phi}$$

Par hypothèse d'induction, $(\emptyset; 1; A, B, \Gamma) \xrightarrow{\gg} \phi$. Si $A^\dagger$ et $B^\dagger$ sont des contraintes, il y a a priori une ambiguïté sur l'agent C dont la traduction est $A^\dagger \wedge B^\dagger$: il peut s'agir de $tell(A \wedge B)$ ou de $tell(A) \parallel tell(B))$. Mais comme les deux configurations ont la même traduction que $(\emptyset; 1; A, B, \Gamma)$, d'après le Lemme 4.1.3, $(\emptyset; 1; C, \Gamma) \xrightarrow{\gg} \phi$.

– π se termine par:

$$\frac{\Gamma^\dagger \vdash \phi \qquad \Delta^\dagger \vdash \psi}{\Gamma^\dagger, \Delta^\dagger \vdash \phi \wedge \psi}$$

Déjà $\phi \wedge \psi$ n'est pas atomique, donc ϕ et ψ sont des contraintes. Le résultat est immédiat: partant de la configuration $(\emptyset; 1; \Gamma, \Delta)$, on met bout à bout les deux exécutions $(\emptyset; 1; \Gamma) \xrightarrow{\gg} \phi$ et $(\emptyset; 1; \Delta) \xrightarrow{\gg} \psi$.

– π se termine par une introduction à droite de $\exists$ (cas où ϕ est une contrainte): immédiat.

– π se termine par:

$$\frac{\Gamma^\dagger, A^\dagger \vdash \phi}{\Gamma^\dagger, \exists x A^\dagger \vdash \phi}\; x \notin vl(\Gamma, \phi)$$

Par hypothèse d'induction, $(\emptyset; 1; A, \Gamma) \stackrel{\gg}{\longrightarrow} \phi$. Comme $x \notin vl(\Gamma)$, on a $(\emptyset; 1; \exists x A, \Gamma) \equiv (x; 1; A, \Gamma)$, mais de plus $x \notin vl(\phi)$, donc par le Lemme 4.1.3, $(\emptyset; 1; \exists x A, \Gamma) \stackrel{\gg}{\longrightarrow} \phi$, cqfd.

– π se termine par:

$$\frac{\Gamma^\dagger, A^\dagger \vdash \phi \qquad \Delta^\dagger \vdash c}{\Gamma^\dagger, \Delta^\dagger, c \Rightarrow A^\dagger \vdash \phi}$$

Par hypothèse d'induction, $(\emptyset; 1; \Delta) \stackrel{\gg}{\longrightarrow} c$, c'est-à-dire qu'il existe une configuration $(\vec{y}; d; \Sigma)$, telle que $d \vdash_{\mathcal{C}} c$ et $(\emptyset; 1; \Delta) \longrightarrow (\vec{y}; d; \Sigma)$. Donc $(\emptyset; 1; c \to A, \Delta) \longrightarrow (\vec{y}; d; c \to A, \Sigma) \longrightarrow (\vec{y}; d; A, \Sigma)$. Par conséquent, $(\emptyset; 1; c \to A, \Delta, \Gamma) \longrightarrow (\vec{y}; d; A, \Sigma, \Gamma)$. De plus par hypothèse d'induction, $(\emptyset; 1; A, \Gamma) \stackrel{\gg}{\longrightarrow} \phi$, d'où $(\emptyset; 1; c \to A, \Delta, \Gamma) \stackrel{\gg}{\longrightarrow} \phi$. ■

Corollaire 4.1.5 (Observation des stores) *Soient A un agent* cc *déterministe et c une contrainte.*

Si $A^\dagger \vdash_{LI(\mathcal{C},\mathcal{D})} c$, alors il existe un store d accessible à partir de A et tel que $d \vdash_{\mathcal{C}} c$.

Preuve. Evident en appliquant le Théorème précédent, c'est juste la définition d'un store accessible. ■

Remarque:

▶ La caractérisation des stores pour des configurations non-déterministes n'est pas évidente dans le cadre de la logique intuitionniste: en effet d'une part l'idée simple de traduire l'opérateur de choix $+$ par la disjonction $\vee$ suppose de modifier sensiblement la sémantique opérationnelle du $+$ (déjà pour la correction, car $A \vee B \not\vdash A$), et d'autre part l'idée de traduire $+$ par la conjonction préserve la correction, mais pas la caractérisation des stores car, par exemple, le store $c \wedge d$ n'est pas accessible à partir de la configuration $(\emptyset; 1; tell(c) + tell(d))$.

C'est une des difficultés qui incitent à interpréter les configurations cc par des formules dans une logique plus fine: la logique linéaire de Girard [17].

4.1.2 Succès et suspensions?

Si l'observation des stores est importante, elle ne représente qu'un aspect du comportement opérationnel des programmes cc.

Ainsi les trois programmes

$$p(x) = x \geq 1$$

$$p(x) = x \geq 1 \parallel p(x)$$

$$p(x) = x \geq 1 \parallel (false \to A)$$

définissent les mêmes stores ($x \geq 1$), donc ils sont équivalents vis-à-vis de l'observation des stores, alors que le premier termine sur un succès, le second boucle et le troisième suspend.

Notons d'abord que par rapport à la caractérisation des succès en programmation logique par contraintes, le sens de l'implication est inversé dans le théorème ci-dessus (on aurait pu s'attendre à quelque chose comme: “Si $(\vec{x}; c; \Gamma)^{\dagger} \vdash_{LI(\mathcal{C},\mathcal{D})} c$ alors c est un store accessible à partir de A”). C'est essentiellement dû à la présence de l'implication (via l'opérateur de suspension $\to$), qui contient une forme de négation et “renverse” le sens de la déduction.

En fait la correspondance $\to - \vdash$ est essentielle. Ainsi on pourrait imaginer caractériser les succès ou les suspensions en gardant le sens usuel ($\to - \dashv$), ou en considérant l'équivalence logique ($\to - \dashv\vdash$), mais comme le montrent les contre-exemples suivants, ni les succès ni les suspensions ne sont caractérisables en logique intuitionniste:

- $\dashv$: En général il est faux que $A \dashv B$ (où B est un succès ou une suspension) implique $(\emptyset; 1; A) \longrightarrow (\emptyset; 1; B)$. Par exemple $c \to d \dashv d$ mais $c \to d$ peut suspendre, et n'avoir donc aucun succès. Par ailleurs $d \dashv d \wedge (c \to d)$ et si d n'implique pas c, $d \parallel (c \to d)$ est une suspension, alors que la contrainte c n'est pas une suspension.

- $\vdash$: On a des problèmes similaires avec $\vdash$. $d \wedge (c \Rightarrow A) \vdash d$ alors que $d \parallel (c \to A)$ suspend dès qu'on n'a pas $d \vdash c$. Par ailleurs $d \wedge (d \Rightarrow e) \vdash d \Rightarrow e$, mais $d \parallel (d \to e)$ a un succès ($d \wedge e$) et ne suspend pas.

- $\dashv\vdash$: De même avec l'équivalence $\dashv\vdash$. Supposons que d n'implique pas c, et considérons l'équivalence suivante: $d \wedge (c \Rightarrow d) \dashv\vdash d$. On ne peut rien en conclure quant au comportement opérationnel des agents d et $d \parallel (c \to d)$.

L'obstacle est la règle structurelle d'affaiblissement (à gauche):

$$\frac{\Gamma \vdash B}{\Gamma, A \vdash B}$$

La logique linéaire de Girard [17] est un raffinement des règles de contraction et d'affaiblissement de la logique usuelle. C'est donc en logique linéaire qu'il semble naturel d'interpréter les programmes cc. C'est l'objet de la Section suivante.

4.2 Observations caractérisables en logique linéaire

Nous avons vu dans le paragraphe précédent que les stores accessibles à partir d'un agent `cc` A sont caractérisables en logique intuitionniste, mais ni les succès, ni les suspensions, à cause de la règle d'affaiblissement. C'est donc en logique linéaire qu'il semble naturel d'interpréter les programmes `cc`. Dans le même temps, il est naturel de passer à une version non-monotone des `cc`: `lcc` (voir Section 2.2).

Nous reprenons donc la version de programmation concurrente par contraintes non-monotone `lcc` présentée en Section 2.2, pour laquelle nous montrons que les succès peuvent être caractérisés dans le fragment multiplicatif-additif de la logique linéaire intuitionniste (ILL). D'autre part, la traduction des `cc` monotones dans `lcc` assure que le résultat sur l'observation des succès est valable également pour `cc`.

Enfin, nous montrons par des contre-exemples qu'en revanche les suspensions ne peuvent pas être caractérisées dans ILL.

4.2.1 L'observation des succès

Fixons un système de contraintes linéaires $(\mathcal{C}, \Vdash_{\mathcal{C}})$ et un ensemble de déclarations $\mathcal{D}$.

Définition 4.2.1 *On traduit les agents `lcc` (non nécessairement déterministes) en formules de la manière suivante:*

$$\begin{array}{rlrl} & c^{\ddagger} = c, \textit{ si } c \textit{ est une contrainte} & & \\ tell(c)^{\ddagger} &= c & p(\vec{x})^{\ddagger} &= p(\vec{x}) \\ (c \rightarrow A)^{\ddagger} &= c \multimap A^{\ddagger} & (A \parallel B)^{\ddagger} &= A^{\ddagger} \otimes B^{\ddagger} \\ (A + B)^{\ddagger} &= A^{\ddagger} \& B^{\ddagger} & (\exists x A)^{\ddagger} &= \exists x A^{\ddagger} \end{array}$$

Si Γ est le multi-ensemble d'agents $(A_1 \dots A_n)$, on définit $\Gamma^{\ddagger} = A_1^{\ddagger} \otimes \cdots \otimes A_n^{\ddagger}$. La traduction $(\vec{x}; c; \Gamma)^{\ddagger})$ d'une configuration $(\vec{x}; c; \Gamma)$ est la formule $\exists \vec{x}(c \otimes \Gamma^{\ddagger})$.

On note LLI($\mathcal{C}$,$\mathcal{D}$) le système de déduction obtenu en ajoutant à LLI:

- l'axiome non-logique $c \vdash d$ pour tout $c \Vdash_{\mathcal{C}} d$ dans $\Vdash_{\mathcal{C}}$,
- l'axiome non-logique $p(\vec{x}) \vdash A^{\ddagger}$ pour toute déclaration $p(\vec{x}) = A$ dans $\mathcal{D}$.

Théorème 4.2.2 (Correction) *Soient $(\vec{x}; c; \Gamma)$ et $(\vec{y}; d; \Delta)$ des configurations `lcc`.*
Si $(\vec{x}; c; \Gamma) \equiv (\vec{y}; d; \Delta)$ alors $(\vec{x}; c; \Gamma)^{\ddagger} \dashv\vdash_{LLI(\mathcal{C},\mathcal{D})} (\vec{y}; d; \Delta)^{\ddagger}$.
Si $(\vec{x}; c; \Gamma) \longrightarrow (\vec{y}; d; \Delta)$ alors $(\vec{x}; c; \Gamma)^{\ddagger} \vdash_{LLI(\mathcal{C},\mathcal{D})} (\vec{y}; d; \Delta)^{\ddagger}$.

Preuve. La preuve est essentiellement la même qu'en logique intuitionniste. Notons que pour l'opérateur de choix + traduit par la conjonction additive &, on a bien $A\&B \vdash A$ et $A\&B \vdash B$. ■

Réciproquement, on peut caractériser l'observation des succès, même en présence de l'opérateur de choix explicite +:

Notation:
▶ Soient $\kappa = (\vec{x}; c; \Gamma)$ une configuration `lcc`, et ϕ une contrainte ou un nom de procédure. On écrit $\kappa \overset{\ggg}{\longrightarrow} \phi$ pour signifier:

- si ϕ est une contrainte: "il existe une configuration $(\vec{y}; d; 1)$, telle que $d \vdash_{\mathcal{C}} \phi$ et $\kappa \longrightarrow (\vec{y}; d; 1)$",
- si ϕ est un nom de procédure: "il existe une configuration $(\vec{y}; d; \phi)$, telle que $vl(\phi) \cap \vec{y} = \emptyset$, $d \vdash_{\mathcal{C}} 1$ et $\kappa \longrightarrow (\vec{y}; d; \phi)$".

Lemme 4.2.3 *Soient κ et λ deux configurations* `lcc` *telle que $\kappa^{\ddagger} = \lambda^{\ddagger}$, et ϕ une contrainte ou un nom de procédure. On a $\kappa \overset{\ggg}{\longrightarrow} \phi$ ssi $\lambda \overset{\ggg}{\longrightarrow} \phi$.*

Preuve. Ici aussi, la preuve est essentiellement la même qu'en logique intuitionniste, la différence entre κ et λ revenant à des *tell* non effectués, ou à des $tell(c \otimes d)$ *vs* $tell(c) \parallel tell(d)$. ■

Théorème 4.2.4 *Soient $\kappa = (\vec{x}; c; \Gamma)$ une configuration* `lcc`*, et ϕ une contrainte ou un nom de procédure.*
Si $\kappa^{\ddagger} \vdash_{LLI(\mathcal{C},\mathcal{D})} \phi$, alors $\kappa \overset{\ggg}{\longrightarrow} \phi$.

Preuve. On prouve le résultat pour des multi-ensembles d'agents: si ${A_1}^{\ddagger}, \ldots, {A_n}^{\ddagger} \vdash_{LLI(\mathcal{C},\mathcal{D})} \phi$, alors $(\emptyset; 1; A_1, \ldots, A_n) \overset{\ggg}{\longrightarrow} \phi$. On procède par induction sur une preuve de $A_1^{\ddagger} \ldots A_n^{\ddagger} \vdash \phi$ en calcul des séquents. (C'est possible pour la même raison que dans la preuve du Théorème 4.1.4.)

Chaque règle logique simule une règle de transition `lcc`.

– π est un axiome: on utilise la réflexivité de $\longrightarrow$ dans le cas d'un axiome logique, la règle *déclarations* pour un axiome $p \vdash q$; le cas d'un axiome $d \vdash_{\mathcal{C}} e$ est trivial.

– π se termine par une coupure. Considérons par exemple:

$$\frac{\Gamma^{\ddagger} \vdash p \qquad p \vdash \phi}{\Gamma^{\ddagger} \vdash \phi}$$

Par hypothèse d'induction, $(\emptyset; 1; \Gamma) \overset{\ggg}{\longrightarrow} p$, c'est-à-dire qu'il existe une configuration $(\vec{y}; d; p)$ telle que $vl(p) \cap \vec{y} = \emptyset$, $d \vdash_{\mathcal{C}} 1$ et $(\emptyset; 1; \Gamma) \longrightarrow (\vec{y}; d; p)$. Donc $(\emptyset; 1; \Gamma) \longrightarrow (\vec{y}; d; \phi)$, avec $vl(\phi) \cap \vec{y} = \emptyset$ puisque $vl(\phi) \subset vl(p)$. Si $\phi^{\ddagger} = \phi$ est un nom de procédure, c'est terminé. Si $\phi^\ddagger$ est une contrainte c, alors $\phi = tell(c)$ d'où $(\emptyset; 1; \Gamma) \longrightarrow (\vec{y}; d \otimes c; 1)$ et $d \otimes c \vdash_{\mathcal{C}} c$, cqfd.

Les autres cas sont similaires.

– π se termine par une introduction à gauche de 1: immédiat par la règle *inaction.*

– π se termine par une introduction à gauche ou à droite de $\otimes$: comme dans le Théorème 4.1.4 (on a traité la conjonction intuitionniste multiplicativement).

– π se termine par:

$$\frac{\Gamma^\ddagger, A^\ddagger \vdash \phi}{\Gamma^\ddagger, A^\ddagger \& B^\ddagger \vdash \phi}$$

Par hypothèse d'induction, $(\emptyset; 1; A, \Gamma) \xrightarrow{\ggg} \phi$, Or $(\emptyset; 1; A+B, \Gamma) \longrightarrow (\emptyset; 1; A, \Gamma)$, donc $(\emptyset; 1; A + B, \Gamma) \xrightarrow{\ggg} \phi$.

– π se termine par une introduction à droite (cas où ϕ est une contrainte) ou à gauche de $\exists$: comme dans le Théorème 4.1.4.

– π se termine par:

$$\frac{\Gamma^\ddagger, A^\ddagger \vdash \phi \qquad \Delta^\ddagger \vdash c}{\Gamma^\ddagger, \Delta^\ddagger, c \multimap A^\ddagger \vdash \phi}$$

Par hypothèse d'induction, $(\emptyset; 1; \Delta) \xrightarrow{\ggg} c$, c'est-à-dire qu'il existe une configuration $(\vec{y}; d; 1)$, telle que $d \vdash_{\mathcal{C}} c$ et $(\emptyset; 1; \Delta) \longrightarrow (\vec{y}; d; 1)$. Donc en appliquant la règle *ask*, on obtient $(\emptyset; 1; c \rightarrow A, \Delta) \longrightarrow (\vec{y}; d; c \rightarrow A) \longrightarrow (\vec{y}; 1; A)$. Par conséquent, $(\emptyset; 1; c \rightarrow A, \Delta, \Gamma) \longrightarrow (\vec{y}; 1; A, \Gamma)$. De plus par hypothèse d'induction, $(\emptyset; 1; A, \Gamma) \xrightarrow{\ggg} \phi$, d'où $(\emptyset; 1; c \rightarrow A, \Delta, \Gamma) \xrightarrow{\ggg} \phi$.

– π se termine par une déréliction:

$$\frac{\Gamma^\ddagger, c \vdash \phi}{\Gamma^\ddagger, !c \vdash \phi}$$

C'est clair, il suffit de se rappeler que $!c \vdash c$.

– π se termine par une promotion: dans ce cas toutes les formules sont nécessairement des contraintes, donc c'est immédiat.

– π se termine par un affaiblissement:

$$\frac{\Gamma^\ddagger \vdash \phi}{\Gamma^\ddagger, !c \vdash \phi}$$

où c est une contrainte. Par hypothèse d'induction, $(\emptyset; 1; \Gamma) \xrightarrow{\ggg} \phi$, et donc $(\emptyset; 1; tell(!c), \Gamma) \xrightarrow{\ggg} \phi$ (on fait le *tell* et on note que $!c \vdash_{\mathcal{C}} 1$).

– π se termine par une contraction:

$$\frac{\Gamma^{\ddagger}, !c, !c \vdash \phi}{\Gamma^{\ddagger}, !c \vdash \phi}$$

où c est une contrainte. Par hypothèse d'induction, $(\emptyset; 1; tell(!c), tell(!c), \Gamma) \stackrel{\ggg}{\longrightarrow} \phi$. Evidemment le fait d'avoir deux occurrences de l'agent $tell(!c)$ n'apporte rien car $!c \otimes !c \dashv\vdash !c$. Donc on a aussi $(\emptyset; 1; tell(!c), \Gamma) \stackrel{\ggg}{\longrightarrow} \phi$. ■

Corollaire 4.2.5 (Observation des succès) *Soient A un agent* `lcc` *et c une contrainte linéaire.*

Si $A^{\ddagger} \vdash_{LLI(\mathcal{C},\mathcal{D})} c$, alors il existe une contrainte d telle que $d \vdash_{\mathcal{C}} c$ et $(\emptyset; 1; A) \longrightarrow (\emptyset; d; 1)$, i.e. d est un succès pour A.

Preuve. Evident en appliquant le Théorème précédent à la configuration $(\emptyset; 1; A)$. ■

Grâce à la traduction de `cc` dans `lcc` (Section 2.2, Proposition 2.2.9), cette caractérisation des succès en logique linéaire vaut aussi pour `cc`.

4.2.2 Exemple de preuve de propriété de programme

On reprend l'exemple de la Section 2.2.3: le programme du dîner des philosophes en `lcc`, et on utilise la sémantique de `lcc` en logique linéaire pour prouver la sûreté de ce programme.

Rappelons le codage de ce problème en `lcc`:

$$
\begin{array}{l}
\texttt{philosophe}_i \texttt{ =} \\
\quad\quad \texttt{ticket} \rightarrow \texttt{fourchette}_i \rightarrow \texttt{fourchette}_{i+1 \bmod n} \rightarrow \\
\quad\quad\quad\quad \texttt{(tell(mange}_i\texttt{)} \parallel \\
\quad\quad\quad\quad \texttt{mange}_i \rightarrow \\
\quad\quad\quad\quad\quad\quad \texttt{(tell(fourchette}_i\texttt{)} \parallel \texttt{tell(fourchette}_{i+1 \bmod n}\texttt{)} \parallel \\
\quad\quad\quad\quad\quad\quad \texttt{tell(ticket)} \parallel \texttt{philosophe}_i\texttt{))}.
\end{array}
$$

$$
\begin{array}{l}
\texttt{init = } \underbrace{\texttt{tell(ticket)} \parallel \cdots \parallel \texttt{tell(ticket)}}_{n-1 \ \ fois} \parallel \\
\quad\quad \texttt{tell(fourchette}_1\texttt{)} \parallel \cdots \parallel \texttt{tell(fourchette}_n\texttt{)} \parallel \\
\quad\quad \texttt{philosophe}_1 \parallel \cdots \parallel \texttt{philosophe}_n .
\end{array}
$$

Il s'agit de montrer que ce codage ne permet pas à deux philosophes de manger avec la même fourchette au même moment. On veut donc montrer que deux philosophes voisins ne mangent pas en même temps:

$$\forall i, \forall c, \forall A, \ \ (\emptyset; 1; init) \not\longrightarrow (\emptyset; mange_i \otimes mange_{i+1 \bmod n} \otimes c; A).$$

Il suffit donc de montrer:

$$\forall i, \forall B, \exists P, \exists \eta, \exists x,\ x \in \eta(init), x \notin \eta(mange_i \otimes mange_{i+1 \bmod n} \otimes B).$$

Considérons l'espace de phases P suivant:

- le monoïde est $(\mathbb{N}, \times, 1)$, où $\times$ est le produit de $\mathbb{N}$,
- $\mathcal{F} = \mathcal{P}(\mathbb{N})$.

Cherchons maintenant une valuation η. Il faut la caractériser sur *ticket*, $fourchette_i$ et $mange_i$ puis sur $philosophe_i$ et *init* en vérifiant les conditions découlant des déclarations:

- $\eta(ticket) = \{t\}, \forall i, \eta(fourchette_i) = \{f_i\}, \eta(philosophe_i) = \{p_i\}$,
- pour tout $i, \eta(mange_i) = \{e_i\}$ avec $e_i = f_i \times f_{i+1 \bmod n} \times t \times p_i$,
- $\eta(init) = \{g\}$ avec $g = t^{n-1} \times f_1 \times \cdots \times f_n \times p_1 \times \cdots \times p_n$,

où t, les f_i et les p_i sont des entiers premiers, deux à deux distincts.

Il reste à vérifier: $\forall i, \{p_i\} \subset E_{1i} = \eta(\text{déf. de } philosophe_i)$ et $\{g\} \subset E_2 = \eta(\text{déf. de } init)$.

Compte tenu de η on a: $E_{1i} = \{x \in \mathbb{N} \mid \exists y \in \mathbb{N}, f_{i+1 \bmod n} \times f_i \times t \times x = e_i \times y$ et $e_i \times y = f_i \times f_{i+1 \bmod n} \times t \times p_i\}$. En prenant $y = 1$, on a $\forall i, \{p_i\} \subset E_{1i}$. Et on a $g = E_2$.

Il reste à démontrer: $g \notin \eta(mange_i \otimes mange_{i+1 \bmod n} \otimes B) = \{x \in \mathbb{N} \mid \exists b, x = e_i \times e_{i+1 \bmod n} \times b\}$.

Par l'absurde: si $g = e_i \times e_{i+1 \bmod n} \times b$ alors $f_{i+1 \bmod n} \times g = e_i \times e_{i+1 \bmod n} \times b \times f_{i+1 \bmod n} = t^{n-3} \times f_1 \times \cdots \times f_{i-1} \times f_{i+3} \times \cdots \times f_n \times p_1 \times \cdots \times p_{i-1} \times p_{i+2} \times \cdots \times p_n \times e_i \times e_{i+1 \bmod n}$, ce qui est impossible (décomposition en facteurs premiers).

4.2.3 Discussion

On a vu que certaines règles de déduction pouvaient s'interpréter comme des étapes de transition `lcc`. Ainsi, les axiomes logiques traduisent une forme d'asynchronie, les axiomes non-logiques $p(\vec{x}) \vdash A$ correspondent à l'utilisation de déclarations récursives, la coupure est la composition d'exécutions, les règles pour le $\otimes$ expriment juste la structure monoïdale de l'opérateur de composition parallèle $\|$, et les règles pour le & correspondent bien à un choix non-déterministe. Les règles gauches pour $\multimap$ et $\exists$ s'interprètent aussi en termes calcul de processus: communication et localisation de variables.

Les seules règles qui sont un obstacle à une parfaite adéquation entre calcul et logique (dans le cadre de notre étude) sont les introductions à droite de $\multimap$ et $\exists$:

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}$$

$$\frac{\Gamma \vdash A[t/x]}{\Gamma \vdash \exists x A}$$

La première empêche l'observation des suspensions en LLI, la raison principale étant que de la déduction

$$A \otimes (c \multimap B) \vdash c \multimap (A \otimes B)$$

on ne peut pas conclure que $A \parallel (c \to B)$ suspend: il se peut que A ajoute suffisamment d'information dans le store pour débloquer la garde c (par exemple $c \otimes (c \multimap 1) \vdash c \multimap (c \otimes 1)$ mais on a aussi $c \otimes (c \multimap 1) \vdash 1$, et en effet l'agent $c \parallel c \to 1$ termine avec 1 comme succès, et ne suspend pas). Cette forme de "porosité" de l'implication – qui est essentielle car c'est exactement le théorème de déduction, qui lie implication et déduction – est toutefois un obstacle majeur au paradigme programmes = formules en programmation concurrente par contraintes. Ce problème est l'objet du Chapitre 6.

Chapter 5

Logique linéaire non-commutative mixte

Ce chapitre présente une version mixte de logique linéaire, combinant des connecteurs commutatifs et des connecteurs non-commutatifs. Nous donnons un calcul des séquents avec règles structurelles explicites (où l'information sur la manière dont les formules sont combinées dans un séquent est représentée par un ordre série-parallèle), un calcul des séquents correspondant avec règles structurelles réversibles implicites et une sémantique des phases (qui permet de montrer en passant l'élimination des coupures).

Cette logique mixte est motivée dans le Chapitre suivant par l'étude sémantique des programmes concurrents par contraintes, mais on peut l'envisager indépendamment de cette motivation. C'est même intentionnel: il ne s'agissait pas de définir une logique "ad hoc". En particulier, un point important était de retrouver les logiques commutative et non-commutative bien connues comme fragments du nouveau système. Ainsi cette version mixte étend d'une part la logique linéaire commutative de Girard [17] et d'autre part la logique linéaire cyclique de Girard et Yetter [18, 58].

Cette logique mixte est basée sur deux travaux précédents: la version intuitionniste multiplicative de de Groote, et un calcul des séquents préliminaire classique (avec des modalités pour l'échange, mais n'étendant pas la LL commutative) [11, 47]. Le travail que nous présentons étend la version de de Groote au cas classique, avec tous les connecteurs. L'étude du calcul des séquents avec les règles structurelles réversibles implicites nous amène à considérer une classe de relations ternaires, les ordres cycliques, que nous étudions plus particulièrement dans la Section 5.4.

5.1 Calcul des séquents

En logique linéaire purement non-commutative, on a la possibilité de considérer deux négations, comme Abrusci [1, 2], ou une seule négation comme Girard et Yetter. Considérer une seule négation revient à autoriser l'échange circulaire:

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

ce qui simplifie le calcul des séquents et permet de voir les réseaux comme des disques avec les conclusions sur le bord.

Nous faisons le choix d'une seule négation, ce qui est a priori assez naturel si l'on souhaite retrouver la LL commutative comme fragment.

Les formules sont définies par:

Définition 5.1.1 (Formules) *Les* formules *sont construites à partir de littéraux* $p, q, \ldots, p^{\perp}, q^{\perp}, \ldots$ *avec:*

– *les connecteurs non-commutatifs: la conjonction* $\odot$ (puis) *et la disjonction* $\lhd$ (séquentiel)*,*

– *les connecteurs multiplicatifs commutatifs:* $\otimes$ (tenseur) *et* $⅋$ (par)*,*

– *les connecteurs additifs:* $\&$ (avec) *et* $\oplus$ (plus)*,*

– *les connecteurs exponentiels:* $!$ (bien sûr) *et* $?$ (pourquoi pas)*,*

– *des constantes: multiplicatives* $\mathbf{1}$ *et* $\perp$*, et additives* $\top$ *et* $\mathbf{0}$*,*

– *les quantificateurs du premier ordre: universel* $\forall$ *et existentiel* $\exists$*.*

L'ensemble des formules est dénoté Φ*.*

On peut définir trois implications:
– implication commutative: $A \multimap B = A^{\perp} ⅋ B$,
– implication directe: $A \multimap\!\!\!\bullet\; B = A^{\perp} \lhd B$,
– rétro-implication: $A \bullet\!\!\!- B = B \lhd A^{\perp}$.

Comme d'habitude la *négation* est un connecteur défini par les lois de De Morgan:

$$\begin{array}{ll}
(p)^{\perp} = p^{\perp} & (p^{\perp})^{\perp} = p \\
(A \odot B)^{\perp} = B^{\perp} \lhd A^{\perp} & (A \lhd B)^{\perp} = B^{\perp} \odot A^{\perp} \\
(A \otimes B)^{\perp} = B^{\perp} ⅋ A^{\perp} & (A ⅋ B)^{\perp} = B^{\perp} \otimes A^{\perp} \\
(A\&B)^{\perp} = B^{\perp} \oplus A^{\perp} & (A \oplus B)^{\perp} = B^{\perp}\&A^{\perp} \\
(!A)^{\perp} = ?A^{\perp} & (?A)^{\perp} = !A^{\perp} \\
\mathbf{1}^{\perp} = \perp & \perp^{\perp} = \mathbf{1} \\
\top^{\perp} = \mathbf{0} & \mathbf{0}^{\perp} = \top \\
(\forall x A)^{\perp} = \exists x A^{\perp} & (\exists x A)^{\perp} = \forall x A^{\perp}
\end{array}$$

La négation est alors une involution: pour toute formule A, $A^{\perp\perp} = A$.

Définir un calcul des séquents pour une logique linéaire combinant des connecteurs multiplicatifs à la fois commutatifs et non-commutatifs pose le problème de représenter l'information commutatif / non-commutatif ($\parr$ / $\triangleleft$). Dans le cas purement non-commutatif d'Abrusci [1], les séquents sont des listes de formules. Dans le calcul ordonné de Retoré [44] cette information est représentée par des ordres attachés aux séquents. Parmi ces ordres, les ordres série-parallèles jouent un rôle important (voir par exemple [39] pour une présentation des ordres série-parallèles), dans la mesure où ces ordres sont ceux pour lesquels un séquent peut être interprété comme une seule formule.

Dans le calcul que nous présentons dans cette section, cette information est représentée par des ordres série-parallèles, et l'associativité de ";" et "," et la commutativité de "," sont explicites. Toutefois notons que le quotient de l'ensemble des séquents par les règles structurelles explicites réversibles n'est pas un ensemble d'ordres série-parallèles, de même que dans la logique cyclique, l'échange circulaire implicite confère aux séquents la structure de cycles (et non de listes). La question de la structure sous-jacente des séquents mixtes est l'objet de la Section 5.4.

La classe des ordres série-parallèles est bien connue: c'est la plus petite classe d'ordres contenant les singletons et close par composition en série ($x <_{\mathfrak{i};\mathfrak{j}} y$ ssi $x <_{\mathfrak{i}} y$ ou $x <_{\mathfrak{j}} y$ ou $(x, y) \in \mathfrak{i} \times \mathfrak{j}$) et composition parallèle ($x \leq_{\mathfrak{i},\mathfrak{j}} y$ ssi $x \leq_{\mathfrak{i}} y$ ou $x \leq_{\mathfrak{j}} y$).

Comme [11], nous adoptons la notation ",-;" pour les ordres série-parallèles.

Définition 5.1.2 (Séquents) *Les* séquents *sont de la forme* $\vdash \Gamma$, *où* $\Gamma \in \mathcal{H}$. $\mathcal{H}$ *et* $\mathcal{H}_0$ *sont respectivement les ensembles de* contextes *et de* contextes non vides *et sont définis par la grammaire suivante:*

- $\mathcal{H} ::= () \mid \mathcal{H}_0$
- $\mathcal{H}_0 ::= \Phi \mid (\mathcal{H}_0, \mathcal{H}_0) \mid (\mathcal{H}_0; \mathcal{H}_0)$

$\Gamma, \Delta \ldots$ dénotent des contextes (éventuellement vides). On emploie la notation $\Gamma[\ ]$ pour désigner un contexte avec un trou (une feuille de l'arbre binaire Γ), et $\Gamma[\Delta]$ est le contexte obtenu en "remplissant" le trou avec Δ, avec la convention suivante pour $\Delta = ()$: $\Gamma[\Gamma'; ()] = \Gamma[(); \Gamma'] = \Gamma[\Gamma', ()] = \Gamma[(), \Gamma']$.

On emploie la notation $?\Gamma$ pour un contexte quelconque dont les formules sont toutes précédées par un ?.

Les règles du *calcul des séquents* sont:

Axiome - Coupure

$$\vdash A^\perp, A \qquad \frac{\vdash \Gamma, A \qquad \vdash A^\perp, \Delta}{\vdash \Gamma, \Delta}$$

Règles structurelles

$$\frac{\vdash \Gamma[(\Delta, \Sigma), \Pi]}{\vdash \Gamma[\Delta, (\Sigma, \Pi)]} \qquad \frac{\vdash \Gamma[\Delta, (\Sigma, \Pi)]}{\vdash \Gamma[(\Delta, \Sigma), \Pi]} \qquad \frac{\vdash \Gamma[(\Delta; \Sigma); \Pi]}{\vdash \Gamma[\Delta; (\Sigma; \Pi)]} \qquad \frac{\vdash \Gamma[\Delta; (\Sigma; \Pi)]}{\vdash \Gamma[(\Delta; \Sigma); \Pi]}$$

$$\frac{\vdash \Gamma[\Delta, \Sigma]}{\vdash \Gamma[\Sigma, \Delta]}\text{échange} \qquad \frac{\vdash \Gamma[\Delta; \Sigma]}{\vdash \Gamma[\Delta, \Sigma]}\text{entropie} \qquad \frac{\vdash \Gamma, \Delta}{\vdash \Gamma; \Delta}$$

Non-commutatifs

$$\frac{\vdash \Gamma, A \qquad \vdash \Delta, B}{\vdash (\Delta; \Gamma), A \odot B} \qquad \frac{\vdash \Gamma, (A; B)}{\vdash \Gamma, A \triangleleft B}$$

Multiplicatifs commutatifs

$$\frac{\vdash \Gamma, A \qquad \vdash \Delta, B}{\vdash (\Delta, \Gamma), A \otimes B} \qquad \frac{\vdash \Gamma, (A, B)}{\vdash \Gamma, A \invamp B}$$

Additifs

$$\frac{\vdash \Gamma, A \qquad \vdash \Gamma, B}{\vdash \Gamma, A \& B}$$

$$\frac{\vdash \Gamma, A}{\vdash \Gamma, A \oplus B} \qquad \frac{\vdash \Gamma, B}{\vdash \Gamma, A \oplus B}$$

Constantes

$$\vdash \mathbf{1} \qquad \frac{\vdash \Gamma}{\vdash \Gamma, \bot}$$

$$\vdash \Gamma, \top \qquad \text{(pas de règle pour } \mathbf{0}\text{)}$$

Exponentiels

$$\frac{\vdash \Gamma, A}{\vdash \Gamma, ?A} \qquad \frac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}$$

$$\frac{\vdash \Gamma, (?A, ?A)}{\vdash \Gamma, ?A} \qquad \frac{\vdash \Gamma}{\vdash \Gamma, ?A}$$

$$\frac{\vdash \Gamma, (?\Delta, ?\Sigma)}{\vdash \Gamma, (?\Delta ; ?\Sigma)}$$

Quantificateurs

$$\frac{\vdash \Gamma, A}{\vdash \Gamma, \forall x A}\, x \notin vl(\Gamma) \qquad \frac{\vdash \Gamma, A[t/x]}{\vdash \Gamma, \exists x A}$$

Remarques:

▶ L'ordre série-parallèle associé à un séquent permet d'exprimer les contraintes de commutativité / non-commutativité sur les formules: les virgules "," représentent des $\parr$ et les points-virgules ";" des $\lhd$. Ainsi la règle d'échange est limitée en général à des contextes séparés par un virgule, et l'entropie dit exactement $A \lhd B \vdash A \parr B$ (et duallement $A \otimes B \vdash A \odot B$).

Le choix d'une seule négation force l'échange circulaire, comme en LL cyclique, et en effet la règle:

$$\frac{\vdash \Gamma, \Delta}{\vdash \Gamma ; \Delta}$$

et la règle d'entropie impliquent l'échange circulaire.

En fait cette règle dit un peu plus que simplement l'échange circulaire: elle permet de remplacer une point-virgule par une virgule lorsque ce séparateur est le séparateur principal du séquent (la racine de l'arbre). Evidemment cela n'implique pas la commutativité dans tout le séquent, et en fait cette règle est presque forcée 'sil on veut que la LL commutative et la LL cyclique soient des fragments d'un calcul (simple) des séquents. En termes de réseaux de preuve, cela signifie simplement que des réseaux avec au plus 2 conclusions peuvent pivoter librement, donc pour de telles sous-preuves d'une preuve plus grande, commutatif et non-commutatif devraient être indiscernables.

▶ Les règles pour les additifs, constantes et quantificateurs sont évidentes.

▶ Les exponentiels doivent permettre la commutativité si l'on souhaite avoir les équivalences habituelles $!A \otimes !B \cong !(A \& B) \cong !A \odot !B \cong !B \odot !A$ et duallement $?A \parr ?B \cong ?(A \oplus B) \cong ?A \lhd ?B \cong ?B \lhd ?A$. C'est la raison de la dernière règle pour les exponentiels (voir plus loin la preuve de $!(A \& B) \vdash !A \odot !B$). Notons que contrairement à [58] les $?A$ ne commutent pas avec toutes les autres formules du séquents (elles ne sont pas centrales) mais seulement avec les autres "?"; c'est d'ailleurs juste ce que disent les équivalences ci-dessus.

▶ A première vue, il y a étonnamment peu de règles (logiques), on s'attendrait à d'autres règles, par exemple pour $\odot$:

$$\frac{\vdash \Gamma, A \qquad \vdash \Delta[B]}{\vdash \Delta[\Gamma; A \odot B]}$$

une reminiscence de la règle intuitionniste [11]:

$$\frac{\Gamma \vdash A \qquad \Delta[B] \vdash C}{\Delta[\Gamma; A \multimapdotbothB B] \vdash C}$$

Comme nous allons le voir, de telles règles avec des contextes imbriqués sont inutiles. En effet la notation ",-;" pour les ordres série-parallèles est pratique pour écrire et lire des preuves, mais à cause des permutations (en particulier l'échange circulaire) il y a plusieurs manières d'écrire un même séquent de manière équivalente. Le point important est qu'avec les règles structurelles que nous avons données, toute formule A dans un séquent donné peut être "sortie", c'est-à-dire qu'on peut mettre le séquent sous la forme équivalente $\vdash \Gamma, A$. Cela a pour conséquence une simplification du calcul, analogue à la simplification quand on passe de la logique avec 2 négations à la logique cyclique.

Proposition 5.1.3 *Soient $\vdash \Gamma$ un séquent et A une formule de Γ. Par une certaine suite finie de règles structurelles, on peut passer du séquent $\vdash \Gamma$ à un séquent $\vdash \Delta, A$ consititué des mêmes formules.*

De plus ce processus est réversible.

Preuve. A est une feuille dans l'arbre binaire Γ représentant un ordre série-parallèle. Les nœuds de Γ sont des séparateurs: ";" ou ",". On procède par induction sur la longueur du chemin de A à la racine de Γ.

– Si $\Gamma = \Delta, A$ ou A, Δ, on applique au plus une règle d'échange, et on obtient le résultat.

– Si $\Gamma = \Delta; A$ ou $A; \Delta$, on applique une entropie et au plus une règle d'échange, et on obtient le résultat.

– Si $\Gamma = (\Delta_1, \Delta_2), \Sigma$ et A est dans Δ_1, alors on applique une associativité pour obtenir $\Delta_1, (\Delta_2, \Sigma)$ puis l'hypothèse d'induction permet de conclure.

– Si $\Gamma = (\Delta_1, \Delta_2), \Sigma$ et A est dans Δ_2, alors on applique un échange pour obtenir $(\Delta_2, \Delta_1), \Sigma$ et une associativité donne $\Delta_2, (\Delta_1, \Sigma)$ puis l'hypothèse d'induction permet de conclure.

– Si $\Gamma = (\Delta_1; \Delta_2), \Sigma$ et A est dans Δ_1, alors on applique la règle

$$\frac{\vdash (\Delta_1; \Delta_2), \Sigma}{\vdash (\Delta_1; \Delta_2); \Sigma}$$

puis associativité, entropie et hypothèse d'induction permettent de conclure.

– Si $\Gamma = (\Delta_1; \Delta_2), \Sigma$ et A est dans Δ_2, alors la règle d'échange donne $\Sigma, (\Delta_1; \Delta_2)$ puis on applique la règle

$$\frac{\vdash \Sigma, (\Delta_1; \Delta_2)}{\vdash \Sigma; (\Delta_1; \Delta_2)}$$

l'associativité, l'entropie et l'hypothèse d'induction.

– Les cas $\Gamma = (\Delta_1, \Delta_1); \Sigma$, $\Gamma = (\Delta_1; \Delta_1); \Sigma$, $\Gamma = \Delta, (\Sigma_1, \Sigma_2)$, $\Gamma = \Delta, (\Sigma_1; \Sigma_2)$, $\Gamma = \Delta; (\Sigma_1, \Sigma_2)$ et $\Gamma = \Delta; (\Sigma_1; \Sigma_2)$ sont parfaitement similaires.

Notons que le processus est réversible, puisque la règle d'entropie n'est utilisée que pour le séparateur principal.

De plus le résultat est unique, modulo l'associativité de ";" et "," et la commutativité de ",". ■

Bien sûr la Propriété ci-dessus est aussi valable pour un sous-contexte Π: il est possible de mettre tout séquent $\vdash \Gamma[\Pi]$ sous la forme $\vdash \Delta, \Pi$ avec les mêmes formules. Ceci justifie l'écriture des règles pour $\lhd$, $\parr$ et la dernière règle pour la modalité ?.

Exemples:

▶ Dans les exemples suivants, nous omettrons quelques parenthèses évidentes. Voici une preuve de $A \otimes B \vdash B \otimes A$:

$$\dfrac{\dfrac{\dfrac{\vdash A^\perp, A \qquad \vdash B^\perp, B}{\vdash (A^\perp, B^\perp), B \otimes A}}{\vdash (B^\perp, A^\perp), B \otimes A}}{\vdash B^\perp \parr A^\perp, B \otimes A}$$

▶ Voici une preuve de $A \otimes (A \multimap B) \vdash B$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash A^\perp, A \qquad \vdash B, B^\perp}{\vdash (A^\perp; B), B^\perp \odot A}}{\vdash A^\perp; B; B^\perp \odot A}}{\vdash B^\perp \odot A; A^\perp; B}}{\vdash (B^\perp \odot A; A^\perp); B}}{\vdash (B^\perp \odot A, A^\perp); B}}{\vdash (B^\perp \odot A) \parr A^\perp; B}}{\vdash (B^\perp \odot A) \parr A^\perp, B}$$

▶ Voici les preuves de l'équivalence $!A \odot !B \cong !(A \& B)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash A^\perp, A}{\vdash ?A^\perp, A}}{\vdash ?B^\perp, ?A^\perp, A} \qquad \dfrac{\dfrac{\vdash B^\perp, B}{\vdash ?B^\perp, B}}{\vdash ?B^\perp, ?A^\perp, B}}{\vdash ?B^\perp, ?A^\perp, A \& B}}{\vdash (?B^\perp; ?A^\perp), A \& B}}{\vdash (?B^\perp; ?A^\perp), !(A \& B)}}{\vdash ?B^\perp \lhd ?A^\perp, !(A \& B)}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash A^\perp, A}{\vdash B^\perp \oplus A^\perp, A}}{\vdash ?(B^\perp \oplus A^\perp), A}}{\vdash ?(B^\perp \oplus A^\perp), !A} \qquad \dfrac{\dfrac{\dfrac{\vdash B^\perp, B}{\vdash B^\perp \oplus A^\perp, B}}{\vdash ?(B^\perp \oplus A^\perp), B}}{\vdash ?(B^\perp \oplus A^\perp), !B}}{\vdash (?(B^\perp \oplus A^\perp); ?(B^\perp \oplus A^\perp)), !A \odot !B}}{\vdash ?(B^\perp \oplus A^\perp), ?(B^\perp \oplus A^\perp), !A \odot !B}}{\vdash ?(B^\perp \oplus A^\perp), !A \odot !B}$$

▶ Voici une preuve de $A \otimes \mathbf{1} \vdash A$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash A^\perp, A}{\vdash A^\perp ; A}}{\vdash (A^\perp ; A), \perp}}{\vdash (A^\perp ; A) ; \perp}}{\vdash \perp ; (A^\perp ; A)}}{\vdash (\perp ; A^\perp) ; A}}{\vdash (\perp , A^\perp), A}}{\vdash \perp \parr A^\perp, A}$$

On peut désormais remarquer que le calcul des séquents sans coupures satisfait la *propriété de la sous-formule.*

Il possède aussi la propriété d'élimination des coupures, que nous montrerons avec la sémantique des phases.

5.2 Sémantique des phases

On ne traite ici que le cas propositionnel, l'extension aux quantificateurs est immédiate, comme dans [17].

Définition 5.2.1 (Espace de phases mixte) *Un* espace de phases mixte *(plus simplement* espace de phases*) consiste en la donnée:*
– d'un ensemble partiellement ordonné $(P, \leq)$, dont les éléments sont appelés phases*,*
– d'un produit monoïdal $\cdot$ et d'un produit monoïdal commutatif $\star$, ayant tous deux 1 pour élément neutre, monotones en leurs deux arguments ($x \leq x'$ et $y \leq y'$ impliquent $x \cdot y \leq x' \cdot y'$ et $x \star y \leq x' \star y'$), et tels que $x \star y \leq x \cdot y$ pour tous $x, y \in P$,
– d'une partie $\perp_P$ de P, dont les éléments sont appelés les antiphases*; $\perp_P$ est un idéal pour l'ordre ($x \in \perp_P$ et $y \leq x$ impliquent $y \in \perp_P$); de plus pour tous $x, y \in P$, $x \cdot y \in \perp_P$ ssi $y \cdot x \in \perp_P$ ssi $x \star y \in \perp_P$.*

On notera aussi P l'espace de phase.

Définition 5.2.2 *Si G est une partie de P, son* dual *est défini par:*

$$G^\perp = \{p \in P \mid \forall q \in G, p \star q \in \perp_P\}$$

Si G et H sont des parties de P, on définit:

$$G \cdot H = \{p \cdot q \mid p \in G, q \in H\}$$

$$G \star H = \{p \star q \mid p \in G, q \in H\}.$$

De manière équivalente: $G^\perp = \{p \in P \mid \forall q \in G, p \cdot q \in \perp_P\} = \{p \in P \mid \forall q \in G, q \cdot p \in \perp_P\}$.

Définition 5.2.3 (Fait) *Un* fait *est une partie A de P telle que $A^{\perp\perp} = A$. A est* valide *quand $1 \in A$.*

Comme dans le cas de la logique linéaire habituelle (commutative ou cyclique), nous avons les propriétés (i,ii,iii,iv,vi) suivantes, plus la propriété (v) spécifique à la logique mixte:

Propriété 5.2.4 *(i) Pour tout $G \subset P$, $G \subset G^{\perp\perp}$.*
(ii) Pour tous $G, H \subset P$, $G \subset H \Rightarrow H^\perp \subset G^\perp$.
(iii) $G \subset P$ est un fait ssi il est de la forme $H^\perp$ pour un certain $H \subset P$.
(iv) $\perp_P$ est un fait puisque $\perp_P = \{1\}^\perp$ (au lieu de $\perp_P$ on écrira parfois $\perp$).
(v) L'ensemble des faits est un idéal pour l'ordre: si A est un fait, $x \in A$ et $y \leq x$, alors $y \in A$.
(vi) L'intersection d'une famille quelconque de faits est un fait.

Preuve. Les assertions (i) à (iv) sont immédiates.

(v) Soient G un fait, $x \in G$ et $y \leq x$. Si $z \in G^\perp$, alors $x \cdot z \in \perp$ donc $y \cdot z \in \perp$ (monotonicité de $\cdot$). D'où $y \in G^{\perp\perp} = G$.

(vi) Il suffit de vérifier que si $(G_i)_{i\in I}$ est une famille de faits, alors $\bigcap G_i = (\bigcup G_i^\perp)^\perp$: si $x \in \bigcap G_i$ alors pour tout $i \in I$, $x \in G_i$, maintenant si $y \in \bigcup G_i^\perp$, on a $y \in G_{i_0}^\perp$ pour un certain $i_0 \in I$, d'où $x \cdot y \in \perp_P$; réciproquement si $x \in (\bigcup G_i^\perp)^\perp$, alors pour tout $i \in I$ et tout $y \in G_i^\perp$, $x \cdot y \in \perp_P$, donc $x \in G_i^{\perp\perp} = G_i$, cqfd. ■

Définition 5.2.5 *Quelques exemples de faits: le plus grand (pour l'inclusion) $\top = \emptyset^\perp = P$, le plus petit $\mathbf{0} = \top^\perp$, et $\mathbf{1} = \perp^\perp$.*

Définition 5.2.6 *Définissons les opérations suivantes pour des faits A, B:*
- $A \odot B = (A \cdot B)^{\perp\perp}$,
- $A \lhd B = (B^\perp \cdot A^\perp)^\perp$,
- $A \otimes B = (A \star B)^{\perp\perp}$,
- $A \parr B = (B^\perp \star A^\perp)^\perp$,
- $A \& B = A \cap B$,
- $A \oplus B = (A \cup B)^{\perp\perp}$.

Lemme 5.2.7 *Si F et G sont deux parties quelconques de P, alors $F^{\perp\perp} \cdot G^{\perp\perp} \subset (F \cdot G)^{\perp\perp}$ et $F^{\perp\perp} \star G^{\perp\perp} \subset (F \star G)^{\perp\perp}$.*

Preuve. La preuve est la même que dans [17], en utilisant le fait que $x \cdot y \in \perp_P$ ssi $x \star y \in \perp_P$.

Considérons le cas de $\cdot$. Soient $p \in F^{\perp\perp}$ et $q \in G^{\perp\perp}$. Si $v \in (F \cdot G)^\perp$ alors pour tout $f \in F$ et $g \in G$, $v \cdot (f \cdot g) = (v \cdot f) \cdot g \in \perp_P$, donc pour tout $g \in G$, $v \cdot f \in G^\perp = G^{\perp\perp\perp}$, et $q \cdot (v \cdot f) = (q \cdot v) \cdot f \in \perp_P$, d'où $q \cdot v \in F^\perp = F^{\perp\perp\perp}$. Par conséquent $p \cdot q \cdot v \in \perp_P$, cqfd. ■

Lemme 5.2.8 *Soit G une partie quelconque de P: $G^{\perp\perp}$ est le plus petit fait contenant G.*

Propriété 5.2.9 *(i) Les opérations $\odot$ et $\lhd$, $\otimes$ et $\parr$, & et $\oplus$ satisfont les lois de De Morgan. De plus ces 6 opérations sont associatives, $\otimes$, $\parr$, & et $\oplus$ sont commutatives, $\mathbf{1}$ est neutre pour $\odot$ et $\otimes$, $\perp$ est neutre pour $\lhd$ et $\parr$, $\top$ et $\mathbf{0}$ sont respectivement neutres pour & et $\oplus$. Les propriétés de distributivité entre $\odot$ et $\oplus$, $\otimes$ et $\oplus$, $\lhd$ et &, $\parr$ et & sont satisfaites.*

(ii) Soient A et B deux faits:

$$A \otimes B \subset A \odot B \text{ (duallement } A \lhd B \subset A \parr B\text{)}.$$

Preuve. Seuls les cas suivants méritent notre attention:
– L'associativité des multiplicatifs (par dualité, on considère juste les conjonctions): en utilisant le Lemme 5.2.7, on a $(A \odot B) \odot C = ((A \cdot B)^{\perp\perp} \cdot C)^{\perp\perp} = ((A \cdot B)^{\perp\perp} \cdot C^{\perp\perp})^{\perp\perp} \subset (A \cdot B \cdot C)^{\perp\perp}$, et $(A \cdot B \cdot C)^{\perp\perp} \subset (A \odot B) \odot C$ est alors immédiat. Donc $(A \odot B) \odot C = (A \cdot B \cdot C)^{\perp\perp}$, cqfd. Le cas de $\otimes$ est similaire.
– Neutralité: ces propriétés reposent sur la neutralité de $1 \in \mathbf{1}$ pour $\cdot$ et $\star$ à la fois.
– Distributivités: les preuves sont exactement les mêmes que pour la LL commutative, en utilisant le Lemme 5.2.8.
– $A \otimes B \subset A \odot B$: il suffit de montrer que $A \star B \subset A \odot B = (A \cdot B)^{\perp\perp}$. Si $a \in A$ et $b \in B$, alors $a \cdot b \in A \cdot B \subset (A \cdot B)^{\perp\perp}$. $(A \cdot B)^{\perp\perp}$ est un fait et $a \star b \leq a \cdot b$ donc par la Propriété 5.2.4 (v), $a \star b \in (A \cdot B)^{\perp\perp}$. ■

Définition 5.2.10 – $A \multimap B = \{x \in P \mid \forall a \in A, a \star x \in B\}$,
– $A \multimap\!\!\bullet\, B = \{x \in P \mid \forall a \in A, a \cdot x \in B\}$,
– $B \,\bullet\!\!\!\multimapinv A = \{x \in P \mid \forall a \in A, x \cdot a \in B\}$.

Propriété 5.2.11 *Soient A et B deux faits:*

$$A \multimap\!\!\bullet\, B = A^\perp \lhd B, \quad B \,\bullet\!\!\!\multimapinv A = B \lhd A^\perp, \quad A \multimap B = A^\perp \parr B,$$
$$A \multimap \perp = A \multimap\!\!\bullet \perp = \perp \bullet\!\!\!\multimapinv A = A^\perp.$$

Donc $A \multimap\!\!\bullet\, B$, $B \,\bullet\!\!\!\multimapinv A$ et $A \multimap B$ sont des faits.

Preuve. On utilise encore ici le fait que pour tous $x, y \in P$, $x \cdot y \in \perp_P$ ssi $x \star y \in \perp_P$.

Considérons juste le cas de $\multimap\!\!\bullet$ (les autres sont similaires). Supposons que $x \in A \multimap\!\!\bullet\, B$. Soient $a \in A$ et $y \in B^\perp$; $a \cdot x \in B$ donc $y \cdot a \cdot x \in \perp_P$, d'où $x \in A^\perp \lhd B = (B^\perp \cdot A)^\perp$. Réciproquement, supposons que $x \in A^\perp \lhd B$, et prenons $a \in A$. Pour tout $y \in B^\perp$, $y \cdot a \cdot x \in \perp_P$, donc $a \cdot x \in B^{\perp\perp} = B$, d'où $x \in A \multimap\!\!\bullet\, B$. ■

Comme dans [20, 29], on étend la sémantique aux connecteurs exponentiels.

Si P est un espace de phases, posons $J(P) = \{x \in \mathbf{1}, x \in \{x \star x\}^{\perp\perp}$ et $x \in \{x \cdot x\}^{\perp\perp}\}$.

Définition 5.2.12 (Espace de phases enrichi) *Un* espace de phases enrichi *consiste en un espace de phases P et une partie K de $J(P)$ telle que $1 \in K$ et pour tous $x, y \in K$, $x \cdot y \in K \cap \{x \star y\}^{\perp\perp}$ et $x \star y \in K \cap \{x \cdot y\}^{\perp\perp}$.*

On écrira encore P pour l'espace enrichi.

Exemples:

▶ Si P est un espace de phases, $\{1\}$ et $I(P) = \{x \in \mathbf{1}, x = x \star x = x \cdot x$ et $\forall y \in P, x \star y = x \cdot y\}$ satisfont les axiomes de la Définition 5.2.12.

Définition 5.2.13 *Soit A un fait, on définit:*
– $?A = (A^{\perp} \cap K)^{\perp}$,
– $!A = (A \cap K)^{\perp\perp}$.

Propriété 5.2.14 *(i) Pour tout fait A, $?A$ et $!A$ sont des faits.*
(ii) $!$ et $?$ satisfont les lois de De Morgan.
(iii) Pour tous faits A et B, $A \subset B \Rightarrow\ !A \subset\ !B$,
(iv) $!!A = !A \subset A$,
(v) $!A \otimes !B \subset !(A \otimes B)$,
(vi) $!A \subset \mathbf{1}$, $!A = !A \otimes !A$ et $!A = !A \odot !A$,
(vii) $!A \otimes !B = !(A \& B) = !A \odot !B = !B \odot !A$.

Preuve. (i), (ii) et (iii) sont immédiats.

(iv) Pour un fait A, on a clairement $!A \subset A^{\perp\perp} = A$; en particulier $!!A \subset !A$, et par ailleurs $A \cap K \subset (A \cap K)^{\perp\perp} = !A$ et $A \cap K \subset K$ donc $A \cap K \subset !A \cap K \subset (!A \cap K)^{\perp\perp}$, d'où $!A = (A \cap K)^{\perp\perp} \subset (!A \cap K)^{\perp\perp} = !!A$.

(v) $!A \otimes !B = ((A \cap K)^{\perp\perp} \star (B \cap K)^{\perp\perp})^{\perp\perp} = ((A \cap K) \star (B \cap K))^{\perp\perp} \subset ((A \star B) \cap K)^{\perp\perp} \subset ((A \star B)^{\perp\perp} \cap K)^{\perp\perp} = !(A \otimes B)$.

(vi) La première inclusion est évidente. Si $x \in A \cap K$, $x \in K$ donc $x \in \{x \star x\}^{\perp\perp} \subset ((A \cap K) \star (A \cap K))^{\perp\perp}$, par conséquent $A \cap K \subset ((A \cap K) \star (A \cap K))^{\perp\perp} = (!A \star !A)^{\perp\perp}$ d'après le Lemme 5.2.7, d'où $!A = (A \cap K)^{\perp\perp} \subset !A \otimes !A$. On procède de même pour $\odot$.

(vii) D'après (vi), $!(A \& B) \subset !(A \& B) \otimes !(A \& B)$, or $A \& B = A \cap B \subset A$, d'où par (iii), $!(A \& B) \subset !A$ et de même $!(A \& B) \subset !B$, donc $!(A \& B) \subset !A \otimes !B$. Pour la même raison $!(A \& B) \subset !A \odot !B$. Réciproquement, $!A \otimes !B \subset !A \otimes \mathbf{1} = !A \subset A$, et de même $!A \otimes !B \subset B$, donc $!A \otimes !B \subset A \& B$. D'où d'après (iv) et (v), $!A \otimes !B = !!A \otimes !!B \subset !(!A \otimes !B) \subset !(A \& B)$. ■

Définition 5.2.15 (Structure de phase, Validité) *Une* structure de phase *(P, S) est la donnée d'un espace de phases enrichi P et d'une* valuation *qui associe un fait $S(p)$ à tout symbole de proposition p.*

*Etant donnée une structure de phase, on définit inductivement l'*interprétation *$S(A)$ d'une formule A de manière évidente. L'interprétation d'un contexte Γ est définie par induction: $S(()) = \perp$, $S(\Gamma; \Delta) = S(\Gamma) \lhd S(\Delta)$ et $S(\Gamma, \Delta) = S(\Gamma) \parr S(\Delta)$.*

Soit A une formule: A est dite valide *dans S quand $1 \in S(A)$. A est une* tautologie *si elle est valide dans toute structure de phase. Un séquent $\vdash \Gamma$ est valide si $1 \in S(\Gamma)$ pour toute structure de phase S.*

Théorème 5.2.16 (Correction) *Si un séquent est prouvable dans le calcul des séquents, alors il est valide.*

Preuve. Soit (P, S) une structure de phase, et $\vdash \Gamma$ un séquent prouvable dans le calcul des séquents. On procède par induction sur une preuve de $\vdash \Gamma$:
– La preuve est un axiome $\vdash A^\perp, A$: l'interprétation est $S(A^\perp, A) = S(A^\perp \parr A) = S(A \multimap A)$ par la Propriété 5.2.11, donc $S(A^\perp, A) = S(A) \multimap S(A)$ et $1 \in S(A^\perp, A)$.
– La preuve est un axiome $\vdash \mathbf{1}$: $1 \in \mathbf{1} = \perp^\perp$.
– La preuve est un axiome $\vdash \Gamma, \top$: l'interprétation est $S(\Gamma, \top) = S(\Gamma) \parr S(\top) = S(\top) \parr S(\Gamma) = S(\mathbf{0}) \multimap S(\Gamma) = \mathbf{0} \multimap S(\Gamma)$, et le plus petit fait $\mathbf{0} \subset S(\Gamma)$, par ailleurs 1 est neutre pour $\star$, donc $1 \in \mathbf{0} \multimap S(\Gamma)$.
– La preuve se termine par une règle de coupure: par hypothèse d'induction $1 \in S(\Gamma) \parr S(A)$ et $1 \in S(A^\perp) \parr S(\Delta)$, ce qui signifie $S(\Gamma)^\perp \subset S(A)$ et $S(A^\perp)^\perp = S(A) \subset S(\Delta)$, cqfd.
– La preuve se termine par une règle d'associativité ou la règle d'échange: le résultat est une conséquence de l'associativité de $\parr$ et $\lhd$, et de la commutativité de $\parr$ (Propriété 5.2.9 (i)).
– La preuve se termine par une entropie: immédiat par la Propriété 5.2.9 (ii).
– La preuve se termine par la règle:

$$\frac{\vdash \Gamma, \Delta}{\vdash \Gamma; \Delta}$$

Par hypothèse d'induction, $1 \in S(\Gamma) \parr S(\Delta)$, c'est-à-dire $S(\Gamma)^\perp \subset S(\Delta)$; comme 1 est neutre pour $\cdot$ c'est équivalent à $1 \in S(\Gamma)^\perp \multimap\!\bullet\, S(\Delta) = S(\Gamma) \lhd S(\Delta)$.
– La preuve se termine par la règle &: conséquence immédiate de $A \& B = A \cap B$.
– La preuve se termine par une règle $\oplus$: conséquence immédiate de $A \oplus B = (A \cup B)^{\perp\perp}$ et du Lemme 5.2.8.
– La preuve se termine par une règle $\lhd$ ou $\parr$: immédiat.
– La preuve se termine par la règle $\odot$: par induction $S(\Gamma)^\perp \subset S(A)$ et $S(\Delta)^\perp \subset S(B)$, donc $(S(\Delta) \lhd S(\Gamma))^\perp = S(\Gamma)^\perp \odot S(\Delta)^\perp \subset S(A) \odot S(B)$, c'est-à-dire $1 \in (S(\Delta) \lhd S(\Gamma)) \lhd S(A \odot B) = S((\Delta; \Gamma); A \odot B)$.
– La preuve se termine par la règle $\otimes$: argument similaire, en utilisant le fait que $G^\perp \subset H$ ssi $1 \in G \parr H$.
– La preuve se termine par la règle de déréliction: par induction $S(\Gamma)^\perp \subset S(A)$, et par la Propriété 5.2.14 (traduite en termes duaux de "?") $S(A) \subset S(?A)$, cqfd.
– La preuve se termine par la règle de promotion: conséquence immédiate de la monotonie de ! (Propriété 5.2.14 (iii)).

– La preuve se termine par la règle de contraction: immédiat car $S(?A) = S(?A \parr ?A)$ (Propriété 5.2.14 (vi)).
– La preuve se termine par la règle d'affaiblissement: conséquence immédiate de $\perp \subset S(?A)$ (Propriété 5.2.14 (vi)), et $\perp$ neutre pour $\parr$ (Propriété 5.2.9 (i)).
– La preuve se termine par la règle:

$$\frac{\vdash \Gamma, (?\Delta, ?\Sigma)}{\vdash \Gamma, (?\Delta; ?\Sigma)}$$

Conséquence de $S(?\Delta \triangleleft ?\Sigma) = S(?\Delta \parr ?\Sigma)$ (Propriété 5.2.14 (vii)).
– La preuve se termine par une introduction de $\perp$: comme pour l'affaiblissement.

■

Théorème 5.2.17 (Complétude) *Si un séquent est valide, alors il est prouvable dans le calcul des séquents.*

Preuve. On définit l'espace de phases enrichi suivant:
– P est l'ensemble des contextes, modulo l'associativité de ";" et "," et la commutativité de "," (autrement dit l'ensemble des ordres série-parallèles étiquetés par des formules), $\Gamma \cdot \Delta$ est la composition séquentielle $(\Gamma; \Delta)$, $\Gamma \star \Delta$ est la composition parallèle (Γ, Δ), l'unité est $()$,
– l'ordre $\leq$ est le plus petit ordre tel que: $(\Gamma, \Delta) \leq (\Gamma; \Delta)$ pour tous $\Gamma, \Delta \in P$; et $\Gamma \leq \Gamma'$ et $\Delta \leq \Delta'$ impliquent $(\Gamma; \Delta) \leq (\Gamma'; \Delta')$ et $(\Gamma, \Delta) \leq (\Gamma', \Delta')$,
– $\perp_P$ est l'ensemble des contextes Γ tels que $\vdash \Gamma$ est prouvable dans le calcul des séquents,
– K est l'ensemble des contextes de la forme $?\Gamma$ (où Γ est un contexte quelconque).

Par la règle d'entropie et la règle

$$\frac{\vdash \Gamma, \Delta}{\vdash \Gamma; \Delta}$$

$\perp_P$ satisfait les axiomes de la Définition 5.2.1.

Par ailleurs K satisfait les axiomes de la Définition 5.2.12: en effet K contient $1 = ()$ et est clos par $\cdot$ et $\star$; par la règle d'affaiblissement, $?\Gamma \in \perp^\perp = \mathbf{1}$; par la règle

$$\frac{\vdash \Gamma, (?\Delta, ?\Sigma)}{\vdash \Gamma, (?\Delta; ?\Sigma)}$$

et son inverse (via la règle d'entropie), $?\Delta \cdot ?\Sigma \in \{?\Delta \star ?\Sigma\}^{\perp\perp}$ et $?\Delta \star ?\Sigma \in \{?\Delta \cdot ?\Sigma\}^{\perp\perp}$; enfin par la règle de contraction, $?\Gamma \in \{?\Gamma \cdot ?\Gamma\}^{\perp\perp}$ et $?\Gamma \in \{?\Gamma \star ?\Gamma\}^{\perp\perp}$.

Donc on a bien défini un espace de phases enrichi.

Soit $Pr(A) = \{\Gamma \in P \mid \vdash \Gamma, A \text{ est prouvable dans le calcul des séquents}\}$.

Les $Pr(A)$ sont des faits, précisément $Pr(A) = Pr(A^\perp)^\perp$ (la preuve est la même que dans [17]). On définit une structure de phase S en posant $S(p) = Pr(p)$ pour tout symbole propositionnel p.

On prouve ensuite facilement comme dans le cas commutatif, par induction sur A, que $S(A) = Pr(A)$: cela revient à prouver des commutations du type $Pr(A \otimes B) = Pr(A) \otimes Pr(B)$. Regardons le cas des exponentiels:
– $?Pr(A) = (Pr(A)^\perp \cap K)^\perp = (Pr(A^\perp) \cap K)^\perp$. Soit $\Gamma \in Pr(?A)$ et $?\Delta \in Pr(A^\perp)$: on a $?\Delta \in Pr(!A^\perp)$ par la règle de promotion, d'où par la règle de coupure $\Gamma, ?\Delta \in \perp$, ce qui montre que $Pr(?A) \subset ?Pr(A)$. Réciproquement par la règle de déréliction, $?A \in Pr(A^\perp)$, et par ailleurs $?A \in K$, donc si $\Gamma \in ?Pr(A)$, alors $\Gamma, ?A \in \perp$, c'est-à-dire $\Gamma \in Pr(?A)$, cqfd.
– $S(!A)^\perp = ?S(A^\perp) = ?Pr(A^\perp) = Pr(?A^\perp) = Pr((!A)^\perp) = Pr(!A)^\perp$, donc $S(!A) = Pr(!A)$.

Finalement si $\vdash \Gamma$ est un séquent valide, $1 = () \in S(\Gamma) = Pr(\Gamma)$ et donc $\vdash \Gamma$ est prouvable. ■

Remarques:
▶ On voit que la sémantique des phases que nous avons définie, quand on se limite aux connecteurs $⅋$ et $\otimes$ (resp. $\lhd$ et $\odot$), est la sémantique des phases de la logique linéaire commutative (resp. cyclique). Pour les exponentiels, il y a une petite différence avec [58]: ? n'est pas centrale, donc un "?" commute seulement avec un autre "?", c'est juste ce qu'impose l'égalité $?A \lhd ?B = ?(A \oplus B)$ de la propriété 5.2.14.

▶ On s'est concentré ici sur un choix particulier de règle d'entropie adapté à la sémantique des langages `cc`. Il peut être intéressant de définir un calcul des séquents avec une règle d'entropie plus large, où par exemple la relation sémantique $\leq$ serait l'inclusion entre ordres série-parallèles. La sémantique des phases s'adapte sans problème.

5.3 Elimination des coupures

Comme dans [41], on peut utiliser la sémantique des phases pour prouver l'élimination des coupures.

Théorème 5.3.1 *Si un séquent est valide, alors il est prouvable dans le calcul des séquents sans coupure.*

Preuve. On définit l'espace de phases enrichi suivant:
– P' est l'ensemble des ordres série-parallèles étiquetés par des formules, $\Gamma \cdot \Delta$ est la composition séquentielle renversée $(\Delta; \Gamma)$, $\Gamma \star \Delta$ est la composition parallèle (Γ, Δ), l'unité est $()$,

– l'ordre $\leq$ est le plus petit ordre tel que: $(\Gamma, \Delta) \leq (\Gamma; \Delta)$ pour tous $\Gamma, \Delta \in P'$; et $\Gamma \leq \Gamma'$ et $\Delta \leq \Delta'$ impliquent $(\Gamma; \Delta) \leq (\Gamma'; \Delta')$ et $(\Gamma, \Delta) \leq (\Gamma', \Delta')$,
– $\perp_{P'}$ est l'ensemble des contextes Γ tels que $\vdash \Gamma$ est prouvable dans le calcul des séquents *sans coupure*,
– K est l'ensemble des contextes de la forme $?\Gamma$ (où Γ est un contexte quelconque).

$\perp_{P'}$ satisfait encore les axiomes de la Définition 5.2.1, et K satisfait les axiomes de la Définition 5.2.12. Donc on a bien défini un espace de phases enrichi.

$\{A\}^\perp = \{\Gamma \in P' \mid \vdash \Gamma, A \text{ est prouvable dans le calcul des séquents sans coupure}\}$.

Les $\{A\}^\perp$ sont des faits, donc on peut définir une structure de phase S' en posant $S'(p) = \{p\}^\perp$ pour tout symbole propositionnel positif p.

On prouve ensuite par induction sur A, que $S'(A) \subset \{A\}^\perp$:
– pour un symbole propositionnel p, c'est clair,
– pour le dual $p^\perp$ d'un symbole propositionnel positif, on a $S'(p^\perp) = S'(p)^\perp = \{p\}^{\perp\perp} \subset \{p^\perp\}^\perp$ car $p \in \{p^\perp\}^\perp$ (axiome),
– $S'(A \parr B) = S'(A) \parr S'(B) = (S'(B)^\perp \star S'(A)^\perp)^\perp \subset (\{B\}^{\perp\perp} \star \{A\}^{\perp\perp})^\perp \subset (\{B\} \star \{A\})^\perp \subset \{A \parr B\}^\perp$ par la règle $\parr$,
– de même $S'(A \lhd B) \subset \{A \lhd B\}^\perp$ par la règle $\lhd$,
– $S'(A \otimes B) = (S'(A) \star S'(B))^{\perp\perp} \subset (\{A\}^\perp \star \{B\}^\perp)^{\perp\perp} \subset \{A \otimes B\}^\perp$ car $\{A\}^\perp \star \{B\}^\perp \subset \{A \otimes B\}^\perp$ par la règle $\otimes$,
– de même $S'(A \odot B) \subset \{A \odot B\}^\perp$ par la règle $\odot$,
– $S'(A\&B) = S'(A) \cap S'(B) \subset \{A\}^\perp \cap \{B\}^\perp \subset \{A\&B\}^\perp$ par la règle $\&$,
– $S'(A \oplus B) = (S'(A) \cup S'(B))^{\perp\perp} \subset (\{A\}^\perp \cup \{B\}^\perp)^{\perp\perp} \subset \{A \oplus B\}^\perp$ car $\{A\}^\perp \cup \{B\}^\perp \subset \{A \oplus B\}^\perp$ par les règles $\oplus$,
– $S'(\perp) = \perp \subset \{\perp\}^\perp$ par la règle $\perp$,
– $S'(\mathbf{1}) = \perp^\perp \subset \{\mathbf{1}\}^\perp$ car $\mathbf{1} \in \perp$ par la règle $\mathbf{1}$,
– $S'(\top) = P' = \{\top\}^\perp$ par la règle $\top$,
– $S'(\mathbf{0}) = P'^\perp \subset \{\mathbf{0}\}^\perp$,
– $S'(?A) = (S'(A)^\perp \cap K)^\perp \subset (\{A\}^{\perp\perp} \cap K)^\perp \subset \{?A\}^\perp$ car $?A \in K$ et $\{A\}^\perp \subset \{?A\}^\perp$ par la règle de déréliction,
– $S'(!A) = (S'(A) \cap K)^{\perp\perp} \subset (\{A\}^\perp \cap K)^{\perp\perp} \subset \{!A\}^\perp$ car $\{A\}^\perp \cap K \subset \{!A\}^\perp$ par la règle de promotion.

Finalement si $\vdash \Gamma$ est un séquent valide, $1 = () \in S'(\Gamma) \subset \{\Gamma\}^\perp$ et donc $\vdash \Gamma$ est prouvable dans le calcul des séquents sans coupure. ■

Corollaire 5.3.2 *Si un séquent est prouvable, alors il est prouvable dans le calcul des séquents sans coupure.*

Preuve. Conséquence immédiate de la correction de la sémantique des phases (Théorème 5.2.16) et de sa complétude vis-à-vis du calcul des séquents sans coupure (Théorème 5.3.1). ■

5.4 Règles structurelles implicites

Le calcul des séquents avec toutes les règles structurelles explicites de la section 5.1 n'est pas entièrement satisfaisant pour plusieurs raisons, en particulier:

- Une preuve d'un séquent ne contenant que des connecteurs non-commutatifs utilise des ",", par exemple

$$\dfrac{\dfrac{\vdash A^{\perp}, A}{\vdash A^{\perp}; A}}{\vdash A^{\perp} \lhd A}\ .$$

- L'élimination des coupures introduit des règles structurelles (réversibles) dont on aimerait bien se passer.

Se pose donc naturellement la question de savoir quel est le calcul des séquents, avec règles structurelles réversibles (associativité, échange, la règle

$$\frac{\vdash \Gamma, \Delta}{\vdash \Gamma; \Delta}$$

et son inverse qui est un cas particulier d'entropie) implicites, sous-jacent au calcul de la section 5.1.

On pourrait penser résoudre le problème en ajoutant $\vdash A^{\perp}; A$ comme axiome et en augmentant le nombre de règles logiques, à la manière indiquée en section 5.1, avec des imbrications de contextes:

$$\frac{\vdash \Gamma, A \qquad \vdash \Delta[B]}{\vdash \Delta[\Gamma; A \odot B]}\ \ldots$$

et en diminuant le nombre de règles structurelles, en particulier en éliminant la règle

$$\frac{\vdash \Gamma, \Delta}{\vdash \Gamma; \Delta}$$

qui semble aller contre l'intuition (elle renverse l'entropie), avec par exemple l'idée d'arriver à se ramener à un calcul portant finalement sur des ordres série-parallèles.

Mais en fait pour prouver l'associativité des connecteurs multiplicatifs, cette dernière règle est essentielle, ce qui signifie que la structure mathématique sous-jacente d'un séquent, invariante par les règles structurelles réversibles, n'est pas un ordre série-parallèle (contrairement au calcul intuitionniste, voir la section 5.6 et [11]).

Nous présentons dans cette section la version du calcul des séquents mixte multiplicatif (la partie la plus intéressante) avec toutes les règles structurelles réversibles implicites.

5.4.1 Ordres cycliques

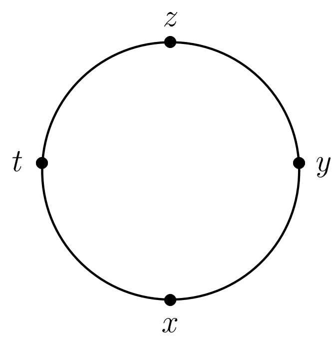
Notre solution repose sur une classe de relations ternaires qui généralise les graphes orientés cycliques: une telle relation R sur un ensemble E est cyclique $(R(x,y,z) \Rightarrow R(y,z,x) \Rightarrow R(z,x,y))$, et a la propriété que pour tout x dans E, la relation binaire $\langle R,x\rangle$ définie sur $E \setminus \{x\}$ par $\langle R,x\rangle(y,z)$ ssi $R(x,y,z)$ est un ordre, mais pas nécessairement total; R exprime donc une sorte d'"orientation" sur E.

Définition 5.4.1 (Ordre cyclique strict) *Soit E un ensemble. Un* ordre cyclique strict *sur E est une relation ternaire R*
– cyclique*: $\forall x,y,z \in E, R(x,y,z)$ implique $R(y,z,x)$,*
– anti-réflexive*: $\forall x,y \in E$, non $R(x,x,y)$,*
– transitive*: $\forall x,y,z,t \in E$, $R(x,y,z)$ et $R(z,t,x)$ impliquent $R(y,z,t)$,*
– étalée[1]*: $\forall x,y,z,t \in E$, $R(x,y,z)$ implique ($R(x,y,t)$ ou $R(y,z,t)$ ou $R(z,x,t)$).*

Un ordre cyclique strict R sur E est dit total *lorsque $\forall x,y,z \in E$, $x \neq y$ et $y \neq z$ et $z \neq x$ impliquent $R(x,y,z)$ ou $R(z,y,x)$.*

Remarques:
▶ Si R est un ordre cyclique (strict), $R(x,y,z)$ peut être lu "y est entre x et z", idée qu'on retrouve dans l'axiome de cyclicité.

▶ La transitivité impose aussi naturellement que $R(x,y,z)$ et $R(z,t,x)$ impliquent $R(t,x,y)$.



Lemme 5.4.2 *Un ordre cyclique strict R sur un ensemble E est* anti-symétrique au sens strict*: $\forall x,y,z \in E$, non $R(x,y,z)$ ou non $R(z,y,x)$.*

Preuve. Soit R un ordre cyclique strict. $R(x,y,z)$ et $R(x,z,y)$ ssi $R(x,y,z)$ et $R(z,y,x)$ (cyclicité), ce qui implique $R(y,x,y)$ (transitivité) c'est-à-dire $R(y,y,x)$, ce qui n'est pas. ■

[1]Les ordres cycliques ont été étudiés, sans la condition d'étalement, par V. Novak: voir par exemple [40]. Nous avons eu connaissance de ces travaux en fin de rédaction.

Définition 5.4.3 (Ordre cyclique large) *Soit E un ensemble. Une* ordre cyclique large *sur E est une relation ternaire R*
– réflexive*: $\forall x, y \in E$, $R(x, x, y)$,*
– anti-symétrique au sens large*: $\forall x, y, z \in E$, $R(x, y, z)$ et $R(x, z, y)$ impliquent $x = y$ ou $y = z$ ou $z = x$,*
– *transitive,*
– *étalée.*

Un ordre cyclique large R sur E est dit total *lorsque $\forall x, y, z \in E$, $R(x, y, z)$ ou $R(z, y, x)$.*

Lemme 5.4.4 *Un ordre cyclique large est cyclique.*

Preuve. Soit R un ordre cyclique large. Par réflexivité $R(z, z, x)$, donc $R(x, y, z)$ implique $R(z, x, y)$ par transitivité, cqfd. ∎

Théorème 5.4.5 *Soient R un ordre cyclique strict (resp. large) sur E et $x \in E$. La relation binaire R_x, définie sur $E \setminus \{x\}$ par $R_x(y, z)$ ssi $R(x, y, z)$, est un ordre strict (resp. large).*

Preuve. Si R est un ordre cyclique strict, R_x est bien
– anti-réflexive: $R_x(y, z)$ ssi $R(x, y, z)$, implique $y \neq z$ (anti-réflexivité de R),
– anti-symétrique au sens strict: $R_x(y, z)$ et $R_x(z, y)$, ssi $R(x, y, z)$ et $R(x, z, y)$, ce qui n'est pas (Lemme 5.4.2),
– transitive: $R_x(y, z)$ et $R_x(z, t)$ ssi $R(x, y, z)$ et $R(x, z, t)$, ssi $R(x, y, z)$ et $R(z, t, x)$ (cyclicité de R), impliquent $R(t, x, y)$ (transitivité de R), ssi $R(x, y, t)$ ssi $R_x(y, t)$.

Si R est un ordre cyclique large, R_x est bien
– réflexive: $R(x, y, y)$ (réflexivité de R) d'où $R_x(y, y)$,
– anti-symétrique au sens large: $R_x(y, z)$ et $R_x(z, y)$ ssi $R(x, y, z)$ et $R(z, y, x)$, impliquent $y = z$ car $x \neq y$ et $x \neq z$ (Lemme 5.4.4),
– transitive, comme dans le cas strict. ∎

Remarque:
▶ Pour le Théorème ci-dessus, on n'a pas besoin du caractère étalé de l'ordre cyclique.

Exemples:
▶ La relation ternaire vide sur un ensemble E est un ordre cyclique strict sur E.

▶ Les graphes orientés cycliques finis:

Proposition 5.4.6 *L'ensemble des graphes orientés cycliques finis est isomorphe à l'ensemble des ordres cycliques stricts totaux finis.*

Preuve. A tout graphe orienté cyclique fini $x_1 \to x_2 \to \cdots \to x_n \to x_1$, on associe la relation R définie sur $\{x_1, \ldots, x_n\}$ par: $R(x_i, x_j, x_k)$ ssi $(k-j)(j-i)(i-k) < 0$ c'est-à-dire $i < j < k$ ou $k < i < j$ ou $j < k < i$. R est clairement cyclique (c'est la clôture cyclique de la relation ρ: $\rho(x_i, x_j, x_k)$ ssi $i < j < k$), anti-réflexive et totale. Elle est transitive: $R(x_i, x_j, x_k)$ et $R(x_k, x_l, x_i)$, ssi ($i < j < k$ ou $k < i < j$ ou $j < k < i$) et ($k < l < i$ ou $i < k < l$ ou $l < i < k$), ssi ($i < j < k < l$ ou $l < i < j < k$ ou $k < l < i < j$ ou $j < k < l < i$), implique ($l < i < j$ ou $j < l < i$ ou $i < j < l$) c'est-à-dire $R(x_l, x_i, x_j)$. Donc R est un ordre cyclique strict total.

Réciproquement, soit R un ordre cyclique strict total sur un ensemble fini $\{x_1, \ldots, x_n\}$. On lui associe un graphe orienté dont les sommets sont les x_i, et les arcs: $x \to y$ pour $x, y \in E$ tels que $y \neq x$ et $\forall z \in E$, non $R(x, z, y)$. Dans le cas où $n = 0$ ou $n = 1$, on a bien un graphe cyclique (aucun arc). Sinon soit x quelconque dans E, il existe un unique $y \neq x$ tel que $x \to y$:
– unicité: si y et y' sont deux candidats distincts, R est totale donc $R(x, y, y')$ ou $R(x, y', y)$, ce qui fournit une contradiction;
– existence: si $\forall y \in E, y \neq x \Rightarrow \exists z, R(x, z, y)$, on part de $y \neq x$ quelconque dans E ($n \geq 2$) et par anti-réflexivité de R, on a une suite infinie $(y_i)_{i \in \mathbb{N}}$ d'éléments de E tous distincts tels que $\forall i \in \mathbb{N}, R(x, y_{i+1}, y_i)$, d'où une contradiction.

De même il existe un unique $y \neq x$ tel que $y \to x$. Donc ce graphe est constitué de cycles. Maintenant il y a un chemin entre deux sommets x et y ssi il existe z tel que $R(x, z, y)$, et c'est bien le cas puisque R est supposée totale. Cela suffit pour conclure que c'est un graphe cyclique, et les deux applications sont bien inverses l'une de l'autre. ■

▶ Plus généralement, tout ordre détermine un ordre cyclique:

Proposition 5.4.7 *Soit $(E, \ll)$ un ensemble ordonné au sens strict (resp. large). On définit la relation binaire $\overset{z}{\ll}$ (étant donné z quelconque dans E) et la relation ternaire $\overline{\ll}$ sur E par:*

– *$x \overset{z}{\ll} y$ ssi $x \ll y$ et z n'est pas comparable avec x ni y,*

– *$\overline{\ll}(x, y, z)$ ssi $x \ll y \ll z$ ou $y \ll z \ll x$ ou $z \ll x \ll y$ ou $x \overset{z}{\ll} y$ ou $y \overset{x}{\ll} z$ ou $z \overset{y}{\ll} x$.*

$\overline{\ll}$ est un ordre cyclique strict (resp. large) sur E.

Preuve. Si $(E, \ll)$ est ordonné strictement, alors $\overline{\ll}$ est en effet:

1. cyclique: c'est clair;

2. anti-réflexive: comme $\ll$ est ordre strict, on n'a ni $x \ll x$ ni $x \overset{x}{\ll} y$, donc on n'a pas $\overline{\ll}(x, x, y)$;

3. transitive: $\overline{\ll}(x, y, z)$ et $\overline{\ll}(z, t, x)$ ssi ($x \ll y \ll z$ ou $y \ll z \ll x$ ou $z \ll x \ll y$ ou $x \overset{z}{\ll} y$ ou $y \overset{x}{\ll} z$ ou $z \overset{y}{\ll} x$) et ($z \ll t \ll x$ ou $t \ll x \ll z$ ou

$x \ll z \ll t$ ou $z \overset{x}{\ll} t$ ou $t \overset{z}{\ll} x$ ou $x \overset{t}{\ll} z$). Si $x \ll y \ll z$, alors $t \ll x \ll y \ll z$ ou $x \ll y \ll z \ll t$ ou t incomparable avec x, y et z, et on a bien dans tous les cas $\overline{\ll}(t, x, y)$. Les cas où $y \ll z \ll x$ ou $z \ll x \ll y$ sont identiques (permutation circulaire).

Si $x \overset{z}{\ll} y$, alors soit $z \overset{x}{\ll} t$ soit $t \overset{z}{\ll} x$. Dans le premier cas, $x \ll y$ et t est incomparable avec x et y ($t \ll y \Rightarrow z \ll y$ contredit $x \overset{z}{\ll} y$, et $y \ll t \Rightarrow x \ll t$ contredit $z \overset{x}{\ll} t$), d'où $x \overset{t}{\ll} y$. Dans le second cas, $t \ll x \ll y$, et donc de nouveau $\overline{\ll}(t, x, y)$. Les cas où $y \overset{x}{\ll} z$ ou $z \overset{y}{\ll} x$ sont identiques (permutation circulaire).

4. étalée: supposons $\overline{\ll}(x, y, z)$ et soit $t \in E$. Si $x \ll y \ll z$, alors ou bien $t \ll y$ (et donc $t \ll y \ll z$, d'où $\overline{\ll}(y, z, t)$), ou bien $y \ll t$ (et donc $x \ll y \ll t$, d'où $\overline{\ll}(x, y, t)$), ou bien y et t sont incomparables (et dans ce cas soit t est incomparable avec x, donc $\overline{\ll}(x, y, t)$, soit t est incomparable avec z, donc $\overline{\ll}(y, z, t)$, soit $x \ll t \ll z$, donc $\overline{\ll}(z, x, t)$).

Si $x \overset{z}{\ll} y$, alors ou bien $t \ll x \ll y$ (d'où $\overline{\ll}(x, y, t)$), ou bien $x \ll t \ll y$ (d'où $x \overset{z}{\ll} t$ et aussi $t \overset{z}{\ll} y$), ou bien $x \ll y \ll t$ (d'où $\overline{\ll}(x, y, t)$), ou bien $x \ll t$ et t et y sont incomparables (d'où $\overline{\ll}(z, x, t)$ si $z \not\ll t$, ou $\overline{\ll}(y, z, t)$ si $z \ll t$), ou bien t et x sont incomparables et $t \ll y$ (et on raisonne comme dans le cas précédent), ou bien t est incomparable avec x et y (d'où $\overline{\ll}(x, y, t)$).

Si $(E, \ll)$ est ordonné au sens large, alors $\overline{\ll}$ est bien

1. réflexive: on a bien $\overline{\ll}(x, x, y)$ puisque $x \overset{y}{\ll} x$ ($\ll$ réflexive) ou $x \ll y$ ou $y \ll x$;

2. anti-symétrique au sens large: $\overline{\ll}(x, y, z)$ ssi $x \ll y \ll z$ ou $y \ll z \ll x$ ou $z \ll x \ll y$ ou $x \overset{z}{\ll} y$ ou $y \overset{x}{\ll} z$ ou $z \overset{y}{\ll} x$. Considérons le cas où $x \ll y \ll z$ (les cas $y \ll z \ll x$ et $z \ll x \ll y$ étant similaires). Par définition, $\overline{\ll}(z, y, x)$ ssi $z \ll y \ll x$ ou $y \ll x \ll z$ ou $x \ll z \ll y$ ou $z \overset{x}{\ll} y$ ou $y \overset{z}{\ll} x$ ou $x \overset{y}{\ll} z$. Les trois derniers cas sont interdits par $x \ll y \ll z$. Si $z \ll y \ll x$, par anti-symétrie au sens large de $\ll$, on a $x = y = z$. Si $y \ll x \ll z$, alors $x = y$ et si $x \ll z \ll y$, alors $y = z$.

Considérons le cas où $x \overset{z}{\ll} y$ (les cas $y \overset{x}{\ll} z$ et $z \overset{y}{\ll} x$ étant similaires). Les cas $z \ll y \ll x$, $y \ll x \ll z$, $x \ll z \ll y$, $z \overset{x}{\ll} y$ et $x \overset{y}{\ll} z$ sont impossibles. Il reste $y \overset{z}{\ll} x$, et alors $x \ll y$ et $y \ll x$, d'où $x = y$. cqfd;

3 et **4.** transitive et étalée: la preuve est la même que dans le cas strict. Donc $\overline{\ll}$ est un ordre cyclique large sur E. ■

En fait tout ordre cyclique provient d'un ordre. Déjà:

Lemme 5.4.8 *Soient $\ll_1$ et $\ll_2$ deux relations d'ordres strictes (resp. larges) sur des ensembles disjoints E_1 et E_2. En notant* **;** *(resp.* **,***) la composition série (resp. parallèle) d'ordres, on a $\overline{\ll_1 \,;\, \ll_2} = \overline{\ll_1 \,,\, \ll_2} = \overline{\ll_2 \,;\, \ll_1}$.*

Preuve. Si $x, y \in E_1$, alors $x(\ll_1 ; \ll_2)y$ ssi $x \ll_1 y$, donc si $x, y, z \in E_1$, $\overline{\ll_1 ; \ll_2}(x, y, z)$ ssi $\overline{\ll_1}(x, y, z)$ ssi $\overline{\ll_1 , \ll_2}(x, y, z)$.

De même pour $x, y, z \in E_2$.

Si $x, y \in E_1$ et $z \in E_2$, alors $\overline{\ll_1 , \ll_2}(x, y, z)$ ssi $x \ll_1 y$ ssi $x\ (\ll_1 ; \ll_2)\ y\ (\ll_1 ; \ll_2)\ z$ ssi $\overline{\ll_1 ; \ll_2}(x, y, z)$. ■

Maintenant

Théorème 5.4.9 *Soient R un ordre cyclique strict (resp. large) sur un ensemble E, $a \in E$, et $\ll$ l'un des trois ordres stricts (resp. larges) induits sur E par R_a en prenant a isolé, minimal ou maximal. Alors $\overline{\ll} = R$.*

Preuve. D'après le Lemme 5.4.8, les trois choix pour $\ll$ donnent bien le même ordre cyclique $\overline{\ll}$. Considérons donc juste le cas où a est minimal.

Si $x = a$ ou $y = a$ ou $z = a$, on a $R(x, y, z)$ ssi $\overline{\ll}(x, y, z)$ par définition de R_a.

Soient donc $x, y, z \in E$, tous différents de a, tels que $R(x, y, z)$. Comme R est étalée, on a $R(x, y, a)$ ou $R(y, z, a)$ ou $R(z, x, a)$, c'est-à-dire $x\ R_a\ y$ ou $y\ R_a\ z$ ou $z\ R_a\ x$, d'où $a \ll x \ll y$ ou $a \ll y \ll z$ ou $a \ll z \ll x$. Par exemple $a \ll x \ll y$ (les deux autres cas sont similaires). Si $y \ll z$ ou $z \ll x$, alors évidemment $\overline{\ll}(x, y, z)$. D'autre part, si $x \ll z$ alors $R(a, x, z)$, et comme $R(x, y, z)$, on a $R(a, y, z)$ par transitivité, donc $y \ll z$. De même si $z \ll y$, alors $z \ll x$. Donc le seul cas restant possible est z incomparable avec x et y, et on a bien encore $\overline{\ll}(x, y, z)$.

Réciproquement, soient $x, y, z \in E$, tous différents de a, tels que $\overline{\ll}(x, y, z)$, c'est-à-dire $x \ll y \ll z$ ou $y \ll z \ll x$ ou $z \ll x \ll y$ ou $x \overset{z}{\ll} y$ ou $y \overset{x}{\ll} z$ ou $z \overset{y}{\ll} x$. Dans le premier cas, on a $R_a(x, y)$ et $R_a(y, z)$ (par définition de $\ll$), d'où $R(a, x, y)$ et $R(a, y, z)$ (par définition de R_a), donc $R(x, y, z)$ puisque R est transitive. De même pour les deux cas suivants.

Si $x \overset{z}{\ll} y$, alors on a en particulier $R(a, x, y)$. Comme R est étalée, cela impose $R(a, x, z)$ ou $R(x, y, z)$ ou $R(y, a, z)$. Mais par ailleurs $x \overset{z}{\ll} y$ implique entre autres non $R(a, z, y)$ et non $R(a, x, z)$. Par conséquent $R(x, y, z)$, cqfd. Les cas $y \overset{x}{\ll} z$ et $z \overset{y}{\ll} x$ sont identiques. ■

Remarque:

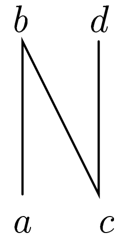
▶ Pour le Théorème ci-dessus, le caractère étalé de l'ordre cyclique est essentiel: par exemple la relation ternaire $R(a, b, c), R(b, c, a), R(c, a, b)$ (c'est tout) sur l'ensemble à 4 éléments $E = \{a, b, c, d\}$ est cyclique anti-réflexive et transitive, mais pas étalée, et de fait elle ne provient d'aucun ordre sur E.

5.4.2 Ordres cycliques série-parallèles

Un ordre cyclique provient en général de plusieurs ordres, mais pas toujours d'un ordre série-parallèle. Quelle est la configuration à interdire pour qu'un ordre cyclique provienne d'un ordre série-parallèle? Déjà, si N est le plus petit ordre non série-parallèle (défini sur $\{a,b,c,d\}$ par: $a\ N\ b$, $c\ N\ b$ et $c\ N\ d$) son ordre cyclique associé $\overline{N}$ est la clôture cyclique de $\{(a,b,d),(a,c,d)\}$, mais $\overline{N}$ provient aussi d'un ordre série-parallèle: $a<b<d$ et $a<c<d$.

Rappelons d'abord la caractérisation des ordres série-parallèles (voir par exemple [56]):

Théorème 5.4.10 *Soit R un ordre strict sur E. R est un ordre série-parallèle ssi la restriction de R à tout sous-ensemble à 4 éléments $\{a,b,c,d\}$ est différente de l'ordre $N(a,b,c,d)=\{(a,b),(c,b),(c,d)\}$.*



Maintenant:

Définition 5.4.11 (Compositions d'ordres cycliques) *Soient E,F deux ensembles non vides d'intersection $\{a\}$, et R,S deux ordres cycliques stricts sur E,F respectivement. On définit:*
- $R\otimes_a S \ = \ \overline{S_a, \{a\}, R_a}$,
- $R\odot_a S \ = \ \overline{S_a; \{a\}; R_a}$,

  deux ordres cycliques sur $E\cup F$, et
- $R\asymp_a S \ = \ (R\otimes_a S)\restriction_{(E\cup F)\setminus\{a\}} \ = \ (R\odot_a S)\restriction_{(E\cup F)\setminus\{a\}}$,

  un ordre cyclique sur $(E\cup F)\setminus\{a\}$.

Soient E,F deux ensembles non vides disjoints, $a\in E, b\in F$ et R,S deux ordres cycliques stricts sur E,F respectivement. On note:
- $R\otimes_{a,b} S \ = \ (R[a\otimes b/a]) \ \otimes_{a\otimes b} \ (S[a\otimes b/b]) \ = \ \overline{S_b, \{a\otimes b\}, R_a}$,
- $R\odot_{a,b} S \ = \ (R[a\odot b/a]) \ \odot_{a\odot b} \ (S[a\odot b/b]) \ = \ \overline{S_b; \{a\odot b\}; R_a}$,

  deux ordres cycliques sur $E[a\otimes b/a]\cup F[a\otimes b/b]$, et
- $R\asymp_{a,b} S = R\asymp_a (S[a/b])$.

Définition 5.4.12 (Ordres cycliques série-parallèles) *La classe des* ordres cycliques stricts série-parallèles *est la plus petite classe d'ordres cycliques stricts contenant les singletons et paires (un ou deux éléments, relation ternaire vide) et close par $\otimes_a$ et $\odot_a$.*

Lemme 5.4.13 *Soient R un ordre cyclique strict série-parallèle sur E et $x\in E$. La relation binaire R_x du Théorème 5.4.5 est un ordre série-parallèle sur $E\setminus\{x\}$.*

Preuve. Par induction sur la construction des ordres cycliques série-parallèles, par $\otimes_a$ et $\odot_a$. Si R est un singleton ou une paire, alors c'est clair.

Supposons que $R = S \otimes_a T$, avec S (resp. T) un ordre cyclique série-parallèle sur E (resp. F).

Si $x = a$, $R_x(y,z)$ ssi ($S_a(y,z)$ et $y, z \in E$) ou ($T_a(y,z)$ et $y, z \in F$), d'où $R_x = (S_a \textbf{ , } T_a)$. Or par hypothèse d'induction, S_a et T_a sont des ordres série-parallèles, donc R_x aussi.

Si $x \neq a$, par exemple $x \in E \setminus \{a\}$ (le cas $x \in F \setminus \{a\}$ étant similaire). On a $R_x(y,z)$ ssi

– $S_x(y,z)$ et $y, z \in E$, ou
– $S_x(y,a)$ et $y \in E$ et $z \in F$, ou
– $S_x(a,z)$ et $y \in E$ et $z \in F$,

d'où $R_x = S_x[(a\textbf{,}T_a)\ /\ a]$. Donc par l'hypothèse d'induction, R_x est série-parallèle.

Supposons que $R = S \odot_a T$.

Si $x = a$, $R_x = (S_a \textbf{ ; } T_a)$, qui est bien série-parallèle.

Si $x \neq a$, par exemple $x \in E \setminus \{a\}$, on a $R_x(y,z)$ ssi

– $S_x(y,z)$ et $y, z \in E$, ou
– $S_x(y,a)$ et $y \in E$ et $z \in F$, ou
– $S_x(a,z)$ et $y \in E$ et $z \in F$, ou
– $y \in F \setminus \{a\}$ et $x \neq y$ et $z = a$,

d'où $R_x = S_x[(T_a\textbf{;}a)\ /\ a]$. Donc par l'hypothèse d'induction, R_x est série-parallèle. On procède de même si $x \in F \setminus \{a\}$. ■

Lemme 5.4.14 *Soit R un ordre cyclique strict sur E. S'il existe un ordre série-parallèle $<$ sur E tel que $\overline{<} = R$, alors pour tout $x \in E$, R_x est un ordre série-parallèle (sur $E \setminus \{x\}$).*

Preuve. La preuve est essentiellement la même que pour la Proposition 5.1.3.

L'ordre série-parallèle $<$ est représenté (de manière non unique) par un arbre α dont les feuilles sont étiquetées par des éléments de E et les nœuds par **,** ou **;**. On procède par induction sur la longueur du chemin de x à la racine de α.

– Si $\alpha = \beta\textbf{,}x$ ou $x\textbf{,}\beta$ ou $x\textbf{;}\beta$ ou $\beta\textbf{;}x$, alors $R_x = \beta$ qui est bien série-parallèle.

– Si $\alpha = (\beta_1\textbf{,}\beta_2)\textbf{,}\gamma$ et x est dans β_1, alors $<$ est aussi représenté par $\alpha' = \beta_1\textbf{,}(\beta_2\textbf{,}\gamma)$ et on applique l'hypothèse d'induction.

– Si $\alpha = (\beta_1\textbf{,}\beta_2)\textbf{,}\gamma$ et x est dans β_2,alors $<$ est aussi représenté par $\alpha' = \beta_2\textbf{,}(\beta_1\textbf{,}\gamma)$ et on applique l'hypothèse d'induction.

– Si $\alpha = (\beta_1\textbf{;}\beta_2)\textbf{,}\gamma$ et x est dans β_1, alors d'après le Lemme 5.4.8, $\overline{<} = \overline{<'}$, où $<'$ est l'ordre série-parallèle représenté par $(\beta_1\textbf{;}\beta_2)\textbf{;}\gamma$, mais aussi par $\beta_1\textbf{;}(\beta_2\textbf{;}\gamma)$, et on applique l'hypothèse d'induction.

– Si $\alpha = (\beta_1;\beta_2),\gamma$ et x est dans β_2, alors $<$ est aussi représenté par $\gamma,(\beta_1;\beta_2)$, et on procède comme ci-dessus.

– Les cas $\alpha = \beta,(\gamma_1,\gamma_2)$, $\alpha = \beta,(\gamma_1;\gamma_2)$, $\alpha = \beta;(\gamma_1,\gamma_2)$, $\alpha = \beta;(\gamma_1;\gamma_2)$ et sont parfaitement similaires. ■

Définition et Lemme 5.4.15 *On note $G(a,b,c,d,e)$ la clôture cyclique de $\{(a,b,d),(a,c,d),(b,c,e),(b,d,e),(a,c,e)\}$.*
(i) $G(a,b,c,d,e)$ est un ordre cyclique strict sur $\{a,b,c,d,e\}$.
(ii) Il n'existe pas d'ordre série-parallèle $<$ sur $\{a,b,c,d,e\}$ tel que $\overline{<} = G(a,b,c,d,e)$.

Preuve.
(i) La relation binaire $\{(b,d),(c,d),(c,e)\}$ est un ordre strict sur $\{b,c,d,e\}$. (C'est $G(a,b,c,d,e)_a$.) Soit $<$ l'ordre strict induit sur $\{a,b,c,d,e\}$ en prenant a minimal. On vérifie que $G(a,b,c,d,e) = \overline{<}$, ce qui permet de conclure par la Proposition 5.4.7.
(ii) $G(a,b,c,d,e)_a$ n'est pas série-parallèle, donc d'après le Lemme 5.4.14, $G(a,b,c,d,e)$ ne provient pas d'un ordre série-parallèle. ■

Théorème 5.4.16 *Soit R un ordre cyclique strict sur E. Les assertions suivantes sont équivalentes:*
(i) R est série-parallèle,
(ii) il existe un ordre série-parallèle $<$ sur E tel que $\overline{<} = R$,
(iii) la restriction de R à tout sous-ensemble de E à 5 éléments $\{a,b,c,d,e\}$ est différente de $G(a,b,c,d,e)$.

Preuve.
(i $\Rightarrow$ ii)
Si $E = \emptyset$, c'est clair. Sinon soit $x \in E$. Comme R est un ordre cyclique série-parallèle, l'ordre R_x est série-parallèle d'après le Lemme 5.4.13, donc aussi par exemple l'ordre $<= (x,R_x)$ sur E. D'après le Théorème 5.4.9, $\overline{<} = R$.

(ii $\Rightarrow$ i)
D'après le Lemme 5.4.13, (ii) implique: pour tout $x \in E$, R_x est un ordre série-parallèle, donc aussi (x,R_x), et par le Théorème 5.4.9, $\overline{x,R_x} = R$. Il reste à montrer que $\overline{x,R_x}$ est série-parallèle, ce qu'on fait par induction sur R_x: si R_x est vide ou un singleton, c'est clair; sinon R_x est soit de la forme $<_1 , <_2$ (et alors $\overline{x,R_x} = \overline{x, <_1} \otimes_x \overline{x, <_2}$) soit de la forme $<_1 ; <_2$, (et alors $\overline{x,R_x} = \overline{x,(<_1 ; <_2)} = \overline{x;(<_1 ; <_2)} = \overline{x, <_1} \odot_x \overline{x, <_2}$), cqfd.

(ii $\Rightarrow$ iii)
Soient $<$ un ordre série-parallèle sur E tel que $\overline{<} = R$, et $\{a,b,c,d,e\} \subset E$. Si $R \restriction_{\{a,b,c,d,e\}} = G(a,b,c,d,e)$, alors $\overline{<} \restriction_{\{a,b,c,d,e\}} = \overline{<\restriction_{\{a,b,c,d,e\}}} = G(a,b,c,d,e)$, alors que $<\restriction_{\{a,b,c,d,e\}}$ est un ordre série-parallèle, d'où une contradiction d'après le Lemme 5.4.15, cqfd.

(iii $\Rightarrow$ ii)
Soit R un ordre cyclique strict ne contenant pas de $G(.,.,.,.,.)$. Si $E = \emptyset$, le résultat est clair. Sinon soit $x \in E$. D'après le Théorème 5.4.9, $\overline{x,R_x} = R$.

Si R_x n'est pas série-parallèle, alors d'après le Théorème 5.4.10, R_x contient un $N(a,b,c,d)$, donc aussi (x,R_x). Par conséquent $\overline{x,R_x}$ contient $\{(x,a,b),(x,c,b),(x,c,d)\} \cup \{(a,b,d),(a,c,d)\}$, et donc aussi sa clôture cyclique $G(a,c,b,d,x)$, donc R contient un $G(.,.,.,.,.)$, contradiction.

Donc R_x est bien série-parallèle, de même que (x,R_x), cqfd. ■

Proposition 5.4.17 (Intersection) *Si R et S sont des ordres cycliques série-parallèles sur un ensemble E, alors $R \cap S$ est aussi un ordre cyclique série-parallèle sur E.*

Preuve. Si $E = \emptyset$, c'est clair. Sinon, soit $x \in E$: $(R \cap S)_x(y,z)$ ssi $(R \cap S)(x,y,z)$ ssi $R_x(y,z)$ et $S_x(y,z)$ ssi $(R_x \cap S_x)(y,z)$. Donc $(R \cap S)_x = R_x \cap S_x$ est série-parallèle d'après le lemme 5.4.13. D'après le théorème 5.4.9, $R \cap S = \overline{x,(R \cap S)_x}$, donc $R \cap S = \overline{x,R_x \cap S_x}$ est la traduction d'un ordre série-parallèle, et est par conséquent série-parallèle d'après le théroème 5.4.16. ■

5.4.3 Calcul des séquents multiplicatif

Définition 5.4.18 *Soit R un ordre cyclique strict sur E. Les relations binaires $\rightarrow_R$ et $\approx_R$ sur E sont définies par:*
– $x \rightarrow_R y$ ssi $\forall z \in E, z \neq x$ et $z \neq y$ impliquent $R(x,y,z)$.
– $x \approx_R y$ ssi $\forall z \in E$, non $R(x,y,z)$ et non $R(z,y,x)$.

Lemme 5.4.19 *Soient R un ordre cyclique strict sur E, et $x, y \in E$.*
(i) $x \rightarrow_R y$
ssi pour tout $z \in E, z \neq x$ et $z \neq y$ impliquent: dans R_z, x est l'unique prédécesseur de y et y est l'unique successeur de x,
ssi x est maximal dans R_y,
ssi y est minimal dans R_x.
(ii) $x \approx_R y$
ssi pour tout $z \in E, z \neq x$ et $z \neq y$ impliquent: dans R_z, x et y ont les mêmes successeurs et les mêmes prédécesseurs,
ssi x est isolé dans R_y,
ssi y est isolé dans R_x.

Preuve. (i) Si $x \rightarrow_R y$, alors dans tous les R_z ($z \neq x$ et $z \neq y$), on a $R_z(x,y)$ donc x est inférieur à y; si pour l'ordre R_z, x est inférieur à un autre $t \in E$, alors $R_z(x,t)$, mais pour ce même t, on a aussi $R(x,y,t)$, donc par transitivité de R, $R(z,y,t)$, c'est-à-dire y inférieur à t; par conséquent y est l'unique successeur de x, et de même x est l'unique prédécesseur de y.

Réciproquement si dans tous les R_z, x est l'unique prédécesseur de y et y est l'unique successeur de x, on a clairement $x \to_R y$.

Maintenant, si $x \to_R y$, alors dans R_y, on a bien x supérieur à tout $z \in E$, $z \neq x$, et dans R_x, on a bien y inférieur à tout $z \in E$, $z \neq y$.

Réciproquement si x est maximal dans R_y, alors pour tout $z \in E$, $z \neq x, y$ implique $R(y, z, x)$, donc $x \to_R y$.

De même si y est minimal dans R_x, on a bien $x \to_R y$.

(ii) Pour prouver $x \approx_R y$ ssi x est isolé dans R_y, ssi y est isolé dans R_x, on procède comme ci-dessus.

Si $x \approx_R y$ et $R(z, x, t)$, alors $R(y, x, t)$ ou $R(z, x, y)$ ou $R(z, y, t)$ puisque R est étalée, et comme on n'a ni $R(z, x, y)$ ni $R(t, y, x)$ (définition de $\approx_R$), il reste $R(z, y, t)$. Donc dans tous les R_z ($z \neq x$ et $z \neq y$), $R_z(x, t)$ implique $R_z(y, t)$, et de même $R_z(y, t)$ implique $R_z(x, t)$, $R_z(t, x)$ implique $R_z(t, y)$ et $R_z(t, y)$ implique $R_z(t, x)$.

Réciproquement si dans tous les R_z, x et y ont les mêmes successeurs et les mêmes prédécesseurs, on a clairement $x \approx_R y$. ∎

Lemme 5.4.20 *Soient R un ordre cyclique strict sur E, et $x, y \in E$ tels que $x \to_R y$ (resp. $x \approx_R y$). La relation $R[x \lhd y \;/\; x, y]$ (resp. $R[x \parr y \;/\; x, y]$) obtenue en identifiant x et y dans R est un ordre cyclique strict sur $E[x \lhd y \;/\; x, y]$ (resp. $E[x \parr y \;/\; x, y]$).*

Preuve. Considérons le cas $x \to_R y$ (le cas $x \approx_R y$ étant similaire). Si $E = \{x, y\}$, c'est clair. Sinon soit $z \in E$, $z \neq x$ et $z \neq y$. D'après le Lemme 5.4.19, x et y sont en série dans R_z, $R_z[x \lhd y \;/\; x, y]$ est un ordre strict, et par conséquent $R[x \lhd y \;/\; x, y]$ est bien un ordre cyclique strict sur $E[x \lhd y \;/\; x, y]$. ∎

Définition 5.4.21 *On définit $\prec$ comme la plus petite relation d'ordre stricte sur l'ensemble des ordres série-parallèles, close par: $(< \,\mathbf{,}\, <') \prec (< \,\mathbf{;}\, <')$.*

Soient R et S deux ordres cycliques stricts série-parallèles sur $E \neq \emptyset$. Soit $x \in E$. On note aussi $R \prec S$ pour: $R_x \prec S_x$. (C'est indépendant du choix de x.)

Définition 5.4.22 (Séquents (allégés)) *Un* séquent allégé *est un ordre cyclique strict série-parallèle R étiqueté par un ensemble Γ de formules. On note $\vdash \Gamma \; [R]$.*

Le calcul des séquents multiplicatif avec règles structurelles réversibles implicites est donné par:

Axiome - Coupure

$$\vdash A^{\perp}, A \quad [\emptyset] \qquad \frac{\vdash \Gamma, A \quad [R] \qquad \vdash A^{\perp}, \Delta \quad [S]}{\vdash \Gamma, \Delta \quad [R \asymp_{A,A^{\perp}} S]}$$

Entropie

$$\frac{\vdash \Gamma \quad [R]}{\vdash \Gamma \quad [R']} \; R' \prec R$$

Non-commutatifs

$$\frac{\vdash \Gamma, A \quad [R] \qquad \vdash \Delta, B \quad [S]}{\vdash \Delta, \Gamma, A \odot B \quad [R \odot_{A,B} S]} \qquad \frac{\vdash \Gamma, A, B \quad [R]}{\vdash \Gamma, A \lhd B \quad [R[A \lhd B/A, B]]} \; A \rightarrow_R B$$

Multiplicatifs commutatifs

$$\frac{\vdash \Gamma, A \quad [R] \qquad \vdash \Delta, B \quad [S]}{\vdash \Delta, \Gamma, A \otimes B \quad [R \otimes_{A,B} S]} \qquad \frac{\vdash \Gamma, A, B \quad [R]}{\vdash \Gamma, A \parr B \quad [R[A \parr B/A, B]]} \; A \approx_R B$$

Théorème 5.4.23 *Si $<$ est l'ordre strict série-parallèle associé à un séquent du calcul des séquents de la section 5.1, alors l'ordre cyclique strict série-parallèle $\overline{<}$ est invariant par les règles structurelles réversibles: associativités, échange et*

$$\frac{\vdash \Gamma, \Delta}{\vdash \Gamma ; \Delta} \, .$$

Preuve. L'invariance par associativités et échange est évidente. Pour la troisième règle, c'est une conséquence immédiate du Lemme 5.4.8. ∎

Théorème 5.4.24 *Soit $\vdash \Gamma$ un séquent multiplicatif (de la section 5.1), donc un ordre série-parallèle étiqueté par les formules de Γ.*

$\vdash \Gamma$ est prouvable dans le calcul avec règles structurelles explicites ssi le séquent allégé $\vdash \overline{\Gamma}$ est prouvable dans le calcul avec règles structurelles réversibles implicites.

Preuve.

($\Rightarrow$)

A une preuve π de $\vdash \Gamma$ dans le calcul des séquents (multiplicatif) avec règles structurelles explicites on associe, par induction sur la taille de π, une preuve $\overline{\pi}$ de $\vdash \overline{\Gamma}$ dans le calcul avec règles structurelles réversibles implicites.

On remplace l'ordre série-parallèle par l'ordre cyclique (strict série-parallèle) associé, et on efface les applications de règles structurelles réversibles conformément au Théorème 5.4.23. L'arbre obtenu est bien une preuve:

– pour la coupure, on a bien $\overline{\Gamma,A} \asymp_{A,A^\perp} \overline{A^\perp,\Delta} = \overline{\Gamma,\Delta}$,

– pour la règle $\odot$, on utilise $\overline{\Gamma,A} \odot_{A,B} \overline{B,\Delta} = \overline{(\Delta;\Gamma),A \odot B}$,

– pour la règle $\otimes$, on utilise $\overline{\Gamma,A} \otimes_{A,B} \overline{B,\Delta} = \overline{(\Delta,\Gamma),A \otimes B}$,

– pour les règles $\lhd$ et $⅋$, on utilise le Lemme 5.4.19,

– pour la règle d'entropie, on a bien $\overline{\Gamma,\Delta} \prec \overline{\Gamma;\Delta}$.

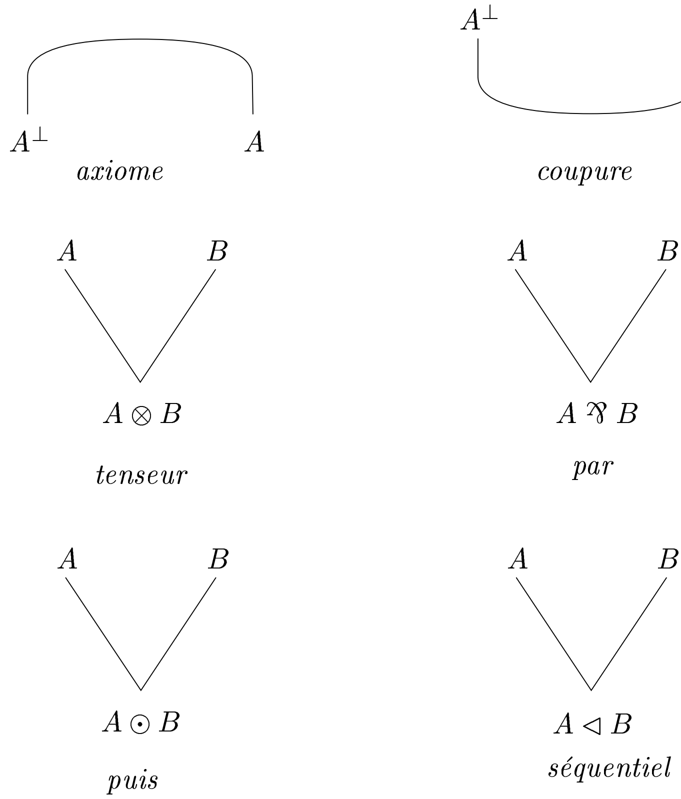
($\Leftarrow$)

Réciproquement on remplace des combinaisons d'ordres cycliques série-parallèles par des compositions d'ordres série-parallèles en fixant la formule active: en vertu du Lemme 5.4.14 et du Théorème 5.4.16, fixer un point induit bien un ordre série-parallèle. Le Lemme 5.4.23 permet toutes les applications nécessaires de règles structurelles réversibles. ∎

5.5 Réseaux de preuve multiplicatifs

Cette section est le fruit d'un travail commun avec Michele Abrusci et Jean-Yves Girard.

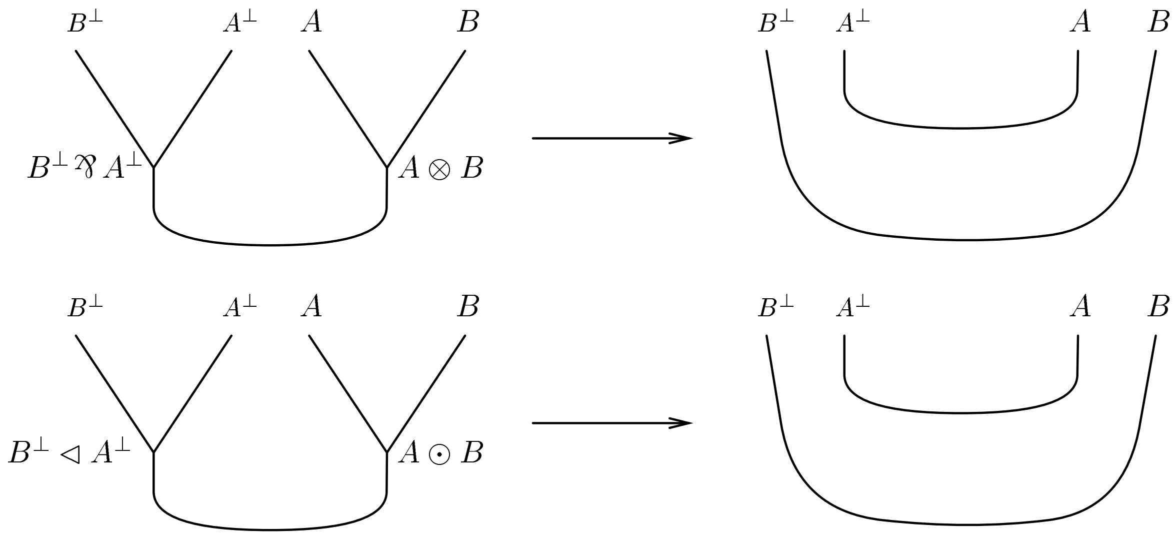
Définition 5.5.1 *Une* structure de preuve *est un graphe construit à partir de (petits) sous-graphes appelés* liens *pour lesquels on distingue des sommets* prémisses *et des sommets* conclusions*; les sommets sont étiquetés par des formules; et les liens sont:*



On demande en outre que toute occurrence de formule soit conclusion d'exactement un lien, et prémisse d'au plus un lien.

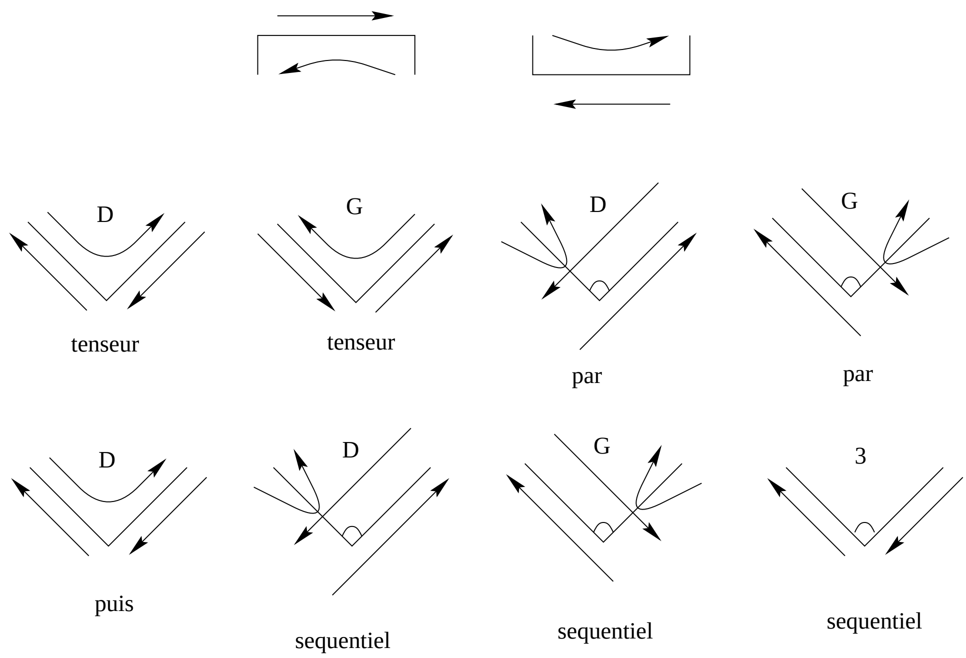
Une formule qui n'est prémisse d'aucun lien est dite terminale, *on dira aussi qu'elle est* conclusion *de la structure.*

On définit localement l'élimination des coupures dans les structures de preuves:



5.5.1 Critère de correction

Définition 5.5.2 *Les* positions d'interrupteurs *associées aux liens sont:*



Pour faire un voyage*, on fixe une position d'interrupteur pour chaque lien de la structure de preuve, on part d'une formule et on parcourt le graphe comme prescrit par les positions d'interrupteurs.*

Un voyage est dit court *si on revient au point de départ avant d'avoir parcouru toute formule A dans les deux sens, $A^\wedge$ et $A^\vee$. Il est dit* long *sinon.*

Posons $\overline{\wedge} = \vee$ et $\overline{\vee} = \wedge$. On dira qu'un voyage v contient une configuration *$ABAB$ s'il existe des sens de passage x et $y \in \{\wedge, \vee\}$ tels que $A^x, B^y, A^{\overline{x}}, B^{\overline{y}}$ apparaissent dans cet ordre cyclique dans v.*

A cause de la troisième position pour les liens $\triangleleft$, on ne peut pas exiger que les voyages soient tous longs, mais on peut quand même s'intéresser aux voyages cycliques (ou plus simplement "cycles").

Définition 5.5.3 (Réseau de preuve) *On dit qu'une structure de preuve β est un* réseau de preuve *lorsque pour toute position des interrupteurs:*

(i) il existe exactement un cycle σ,

(ii) σ contient toutes les conclusions,

(iii) σ ne contient pas de configuration $ABAB$,

(iv) étant donnés deux liens quelconques $\frac{A_1 \quad A_2}{A_1 \triangleleft A_2}$ et $\frac{B_1 \quad B_2}{B_1 \; p \; B_2}$ ($p \in \{\triangleleft, \parr\}$) de β, σ ne contient pas A_2, B_1, A_2, B_2, A_1 ou A_2, B_2, A_2, B_1, A_1 dans cet ordre cyclique.

On dit alors aussi: un réseau correct*.*

Définition 5.5.4 *On dira qu'une structure de preuve β est* correcte au sens commutatif *si la structure de preuve de LL commutative obtenue en remplaçant les $\lhd$ et $\odot$ par $\parr$ et $\otimes$ respectivement, est correcte.*

Définition 5.5.5 (Parties interne, externe, inférieure, supérieure) *Soient β un réseau de preuve, s une position d'interrupteurs, et $\frac{A_1 \quad A_2}{A_1\ p\ A_2}$ $(p \in \{\lhd, \parr\})$ un lien de β.*

La partie interne *(resp.* externe; inférieure*) de p est la partie ${A_2}^\wedge \ldots {A_1}^\vee$ (resp. ${A_1}^\wedge \ldots {A_2}^\vee$; $(A_1pA_2)^\vee \ldots (A_1pA_2)^\wedge$) de l'unique cycle déterminé par s. La* partie supérieure *de p est la réunion de la partie interne et de la partie externe.*

Les trois premières conditions du critère sont équivalentes à la correction au sens commutatif + des conditions sur les parties internes des liens $\lhd$:

Lemme 5.5.6 *Soit β une structure de preuve. β satisfait (i), (ii) et (iii) pour toute position des interrupteurs ssi β est correcte au sens commutatif, et pour une position d'interrupteurs donnée quelconque, les parties internes des liens $\lhd$ ne contiennent pas de conclusion et ne se chevauchent pas (i.e. les configurations $A_2^\wedge, B_2^\wedge, A_1^\vee, B_1^\vee$ et $A_2^\wedge, B_1^\vee, A_1^\vee, B_2^\wedge$ sont interdites).*

Preuve. $\Leftarrow$ Soit s une position d'interrupteurs. On montre, par induction sur le nombre n de liens en position 3, que s satisfait (i), (ii), (iii), et (PI) : pour tout lien $\frac{A_1 \quad A_2}{A_1 \lhd A_2}$ de β, $A_2^\wedge$ est dans le cycle ssi $A_1^\vee$ l'est.

Si $n = 0$, comme β est correcte au sens commutatif, s détermine un voyage long, qui satisfait donc (i), (ii) et (PI). De plus dans un réseau commutatif correct, les voyages ne contiennent pas de configuration $ABAB$ (preuve triviale par induction sur la construction des réseaux corrects, en vertu du théorème de séquentialisation commutatif), donc s satisfait la condition (iii).

Si $n > 0$, alors soit $\frac{A_1 \quad A_2}{A_1 \lhd A_2}$ un lien $\lhd$ de β en position 3, et plaçons-le en position D, d'où une nouvelle position des interrupteurs s'. Par hypothèse d'induction, s' satisfait (i), (ii), (iii) et (PI). Donc soit $A_2^\wedge$ et $A_1^\vee$ sont tous les deux dans l'unique cycle σ' de s', soit aucun des deux (condition PI). Dans ce dermier cas, le résultat est immédiat. Dans le premier cas, s a un unique cycle σ, obtenu en enlevant de σ' la partie interne ${A_2}^\wedge \ldots A_1^\vee$; σ contient toutes les conclusions, car $A_2^\wedge \ldots A_1^\vee$ n'en contient pas, par hypothèse; σ ne contient pas plus de configuration $ABAB$ que σ', c'est-à-dire pas du tout; enfin, le fait que les parties internes des liens $\lhd$ ne se chevauchent pas préserve la condition (PI).

(Un voyage avec positions 3 est donc obtenu à partir d'un voyage sans position 3 (disons le voyage avec tous les liens $\lhd$ en position D par exemple) en enlevant les parties internes des liens $\lhd$ en position 3.)

$\Rightarrow$ Soit β une structure de preuve satisfaisant (i), (ii) et (iii) pour toute position des interrupteurs. En particulier, en l'absence de lien $\lhd$ en position 3, on a un unique cycle, mais d'autre part, on n'a que des cycles (un voyage ne s'arrête jamais en l'absence de position 3, et le réseau est fini), donc ce cycle est un long voyage. Pour conclure que la structure est correcte au sens commutatif, il faut aussi vérifier la condition de long voyage avec les liens $\odot$ en position G: on procède par induction sur le nombre n de liens $\odot$ en position G. Le cas où $n = 0$ vient d'être traité. Si $n > 0$, alors soit $\frac{A_1 \quad A_2}{A_1 \odot A_2}$ un lien $\lhd$ de β en position G, et plaçons-le en position D, d'où une nouvelle position des interrupteurs s'. Par hypothèse d'induction, s' détermine un long voyage, qui a donc l'une des deux formes suivantes:

$$A_1^\wedge \dots A_1^\vee A_2^\wedge \dots A_2^\vee (A_1 \odot A_2)^\vee \dots (A_1 \odot A_2)^\wedge$$

$$A_1^\wedge \dots A_2^\vee (A_1 \odot A_2)^\vee \dots A_1^\vee A_2^\wedge \dots (A_1 \odot A_2)^\wedge$$

mais la deuxième contient une configuration $ABAB$ (à savoir $A_1^\wedge A_2^\vee A_1^\vee A_2^\wedge$), donc le long voyage de s' est de la première forme. On repasse le lien $\odot$ dans la position G, et on obtient le long voyage $A_1^\wedge \dots A_1^\vee (A_1 \odot A_2)^\vee \dots (A_1 \odot A_2)^\wedge A_2^\wedge \dots A_2^\vee$, cqfd.

L'absence de conclusion dans les parties internes des liens $\lhd$ est évidente: par l'absurde, si la partie interne d'un lien $\lhd$ contient une conclusion C, considérez une position d'interrupteurs s avec seulement ce lien en position 3, le cycle de s est obtenu à partir d'un voyage long en enlevant la partie interne, donc en particulier la conclusion, ce qui contredit la condition (ii).

Reste à montrer que les parties internes des liens $\lhd$ ne se chevauchent pas. Supposons au contraire que pour deux liens $\frac{A_1 \quad A_2}{A_1 \lhd A_2}$ et $\frac{B_1 \quad B_2}{B_1 \lhd B_2}$, et pour une certaine position d'interrupteurs, le cycle soit de la forme $A_2^\wedge \overset{1}{\dots} B_2^\wedge \overset{2}{\dots} A_1^\vee \overset{3}{\dots} B_1^\vee \overset{4}{\dots}$ par exemple. Notons que les deux liens $\lhd$ considérés ne sont donc pas en position 3. Si on met $\frac{A_1 \quad A_2}{A_1 \lhd A_2}$ en position 3, on obtient un cycle

$$\overset{3}{\dots} B_1^\vee \overset{4}{\dots},$$

et un segment

$$A_2^\wedge \overset{1}{\dots} B_2^\wedge \overset{2}{\dots} A_1^\vee.$$

Si ensuite on met $\frac{B_1 \quad B_2}{B_1 \lhd B_2}$ en position 3, on obtient deux segments

$$A_2^\wedge \overset{1}{\dots}\overset{4}{\dots}\overset{3}{\dots} B_1^\vee$$

et

$$B_2^\wedge \overset{2}{\dots} A_1^\vee,$$

et pas de cycle: contradiction. La seconde configuration interdite admet une preuve très similaire. ■

Théorème 5.5.7 (Stabilité par élimination des coupures) *Si β est un réseau de preuve et $\beta \to \beta'$, alors β' est un réseau de preuve.*

Preuve. Soit β un réseau de preuve et β' la structure obtenue en éliminant la coupure portant sur la paire de liens $\frac{A \quad B}{AtB}$ et $\frac{B^\perp \quad A^\perp}{B^\perp t^\perp A^\perp}$ ($t \in \{\otimes, \odot\}$ et $\otimes^\perp = \parr$, $\odot^\perp = \lhd$). On montre d'abord la stabilité de (i), (ii) et (iii) en utilisant la version équivalente du lemme 5.5.6, puis la stabilité de (iv).

1. Correction au sens commutatif. C'est déjà fait [17].

2. Absence de conclusion dans les parties internes des liens $\lhd$. Soit s une position des interrupteurs sans la position 3. Soit un autre lien $\frac{C \quad D}{C \lhd D}$.

- $t = \odot$. Le voyage (long) v dans β se déroule de l'une des deux manières suivantes, selon la position du lien $\frac{B^\perp \quad A^\perp}{B^\perp \lhd A^\perp}$:

  $A^{\perp\wedge} \overset{1}{\ldots} B^{\perp\vee} B^{\perp\wedge} \overset{2}{\ldots} A^{\perp\vee} (B^\perp \lhd A^\perp)^\vee (A \odot B)^\wedge A^\wedge \overset{3}{\ldots} A^\vee B^\wedge \overset{4}{\ldots} B^\vee (A \odot B)^\vee (B^\perp \lhd A^\perp)^\wedge$ ou

  $B^{\perp\wedge} \overset{2}{\ldots} A^{\perp\vee} A^{\perp\wedge} \overset{1}{\ldots} B^{\perp\vee} (B^\perp \lhd A^\perp)^\vee (A \odot B)^\wedge A^\wedge \overset{3}{\ldots} A^\vee B^\wedge \overset{4}{\ldots} B^\vee (A \odot B)^\vee (B^\perp \lhd A^\perp)^\wedge$. Après réduction, le voyage long est v':

  $$A^{\perp\wedge} \overset{1}{\ldots} B^{\perp\vee} B^\wedge \overset{4}{\ldots} B^\vee B^{\perp\wedge} \overset{2}{\ldots} A^{\perp\vee} A^\wedge \overset{3}{\ldots} A^\vee.$$

  Par hypothèse 1 ne contient pas de conclusion. La condition de non-chevauchement des parties internes des liens $\lhd$ impose que $D^\wedge \in 1$ ssi $C^\vee \in 1$ ssi $D^\wedge \ldots C^\vee \subset 1$, et dans ce cas la partie interne $D^\wedge \ldots C^\vee$ ne contient pas de conclusion. Sinon $D^\wedge$ et $C^\vee$ sont dans $2, 3, 4$: que ce soit avant ou après réduction, les segments $2, 3$ et 4 sont dans cet ordre cyclique dans le voyage long, donc $D^\wedge \ldots C^\vee$ contient une conclusion dans v' ssi elle en contient une dans v, ce qui n'est pas.

- $t = \otimes$. Avec les mêmes notations que ci-dessus pour les 4 segments, le voyage v dans β est $1, 2, 3, 4$ ou $2, 1, 3, 4$ ou $1, 2, 4, 3$ ou $2, 1, 4, 3$, et le voyage v' dans β' est toujours $1, 4, 2, 3$. Dans tous les cas on parle de voyage “à permutation circulaire près”, c'est-à-dire de l'ordre cyclique total (large). Supposons que $D^\wedge \ldots C^\vee$ contienne une conclusion ϕ dans β'. La position de ces trois formules dans $1, 4, 2, 3$ est un triplet $(D^\wedge, \phi, C^\vee) \in \{1, 2, 3, 4\}^3$. Or quel que soit le triplet $(x, y, z) \in \{1, 2, 3, 4\}^3$ tel que $(x, y, z) \in$ l'ordre cyclique $1, 4, 2, 3$, $(x, y, z) \in$ l'un au moins des ordres cycliques $1, 2, 3, 4$ ou $2, 1, 3, 4$ ou $1, 2, 4, 3$ ou $2, 1, 4, 3$: si deux sont égaux parmi x, y, z, c'est évident; par ailleurs $(1, 4, 2) \in (2, 1, 3, 4)$, $(1, 4, 3) \in (1, 2, 4, 3)$, $(1, 2, 3) \in (1, 2, 3, 4)$ et $(4, 2, 3) \in (2, 1, 3, 4)$. Cela fournit une contradiction.

3. Non-chevauchement des parties internes des liens $\lhd$. Soit s une position des interrupteurs sans la position 3. Soient $\frac{C_1 \quad C_2}{C_1 \lhd C_2}$ et $\frac{D_1 \quad D_2}{D_1 \lhd D_2}$ deux autres liens.

- $t = \odot$. On reprend les notations ci-dessus. Dans β', $D_2^\wedge \in 1$ ssi $D_1^\vee \in 1$ ssi $D_2^\wedge \dots D_1^\vee \subset 1$, donc c'est aussi le cas dans β, et alors quel que soit la position de $C_2^\wedge$ et $C_1^\vee$, un chevauchement dans β' provient d'un chevauchement dans au moins l'un des voyages $1, 2, 3, 4$ et $2, 1, 3, 4$ dans β.

  Le cas où $C_2^\wedge \dots C_1^\vee \subset 1$ est similaire.

  Sinon $D_2^\wedge$, $D_1^\vee$, $C_2^\wedge$ et $C_1^\vee$ sont toutes les 4 dans $2, 3, 4$, et le chevauchement est préservé, comme l'ordre cyclique $(2, 3, 4)$.

- $t = \otimes$. Soit un chevauchement $C_2^\wedge, D_2^\wedge, C_1^\vee, D_1^\vee$ (l'autre cas est absolument similaire) dans β', montrons qu'il provient d'un chevauchement dans β. Si les 4 formules sont réparties dans 1, 2 ou 3 segments (parmi $1, 2, 3$ et 4), alors c'est clair. Par ailleurs, elles ne peuvent pas être réparties dans les 4 segments. En effet, si par exemple $C_2^\wedge, D_2^\wedge, C_1^\vee, D_1^\vee$ sont respectivement dans $1, 4, 2, 3$, alors le voyage de la forme $2, 1, 4, 3$ dans β interdit d'une part toute conclusion dans 3 et 4 (pas de conclusion dans la partie interne $C_2^\wedge \dots C_1^\vee$ dans β), et d'autre part toute conclusion dans 1 et 2 (pas de conclusion dans $D_2^\wedge \dots D_1^\vee$), ce qui est impossible. Les autres permutations circulaires de $C_2^\wedge, D_2^\wedge, C_1^\vee, D_1^\vee$ dans $1, 4, 2, 3$ admettent une preuve identique.

4. Condition (iv). Soient s une position des interrupteurs, et $\frac{C_1 \quad C_2}{C_1 \lhd C_2}$ et $\frac{D_1 \quad D_2}{D_1 \ p \ D_2}$ ($p \in \{\lhd, ⅋\}$) deux liens quelconques de β'. Supposons que l'unique cycle déterminé par s dans β' contienne C_2, D_1, C_2, D_2, C_1 dans cet ordre cyclique (le cas C_2, D_2, C_2, D_1, C_1 est analogue). Nécessairement $\frac{C_1 \quad C_2}{C_1 \lhd C_2}$ est en position D et $\frac{D_1 \quad D_2}{D_1 \ p \ D_2}$ n'est pas en position 3. Le cycle contient donc la configuration suivante: $C_2^\vee, D_1, C_2^\wedge, D_2, C_1^\vee, C_1^\wedge$. On reprend les notations $1, 2, 3, 4$ pour les segments du cycle.

Par le même argument que précédemment, on peut "remonter" une configuration C_2, D_1, C_2, D_2, C_1 dans β à partir de la même configuration dans β', dès que les 5 formules sont réparties dans au plus 3 segments (parmi $1, 2, 3, 4$).

Soit donc la configuration C_2, D_1, C_2, D_2, C_1 dans β' répartie dans les 4 segments. Dans un réseau quelconque correct au sens commutatif, les parties inférieures ne se chevauchent pas, donc les deux extrémités $C_2^\vee$ et $C_2^\wedge$ de la partie inférieure de $\frac{C_1 \quad C_2}{C_1 \lhd C_2}$ doivent être soit dans $1, 2$ soit dans $3, 4$ toutes les deux. Pour que les 5 formules soient réparties dans les 4 segments, il faut que $C_2^\vee$ et $C_2^\wedge$ ne soient pas dans un même segment. Donc on est dans l'un des 4 cas suivants:

4.1. $C_2^\vee \in 1$, $C_2^\wedge \in 2$ et $D_1 \in 4$. Déjà $C_1 \notin 2$ (car sinon $D_2 \in 2$ et personne n'est dans 3), et $C_1 \notin 3$ (sinon les trois passages par le lien disjonction $\frac{C_1 \quad C_2}{C_1 \lhd C_2}$ sont répartis dans les trois parties: interne 1, externe 2 et inférieure $3, 4$, d'un autre lien disjonction: $\frac{B^\perp \quad A^\perp}{B^\perp t^\perp A^\perp}$, ce qui n'arrive jamais dans un réseau correct au sens commutatif), donc $C_1 \in 1$, et par conséquent $D_2 \in 3$. Le cycle dans β' est donc:

$A^{\perp\wedge} \overset{1.1}{\therefore} C_1 \overset{1.2}{\therefore} C_2^{\vee} \overset{1.3}{\therefore} B^{\perp\vee} B^{\wedge} \overset{4.1}{\therefore} D_1 \overset{4.2}{\therefore} B^{\vee} B^{\perp\wedge} \overset{2.1}{\therefore} C_2^{\wedge} \overset{2.2}{\therefore} A^{\perp\vee} A^{\wedge} \overset{3.1}{\therefore} D_2 \overset{3.2}{\therefore} A^{\vee}$

mais alors le voyage $2, 1, 3, 4$ dans β, à savoir:

$B^{\perp\wedge} \overset{2.1}{\therefore} C_2^{\wedge} \overset{2.2}{\therefore} A^{\perp\vee} A^{\perp\wedge} \overset{1.1}{\therefore} C_1 \overset{1.2}{\therefore} C_2^{\vee} \overset{1.3}{\therefore} B^{\perp\vee} (B^{\perp} \lhd A^{\perp})^{\vee} (A \odot B)^{\wedge} A^{\wedge} \overset{3.1}{\therefore} D_2 \overset{3.2}{\therefore} A^{\vee} B^{\wedge} \overset{4.1}{\therefore} D_1 \overset{4.2}{\therefore} B^{\vee} (A \odot B)^{\vee} (B^{\perp} \lhd A^{\perp})^{\wedge}$

fait apparaître la configuration $C_2^{\wedge}, A^{\perp}, C_1, C_2^{\vee}, B^{\perp}$ qui contredit la condition (iv) pour β: absurde.

4.2. $C_2^{\vee} \in 2$, $C_2^{\wedge} \in 1$ et $D_1 \in 3$: analogue.

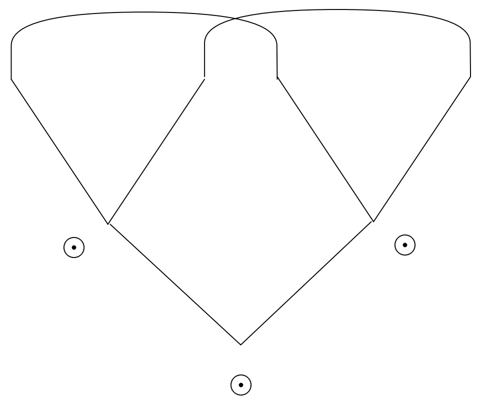
4.3. $C_2^{\vee} \in 3$, $C_2^{\wedge} \in 4$ et $D_1 \in 1$. On procède comme ci-dessus. La seule différence essentielle est la raison pour laquelle $C_1 \notin 2$: sinon on aurait dans β un enchevêtrement entre une disjonction et une conjonction, i.e. les trois passages par le lien disjonction $\frac{C_1 \quad C_2}{C_1 \lhd C_2}$ répartis dans les trois parties $(1, 2;\ 3;\ 4)$ d'un lien conjonction $(\frac{A \quad B}{AtB})$, ce qui n'arrive jamais dans un réseau correct au sens commutatif.

4.4. $C_2^{\vee} \in 4$, $C_2^{\wedge} \in 3$ et $D_1 \in 2$: analogue. ■

5.5.2 Séquentialisation

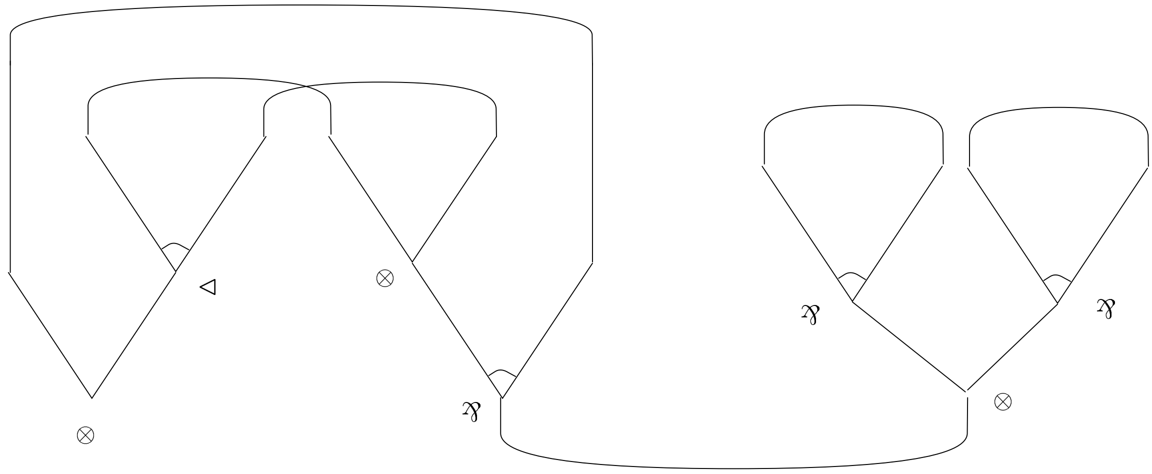
Définition 5.5.8 *A une preuve π (de conclusion $\vdash A_1, \ldots, A_n$ $[R]$) dans le calcul des séquents multiplicatif, on associe une structure de preuve π^{-} (ayant pour conclusions $A_1, \ldots, A_n$) de manière évidente.*

Déjà quelques remarques: le lemme 5.5.6 dit en particulier que la condition (iii) est nécessaire. Ainsi par exemple, la structure suivante satisfait (i) et (ii) pour toute position d'interrupteurs, mais n'est évidemment pas correcte:



La condition (iii) est en fait équivalente, modulo les autres conditions, à la contrainte d'ordre de passage dans les liens tenseurs du lemme 2.9.7 de [17].

Par ailleurs, la condition (iv) est utile en présence de coupures: ainsi la structure suivante satisfait (i), (ii) et (iii) pour toute position d'interrupteurs, mais n'est pas correcte parce que la partie interne du lien $\lhd$ passe par un lien $⅋$ qui est "en-dessous":



Nous avons besoin de la notation suivante:

Définition 5.5.9 *Soit β un réseau de preuve de conclusions $A_1, \ldots, A_n$, et s une position d'interrupteurs pour β. L'unique cycle sur β déterminé par s contient par définition $A_1, \ldots, A_n$, et induit donc un ordre cyclique total, noté $R_{\beta,s}$, sur $A_1, \ldots, A_n$.*

Comme un ordre cyclique total est série-parallèle, l'intersection d'ordres cycliques totaux (sur un même ensemble E) est un ordre cyclique série-parallèle (sur E), en vertu de la proposition 5.4.17.

Déjà, on a la proposition suivante:

Proposition 5.5.10 *Si π est une preuve séquentielle de conclusion $\vdash \Gamma\ [R]$, π^- est un réseau de preuve, et $R \subseteq \bigcap_s R_{\pi^-,s}$.*

Preuve. Induction facile. ∎

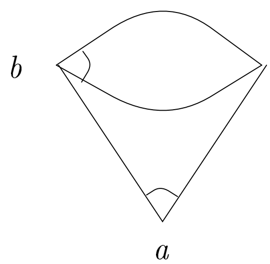
On définit un ordre entre les liens disjonctions ($\lhd$ et $⅋$) d'une structure de preuve correcte au sens commutatif (cette définition n'a rien de spécifiquement non-commutatif). La notation suivante est très utile:

Définition 5.5.11 *Soient β une structure de preuve correcte au sens commutatif, et $a = \frac{A_1 \quad A_2}{A_1 p A_2}$ et $b = \frac{B_1 \quad B_2}{B_1 q B_2}$ ($p, q \in \{\lhd, ⅋\}$) deux liens disjonctions quelconques de β.*

Soit $s = s_{a,b}$ une position d'interrupteurs à la Danos-Régnier (on coupe l'une des deux branches des liens disjonctions) pour tous les liens disjonctions de β sauf *a et b. Le graphe obtenu contient exactement deux cycles indépendants, et des arètes pendantes. On note* $\beta^s_{a,b}$ *le graphe obtenu en enlevant en outre les arètes pendantes.*

Définition 5.5.12 *Soient* β *une structure de preuve correcte au sens commutatif, et* $a = \frac{A_1 \quad A_2}{A_1 p A_2}$ *et* $b = \frac{B_1 \quad B_2}{B_1 q B_2}$ $(p, q \in \{\triangleleft, \parr\})$ *deux liens disjonctions quelconques de* β.

On définit la relation $<_\beta$ *par* $a <_\beta b$ *ssi pour une certaine position d'interrupteurs (sans la position 3), à la fois la partie supérieure et la partie inférieure de b passent par a, autrement dit ssi pour une certaine position d'interrupteurs* $s = s_{a,b}$ *à la Danos-Régnier pour toutes les disjonctions sauf a et b,* $\beta^s_{a,b}$ *est le graphe:*

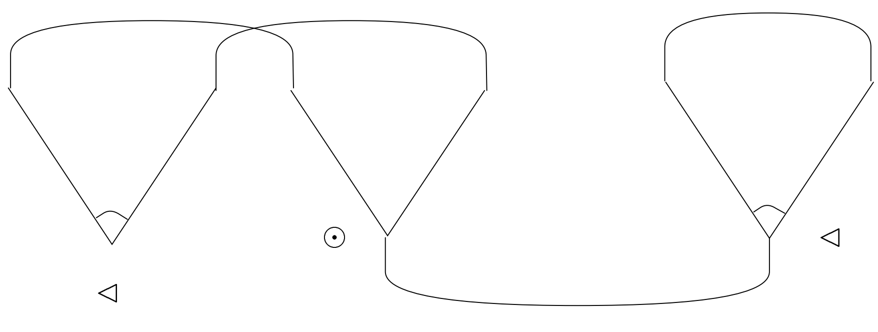


Les deux lemmes (faciles) suivants expliquent notre intérêt pour $<_\beta$:

Lemme 5.5.13 (Ordre sur les disjonctions) *Soit* β *est une structure de preuve correcte au sens commutatif.* $<_\beta$ *est un ordre sur les liens disjonctions.*

Lemme 5.5.14 (Scission) *Soit* β *est un réseau de preuve, et* $a = \frac{A_1 \quad A_2}{A_1 \ p \ A_2}$ $(p \in \{\triangleleft, \parr\})$ *un lien de* β. *a est minimal pour* $<_\beta$ *ssi il est scindant, i.e. le graphe obtenu en enlevant les deux arètes de a a deux composantes connexes.*

Venons-en au théorème. Un lien scindant dans le cas commutatif ne l'est pas forcément dans le cas mixte (ou même purement non-commutatif), par exemple la coupure dans le réseau (correct) suivant:



mais on a une relation d'ordre sur les disjonctions, donc il est naturel de prouver la séquentialisation avec les disjonctions ($\lhd$ et $⅋$) scindantes, à la manière de Danos [9]. On va donc considérer des structures de preuve avec axiomes non-logiques à une conclusion, et on imagine sans peine les adaptations évidentes nécessaires.

Théorème 5.5.15 (Séquentialisation) *Soit β un réseau de preuve de conclusions $A_1, \ldots, A_n$, et soit $R = \bigcap_s R_{\beta,s}$. Il existe une preuve en calcul des séquents π, de conclusion $\vdash A_1, \ldots, A_n$ $[R]$, telle que $\beta = \pi^-$.*

Preuve. Soit β un réseau de preuve de conclusions $A_1, \ldots, A_n$. On procède par induction sur le nombre n de liens disjonctions. Si $n = 0$, β est un arbre: clair.

Si $n > 0$, on considère tous les liens disjonctions scindants $a_1, \ldots, a_k$ $(k \leq n)$ de β. Comme β est en particulier correcte au sens commutatif, alors d'après [9], $k \geq 1$. L'un au moins de ces liens, disons a_i, n'est sur le trajet d'aucune partie interne de lien $\lhd$.

En effet, sinon les composantes connexes du graphe obtenu en enlevant les deux arètes des $a_1, \ldots, a_k$, forment un arbre dans β (car sinon on exhibe un cycle). Comme les conclusions ne sont pas dans les parties internes des liens $\lhd$, il existe une composante feuille γ qui contient un lien $\lhd$ non-scindant b et est connectée au reste de β par les deux prémisses d'un unique lien scindant a_j. D'après le lemme 5.5.14, b n'est pas minimal pour l'ordre $< \beta$, donc fatalement $a_j < \beta b$, et on vérifie aisément que cela contredit la condition (iv) de correction.

Soit donc a_i, sur le trajet d'aucune partie interne de lien $\lhd$. Considérons les deux composantes connexes β_1 et β_2 du graphe obtenu en enlevant les deux arètes de a_i. Ce sont deux structures de preuve (l'une avec un nouvel axiome non-logique à une conclusion $A_1 p A_2$). D'après [9], β_1 et β_2 sont correctes au sens commutatif. On a choisi a_i entre autres pour que la partie interne d'un lien $\lhd$ quelconque de β_1 ou β_2 ne passe pas par a_i, donc les parties internes des liens $\lhd$ de β_1 et β_2 ne contiennent pas de conclusion. Le non-chevauchement et la condition (iv) se projètent bêtement sur β_1 et β_2. Par conséquent β_1 et β_2 sont des réseaux de preuve, on leur associe grâce à l'hypothèse d'induction deux preuves séquentielles π_1 et π_2 respectives (la vérification des ordres cycliques impose une application de la règle d'entropie dans le cas où $p = ⅋$), qu'on associe pour former une preuve séquentielle π telle que $\beta = \pi^-$. ■

5.6 Version intuitionniste

Nous présentons un calcul des séquents pour la version intuitionniste de la logique mixte, nous en aurons besoin dans le Chapitre suivant. On l'obtient comme d'habitude en associant aux formules des polarités intuitionnistes, et en se limitant à des séquents ayant au plus une formule positive (les formules négatives étant passées à gauche du signe $\vdash$). Le fragment multiplicatif intuitionniste est exactement celui présenté par de Groote [11]. Nous ne donnons pas le détail de la sémantique des phases intuitionniste qui ne présente pas de difficulté particulière par rapport à [11].

Les formules intuitionnistes sont définies par:

Définition 5.6.1 (Formules intuitionnistes) *Les* formules intuitionnistes *sont construites à partir d'atomes* $p, q, \ldots$ *avec:*

– *les connecteurs non-commutatifs: conjonction* $\odot$ (puis) *et implications gauche* $\multimap\!\bullet$ *et droite* $\bullet\!\!-$ *,*

– *les connecteurs multiplicatifs commutatifs:* $\otimes$ (tenseur) *et implication* $\multimap$*,*

– *les connecteurs additifs:* $\&$ (avec) *et* $\oplus$ (plus)*,*

– *le connecteur exponentiel* ! (bien sûr)*,*

– *les constantes: multiplicatives* $\mathbf{1}$ *et* $\bot$*, et additives* $\top$ *et* $\mathbf{0}$*,*

– *les quantificateurs: universel* $\forall$ *et existentiel* $\exists$*.*

Définition 5.6.2 (Séquents intuitionnistes) *Les* séquents *sont de la forme* $\Gamma \vdash A$ *ou* $\Gamma \vdash$*, où* A *est une formule et* $\Gamma \in \mathcal{H}$*.* $\mathcal{H}$ *et* $\mathcal{H}_0$ *sont respectivement les ensembles de* contextes *et de* contextes non vides *et sont définis par la grammaire suivante:*

- $\mathcal{H} ::= () \mid \mathcal{H}_0$
- $\mathcal{H}_0 ::= \mathcal{F} \mid (\mathcal{H}_0, \mathcal{H}_0) \mid (\mathcal{H}_0; \mathcal{H}_0)$

Le calcul des séquents que nous présentons a des règles structurelles explicites, ce qui nous semble plus clair, mais remarquons que dans le cas intuitionniste, le fait de fixer une formule à droite interdit bien sûr l'échange circulaire, et le quotient de l'ensemble des séquents par les règles structurelles réversibles est simplement, dans ce cas, l'ensemble des ordres série-parallèles étiquetés par des formules.

On reprend les notations de la Section 5.1.3; de plus | désignera l'un ou l'autre des séparateurs "," ou ";". Les règles du calcul des séquents intuitionniste sont:

Axiome - Coupure

$$A \vdash A \qquad \frac{\Gamma \vdash A \qquad \Delta[A] \vdash B}{\Delta[\Gamma] \vdash B}$$

Règles structurelles

$$\frac{\Gamma[(\Delta,\Sigma),\Pi] \vdash A}{\Gamma[\Delta,(\Sigma,\Pi)] \vdash A} \qquad \frac{\Gamma[\Delta,(\Sigma,\Pi)] \vdash A}{\Gamma[(\Delta,\Sigma),\Pi] \vdash A} \qquad \frac{\Gamma[(\Delta;\Sigma);\Pi] \vdash A}{\Gamma[\Delta;(\Sigma;\Pi)] \vdash A} \qquad \frac{\Gamma[\Delta;(\Sigma;\Pi)] \vdash A}{\Gamma[(\Delta;\Sigma);\Pi] \vdash A}$$

$$\frac{\Gamma[\Delta,\Sigma] \vdash A}{\Gamma[\Sigma,\Delta] \vdash A}\text{échange} \qquad \frac{\Gamma[\Delta;\Sigma] \vdash A}{\Gamma[\Delta,\Sigma] \vdash A}\text{entropie}$$

Non-commutatifs

$$\frac{\Gamma[A;B] \vdash C}{\Gamma[A \odot B] \vdash C} \qquad \frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Delta;\Gamma \vdash A \odot B}$$

$$\frac{\Gamma \vdash A \qquad \Delta[B] \vdash C}{\Delta[\Gamma; A \multimap\!\!\bullet\, B] \vdash C} \qquad \frac{A;\Gamma \vdash B}{\Gamma \vdash A \multimap\!\!\bullet\, B}$$

$$\frac{\Gamma \vdash A \qquad \Delta[B] \vdash C}{\Delta[B \bullet\!\!\!- A;\Gamma] \vdash C} \qquad \frac{\Gamma; A \vdash B}{\Gamma \vdash B \bullet\!\!\!- A}$$

Multiplicatifs commutatifs

$$\frac{\Gamma[A,B] \vdash C}{\Gamma[A \otimes B] \vdash C} \qquad \frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Delta,\Gamma \vdash A \otimes B}$$

$$\frac{\Gamma \vdash A \qquad \Delta[B] \vdash C}{\Delta[\Gamma, A \multimap B] \vdash C} \qquad \frac{A,\Gamma \vdash B}{\Gamma \vdash A \multimap B}$$

Additifs

$$\frac{\Gamma[A] \vdash C \qquad \Gamma[B] \vdash C}{\Gamma[A \oplus B] \vdash C} \qquad \frac{\Gamma \vdash A}{\Gamma \vdash A \oplus B} \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \oplus B}$$

$$\frac{\Gamma[A] \vdash C}{\Gamma[A \& B] \vdash C} \qquad \frac{\Gamma[B] \vdash C}{\Gamma[A \& B] \vdash C} \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \& B}$$

Constantes

$$\frac{\Gamma[\Delta] \vdash A}{\Gamma[\Delta \mid \mathbf{1}] \vdash A} \qquad \frac{\Gamma[\Delta] \vdash A}{\Gamma[\mathbf{1} \mid \Delta] \vdash A} \qquad \vdash \mathbf{1} \qquad \bot \vdash \qquad \frac{\Gamma \vdash}{\Gamma \vdash \bot}$$

$$\Gamma \vdash \top \qquad \Gamma[\mathbf{0}] \vdash A$$

Exponentiel

$$\frac{\Gamma[A] \vdash B}{\Gamma[!A] \vdash B} \qquad \frac{!\Gamma \vdash A}{!\Gamma \vdash !A} \qquad \frac{\Gamma[!\Delta, !\Sigma] \vdash C}{\Gamma[!\Delta; !\Sigma] \vdash C}$$

$$\frac{\Gamma[!A \mid !A] \vdash B}{\Gamma[!A] \vdash B} \qquad \frac{\Gamma[\Delta] \vdash B}{\Gamma[\Delta \mid !A] \vdash B} \qquad \frac{\Gamma[\Delta] \vdash B}{\Gamma[!A \mid \Delta] \vdash B}$$

Quantificateurs

$$\frac{\Gamma[A[t/x]] \vdash B}{\Gamma[\forall x A] \vdash B} \qquad \frac{\Gamma \vdash A}{\Gamma \vdash \forall x A}\, x \notin vl(\Gamma)$$

$$\frac{\Gamma[A] \vdash B}{\Gamma[\exists x A] \vdash B}\, x \notin vl(\Gamma, B) \qquad \frac{\Gamma \vdash A[t/x]}{\Gamma \vdash \exists x A}$$

Le calcul des séquents ci-dessus satisfait la propriété d'élimination des coupures et les preuves sans coupure ont la propriété de la sous-formule.

Chapter 6

Correspondance cc – logique linéaire mixte

Dans ce chapitre nous montrons que la logique linéaire mixte permet de rendre compte finement de la concurrence dans les langages `cc`: ainsi les suspensions d'un agent `cc` peuvent être caractérisées dans la logique [48].

6.1 Contraintes de synchronisation

Nous avons vu dans le Chapitre 4 que les suspensions ne pouvaient pas être caractérisées en logique linéaire commutative, une raison importante étant la déduction

$$A \otimes (c \multimap B) \vdash c \multimap (A \otimes B)$$

qui introduit des formules ressemblant à des traductions de suspensions. En logique mixte, nous avons bien sûr toujours cette propriété pour $\otimes$ et $\multimap$, mais pas pour $\otimes$ et $\multimap\!\!\bullet$ en général.

Il y a évidemment d'autres obstacles à l'observation des suspensions, à savoir:

$$d \otimes (c \multimap\!\!\bullet d) \otimes (d \multimap\!\!\bullet A) \vdash d \otimes (c \multimap\!\!\bullet A)$$

$$\frac{c \vdash A}{1 \vdash c \multimap\!\!\bullet A}$$

et

$$\frac{c \vdash c'}{c \otimes (d \multimap\!\!\bullet A) \vdash c' \otimes (d \multimap\!\!\bullet A)} \qquad \frac{d' \vdash d}{c \otimes (d \multimap\!\!\bullet A) \vdash c \otimes (d' \multimap\!\!\bullet A)}$$

Occupons-nous d'abord de la première implication: elle consistitue un obstacle à la caractérisation des suspensions si on choisit de traduire l'agent $c \to A$ par $c \multimap\!\!\bullet A$: alors en effet, si $d \nvdash c$, $d \otimes (c \multimap\!\!\bullet A)$ a l'apparence d'une suspension, alors que l'agent traduit par $d \otimes (c \multimap d) \otimes (d \multimap A)$ peut très bien avoir un succès (si A débloque c) et ne pas suspendre.

L'idée la plus simple est donc de traduire $c \to A$ par $c \multimap (\diamond \otimes A)$, où $\diamond$ est une nouvelle formule atomique (ni une contrainte, ni un nom de procédure). Intuitivement $\diamond$ assure que la communication entre agents s'effectue à travers le store, et évite ce type de "composition des suspensions".

Les obstacles suivants sont plus importants et portent sur la nature des contraintes considérées. Ce que nous proposons, c'est de nous limiter à caractériser les suspensions pour lesquelles les contraintes bloquantes sont des informations élémentaires: parmi toutes les contraintes, on en distingue donc certaines, atomiques, et n'apparaissant pas dans $\Vdash_{\mathcal{C}}$. Remarquons que les exemples standards de programmes `lcc` utilisés pour la spécification de protocoles (comme le dîner des philosophes, voir Chapitre 2) satisfont ce type de restriction.

Définition 6.1.1 *Etant donné un système de contraintes linéaires* $(\mathcal{C}, \Vdash_{\mathcal{C}})$*, un ensemble* $\mathcal{C}_s$ *de* contraintes de synchronisation *est un ensemble de formules atomiques de* $\mathcal{C}$ *n'apparaissant pas dans* $\Vdash_{\mathcal{C}}$*.*

Les contraintes linéaires *sont les contraintes* c *telles qu'aucune sous-formule de* c *n'est une contrainte de synchronisation.*

Les contraintes définies *sont celles obtenues en combinant des contraintes linéaires et des contraintes de synchronisation avec des* $\otimes$*.*

Un agent tell(c) sera dit défini *si la contrainte* c *est* définie*.*

Une configuration κ *sera dite* définie *si le store de* κ *et les agents de* κ *sont tous* définis.

Nous les appelons des contraintes de synchronisation pour indiquer que c'est ce type de contraintes qui est considéré dans les programmes `cc` de spécification de protocoles. Ce sont les contraintes qui permettent de traduire en `cc` les calculs de processus: par exemple la version asynchrone du π-calcul de Milner [38, 7] peut être traduite dans un langage `lcc` [50] avec comme contraintes les formules atomiques $x : y$ où x et y sont des variables et $:$ un symbole de prédicat ($x : y$ signifie: "le nom de canal x transporte le nom de canal y), et ces contraintes sont bien des contraintes de synchronisation. Le dîner des philosophes (Chapitre 2) fournit un autre exemple.

On rappelle que la relation $>$ entre contraintes linéaires est définie (Chapitre 2, Définition 2.2.6) comme la plus petite relation contenant $\vdash_{\mathcal{C}}$ et close par: $\forall e \in \mathcal{C}(c > d \;\Rightarrow\; (c \otimes e) > d)$.

Notation:
▶ Soient $\kappa = (\vec{x}; c; \Gamma)$ une configuration `lcc`, et ϕ:

- un nom de procédure,

- ou un agent $c \otimes (d_1 \multimap A_1) \otimes \cdots \otimes (d_n \multimap A_n)$, $n \geq 0$, telle que c est une contrainte, et pour tout $i(0 \leq i \leq n)$ d_i est une contrainte de synchronisation et pour tout $i(0 \leq i \leq n)$ $c \not\succ d_i$.

Dans le second cas on dira que la formule ϕ est une "suspension", même si $n = 0$, on considère donc un succès comme un cas particulier de suspension.

On écrit $\kappa \xrightarrow{\triangleright} \phi$ pour signifier:

- si ϕ est un nom de procédure: "il existe une configuration $(\vec{y}; d; \phi)$, telle que $vl(\phi) \cap \vec{y} = \emptyset$, $d \vdash_{\mathcal{C}} 1$ et $\kappa \longrightarrow (\vec{y}; d; \phi)$",
- si ϕ est une suspension: "il existe une configuration
$$\lambda = (\vec{y}; d; d_1 \rightarrow B_1, \ldots, d_n \rightarrow B_n),$$
telle que $\kappa \longrightarrow \lambda$, $d \vdash_{\mathcal{C}} c$ et pour tout $i(0 \leq i \leq n)$: $d \not\succ d_i$".

6.2 Traduction des agents en formules

On traduit les agents et configurations `lcc` en formules de logique linéaire mixte intuitionniste, plus précisément dans le fragment de NLLI comprenant la constante 1, les connecteurs &, $\otimes$ et $\multimap$, et le quantificateur $\exists$.

Fixons un système de contraintes linéaires $(\mathcal{C}, \Vdash_{\mathcal{C}})$ et un ensemble de déclarations $\mathcal{D}$. Fixons aussi un ensemble $\mathcal{C}_s$ de contraintes de synchronisation. On fait désormais l'hypothèse que les contraintes de $\Vdash_{\mathcal{C}}$ et les agents dans les déclarations de $\mathcal{D}$ sont tous définis. (Cette hypothèse ne sert en fait que pour l'observation des suspensions.)

Le fait d'être une configuration définie est alors conservé lors de l'exécution:

Lemme 6.2.1 *Une sous-formule d'une contrainte définie est une contrainte définie. Un sous-agent d'un agent défini est un agent défini. Si κ est une configuration `lcc` définie et $\kappa \xrightarrow{\triangleright} \lambda$, alors λ est définie.*

Comme il est suggéré dans le paragraphe précédent, on introduit une nouvelle formule atomique $\diamond$, considérée ni comme une contrainte, ni comme un nom de procédure. On note $\vec{\diamond}$ le produit de n occurrences de $\diamond$ (n quelconque ≥ 0): $\diamond \otimes \cdots \otimes \diamond$.

Définition 6.2.2 *On traduit les agents `lcc` (non nécessairement déterministes) en formules de logique mixte (intuitionniste) de la manière suivante:*

$$c^* = c, \textit{ si } c \textit{ est une contrainte}$$
$$\begin{array}{ll} tell(c)^* = c & p(\vec{x})^* = p(\vec{x}) \\ (c \rightarrow A)^* = c \multimap (\diamond \otimes A^*) & (A \parallel B)^* = A^* \otimes B^* \\ (A + B)^* = A^* \& B^* & (\exists x A)^* = \exists x A^* \end{array}$$

Si Γ *est le multi-ensemble d'agents* $(A_1 \dots A_n)$*, on définit* $\Gamma^* = A_1^* \otimes \cdots \otimes A_n^*$*.*
La traduction $(\vec{x}; c; \Gamma)^*$ *d'une configuration* $(\vec{x}; c; \Gamma)$ *est la formule* $\exists \vec{x}(c \otimes \Gamma^*)$*.*

On note NLLI($\mathcal{C}$,$\mathcal{D}$) le système de déduction obtenu en ajoutant à NLLI:

- l'axiome non-logique $c \vdash d$ pour tout $c \Vdash_{\mathcal{C}} d$ dans $\Vdash_{\mathcal{C}}$,
- l'axiome non-logique $p(\vec{x}) \vdash A^*$ pour toute déclaration $p(\vec{x}) = A$ dans $\mathcal{D}$.

Théorème 6.2.3 (Correction) *Soient* $(\vec{x}; c; \Gamma)$ *et* $(\vec{y}; d; \Delta)$ *des configurations* `lcc`*.*
Si $(\vec{x}; c; \Gamma) \equiv (\vec{y}; d; \Delta)$ *alors* $(\vec{x}; c; \Gamma)^* \dashv\vdash_{NLLI(\mathcal{C},\mathcal{D})} (\vec{y}; d; \Delta)^*$*.*
Si $(\vec{x}; c; \Gamma) \longrightarrow (\vec{y}; d; \Delta)$ *alors* $(\vec{x}; c; \Gamma)^* \vdash_{NLLI(\mathcal{C},\mathcal{D})} (\vec{y}; d; \Delta)^* \otimes \vec{\diamond}$*.*

Preuve. Par induction sur $\equiv$ et $\longrightarrow$.
– Pour la *composition parallèle*, la α*-conversion*, c'est immédiat.
– Pour l'*inaction*, on a $A^\dagger \otimes 1 \dashv\vdash A^\dagger$.
– Pour les *variables locales*, on a $\exists x(A \otimes B) \dashv\vdash A \otimes \exists x B$ et $\exists x A \dashv\vdash A$ dès que $x \notin vl(A)$.
– Pour *Tell*, $\equiv$ et les *déclarations*, c'est immédiat.
– Pour *Ask*, on a bien $c \otimes (d \multimap (\diamond \otimes A)) \vdash e \otimes \diamond \otimes A$ si $c \Vdash_{\mathcal{C}} d \otimes e$:

$$\dfrac{c \vdash d \otimes e \qquad \dfrac{\dfrac{\dfrac{\dfrac{d \vdash d \qquad \dfrac{e \vdash e \qquad \diamond \otimes A \vdash \diamond \otimes A}{e, \diamond \otimes A \vdash e \otimes \diamond \otimes A}}{e, (d; d \multimap (\diamond \otimes A)) \vdash e \otimes \diamond \otimes A}}{e, d, d \multimap (\diamond \otimes A) \vdash e \otimes \diamond \otimes A}}{d, e, d \multimap (\diamond \otimes A) \vdash e \otimes \diamond \otimes A}}{d \otimes e, d \multimap (\diamond \otimes A) \vdash e \otimes \diamond \otimes A}}{\dfrac{c, d \multimap (\diamond \otimes A) \vdash e \otimes \diamond \otimes A}{c \otimes (d \multimap (\diamond \otimes A)) \vdash e \otimes \diamond \otimes A}}$$

■

6.3 L'observation des succès et des suspensions dans `lcc`

On a, exactement comme au Chapitre 4, le

Lemme 6.3.1 *Soient* κ *et* λ *deux configurations* `lcc` *telle que* $\kappa^* = \lambda^*$*, et* ϕ *un nom de procédure ou une suspension. On a* $\kappa \xrightarrow{\triangleright} \phi$ *ssi* $\lambda \xrightarrow{\triangleright} \phi$*.*

Lemme 6.3.2 *Si* c *est une contrainte de synchronisation,* A *une traduction d'agent et* Γ *et* Δ *deux contextes dont les formules sont des traductions d'agents, alors* c; $(\diamond, \Delta)$; $\Gamma \not\vdash_{NLLI(\mathcal{C},\mathcal{D})} \diamond \otimes A$*.*

Preuve. C'est à peu près évident par l'absurde: on montre par induction que si on a une preuve de $c; (\diamond, \Delta); \Gamma \vdash_{NLLI(\mathcal{C},\mathcal{D})} \diamond \otimes A$, c n'est pas une contrainte de synchronisation.

La dernière règle appliquée est une règle gauche unaire (1, &, $\otimes$, $\exists$ ou entropie) ou binaire ($\multimap$ ou une coupure), pas une règle droite à cause des ";" à gauche. On vérifie alors aisément que dans tous les cas la (ou des) prémisse a bien la forme considérée.

L'important est que le $\diamond$ à gauche ne disparaît pas quand on remonte dans la preuve, car les contraintes d dans les formules $d \multimap D$ qui sont des traductions d'agents ne contiennent pas de $\diamond$. ■

Le Lemme suivant est immédiat:

Lemme 6.3.3 *Si c et d sont des contraintes de synchronisation et $c \vdash_{\mathcal{C}} d$, alors $c = d$.*

Si c est une contrainte de synchronisation, d une contrainte définie, et $c \vdash_{\mathcal{C}} d$, alors $c \dashv\vdash d$.

On évitera les distinctions bureaucratiques entre deux formules équivalentes, donc si $c \dashv\vdash d$ on considèrera que $c = d$.

Théorème 6.3.4 *Soient $\kappa = (\vec{x}; c; \Gamma)$ une configuration lcc définie, et ϕ un nom de procédure ou une suspension.*

Si $\kappa^* \vdash_{NLLI(\mathcal{C},\mathcal{D})} \vec{\diamond} \otimes \phi^*$, alors $\kappa \xrightarrow{\triangleright} \phi$.

Preuve. Commençons par noter que dans une preuve du calcul des séquents $NLLI(\mathcal{C}, \mathcal{D})$ d'un séquent dont les formules sont formées avec 1, $\otimes$, $\multimap$, $\exists$ et &, on peut supposer que les coupures portent uniquement sur des axiomes non-logiques provenant de $\mathcal{C}$ et $\mathcal{D}$, donc que toute formule de la preuve ne contient que les connecteurs et le quantificateur cités.

Dans une telle preuve, les membres gauches des séquents contiennent des séparateurs de deux types: "," et ";". Supposons qu'on ait introduit un ";", nécessairement par la règle $\multimap \vdash$

$$\frac{\Gamma \vdash c \qquad \Delta[A] \vdash B}{\Delta[\Gamma; c \multimap A] \vdash B}$$

Descendons dans cette preuve en suivant ce ";". A un moment donné il est éliminé par une des deux règles suivantes: l'entropie ou $\vdash \multimap$.

Le point important est que dans le cas où il est éliminé par $\vdash \multimap$, la portion de preuve entre l'introduction et l'élimination est très simple, à cause de l'hypothèse sur la nature de ϕ (suspensions portant sur des contraintes de synchronisation). En effet en descendant dans la preuve, les formules à gauche du ";" considéré sont en particulier composées des formules de Γ. L'hypothèse sur la nature de ϕ

impose donc que Γ est soit $\emptyset$ soit juste une contrainte de synchronisation (formule atomique).

Dans le premier cas on aurait donc $\Delta[\,] = d; [\,]; \Delta'$ pour une certaine contrainte de synchronisation d, mais comme $A = \diamond \otimes A'$ on aurait alors une preuve de $d; \diamond \otimes A'; \Delta' \vdash B$, ce qui est impossible d'après le Lemme 6.3.2 puisque B est de la forme $\diamond \otimes B'$.

Par conséquent $\Delta[\ ] = [\ ]$ et $\Gamma = d$, avec d contrainte de synchronisation. Comme $d \vdash c$ et c est une contrainte définie (par hypothèse), le Lemme 6.3.3 dit que $d \dashv\vdash c$, soit $d = c$ à équivalence logique près. De plus comme $\Delta[\Gamma; c \multimapdot A] = c; c \multimapdot A$, la seule suite possible de la preuve est $\vdash\ \multimapdot$. La forme nécessaire de la preuve dans le cas d'un ";" éliminé par $\vdash\ \multimapdot$ est donc:

$$\dfrac{\dfrac{c \vdash c \qquad A \vdash B}{c; c \multimapdot A \vdash B}}{c \multimapdot A \vdash c \multimapdot B}$$

ce qu'on conviendra d'abréger en

$$\dfrac{A \vdash B}{c \multimapdot A \vdash c \multimapdot B}$$

considérant ainsi plus simplement qu'on n'a pas introduit de séquentialité (";") dans cette portion de preuve.

Donc (avec la convention ci-dessus) tout ";" introduit est éliminé par la règle d'entropie.

Maintenant que les ";" utiles ont étés isolés, on peut oublier ceux qui restent (ils seront éliminés par une entropie): montrons que si $\Gamma \vdash_{NLLI(\mathcal{C},\mathcal{D})} \vec{\diamond}\otimes\phi^*$, où Γ un contexte (quelconque) dont les formules sont des traductions d'agents $A_1, \ldots, A_n$, alors $(\emptyset; 1; A_1, \ldots, A_n) \xrightarrow{\triangleright} \phi$. On procède par induction sur une preuve de $\Gamma \vdash \phi$ dans le calcul des séquents $NLLI(\mathcal{C}, \mathcal{D})$, avec l'associativité de "," et ";" et la commutativité de "," implicites, et avec la convention ci-dessus. Cette induction a un sens pour la même raison que dans le Chapitre 4.

Chaque règle logique simule une règle de transition `lcc`.

– π est un axiome: on utilise la réflexivité de $\longrightarrow$ dans le cas d'un axiome logique, la règle *déclarations* pour un axiome $p \vdash q$; le cas d'un axiome $d \vdash_{\mathcal{C}} e$ est trivial.

– π se termine par une coupure. Considérons par exemple:

$$\dfrac{p \vdash \psi^* \qquad \Gamma^*[\psi^*] \vdash \phi^*}{\Gamma^*[p] \vdash \phi^*}$$

Par hypothèse d'induction, $(\emptyset; 1; \Gamma, \psi^*) \xrightarrow{\triangleright} \phi$, et $(p = \psi) \in \mathcal{P}$, donc en appliquant la règle *déclaration*, on obtient $(\emptyset; 1; \Gamma, p) \xrightarrow{\triangleright} \phi$, cqfd.

Les autres cas sont similaires.

– π se termine par une introduction à gauche de 1: immédiat par la règle *inaction*.

– π se termine par une introduction à gauche ou à droite de $\otimes$: la preuve est la même que pour le Théorème 4.1.4.

– π se termine par:

$$\frac{\Gamma^*[A^*] \vdash \phi^*}{\Gamma^*[A^* \& B^*] \vdash \phi^*}$$

Par hypothèse d'induction, $(\emptyset; 1; A, \Gamma) \stackrel{\triangleright}{\longrightarrow} \phi$, Or $(\emptyset; 1; A+B, \Gamma) \longrightarrow (\emptyset; 1; A, \Gamma)$, donc $(\emptyset; 1; A + B, \Gamma) \stackrel{\triangleright}{\longrightarrow} \phi$.

– π se termine par une introduction à droite (cas où ϕ est une contrainte, $n = 0$) ou à gauche de $\exists$: la preuve est la même que pour le Théorème 4.1.4.

– π se termine par:

$$\frac{\Delta^* \vdash c \qquad \Gamma^*[A^*] \vdash \diamond \otimes \phi^*}{\Gamma^*[\Delta^*; c \multimap A^*] \vdash \diamond \otimes \phi^*}$$

Par hypothèse d'induction, $(\emptyset; 1; \Delta) \stackrel{\triangleright}{\longrightarrow} c$, c'est-à-dire qu'il existe une configuration $(\vec{y}; d; 1)$, telle que $d \vdash_{\mathcal{C}} c$ et $(\emptyset; 1; \Delta) \longrightarrow (\vec{y}; d; 1)$. Donc en appliquant la règle *ask*, on obtient $(\emptyset; 1; c \to A, \Delta) \longrightarrow (\vec{y}; d; c \to A) \longrightarrow (\vec{y}; 1; A)$. Par conséquent, $(\emptyset; 1; c \to A, \Delta, \Gamma) \longrightarrow (\vec{y}; 1; A, \Gamma)$. De plus par hypothèse d'induction, $(\emptyset; 1; A, \Gamma) \stackrel{\triangleright}{\longrightarrow} \phi$, d'où $(\emptyset; 1; c \to A, \Delta, \Gamma) \stackrel{\triangleright}{\longrightarrow} \phi$.

– π se termine par:

$$\frac{A^* \vdash B^*}{c \multimap A^* \vdash c \multimap B^*}$$

(Avec notre convention, cette règle remplace l'introduction à droite de $\multimap$.) C'est clair: on a bien $(\emptyset; 1; c \to A) \stackrel{\triangleright}{\longrightarrow} c \to B$.

– π se termine par une entropie: immédiat, on a déjà fait le travail pour regrouper les ";" qui sont intéressants (c'est-à-dire ceux qui sont éliminés par la règle $\vdash \multimap$).

– π se termine par une déréliction, un affaiblissement ou une contraction: la preuve est la même que pour le Théorème 4.2.4.

– π se termine par une promotion: dans ce cas toutes les formules sont nécessairement des contraintes, donc c'est immédiat. ∎

Corollaire 6.3.5 (Observation des succès) *Soient A un agent* `lcc` *et c une contrainte linéaire.*

Si $A^* \vdash_{NLLI(\mathcal{C},\mathcal{D})} \vec{\diamond} \otimes c$, alors il existe une contrainte d telle que $d \vdash_{\mathcal{C}} c$ et $(\emptyset; 1; A) \longrightarrow (\emptyset; d; 1)$, i.e. d est un succès pour A.

Preuve. On applique le Théorème précédent à la configuration $(\emptyset; 1; A)$, avec ϕ une contrainte ($n = 0$), en remarquant qu'alors, l'hypothèse "configuration définie" est inutile. ■

Corollaire 6.3.6 (Observation des suspensions) *Soient A un agent* `lcc` *défini et $\phi = c \otimes (d_1 \multimapdot A_1) \otimes \cdots \otimes (d_n \multimapdot A_n)$ une formule suspension.*

Si $A^* \vdash_{NLLI(\mathcal{C},\mathcal{D})} \vec{\diamond} \otimes \phi$, alors A suspend avec le store c sur les contraintes $d_1, ..., d_n$.

Preuve. Evident, par définition d'une suspension (Chapitre 2, Définition 2.2.6), en appliquant le Théorème précédent à la configuration $(\emptyset; 1; A)$. ■

6.4 L'observation des suspensions dans les `cc` monotones

D'après la Proposition 2.2.9 (Chapitre 2.1), la traduction de `cc` dans `lcc` respecte l'observation des succès.

Pour l'observation des suspensions, on modifie légèrement la traduction du Chapitre 2.1. On ne change pas la traduction des agents; pour les contraintes:

$c^\circ = !c$, si c est une contrainte atomique et $c \notin \mathcal{C}_s$
$c^\circ = c$, si c est une contrainte de synchronisation
$(c \wedge d)^\circ = c^\circ \otimes d^\circ$ $\qquad (\exists x c)^\circ = \exists x c^\circ$

Cette traduction permet de distinguer entre les contraintes de synchronisation et les autres contraintes (monotones), et le Corollaire 6.3.6 est encore valide.

Chapter 7

Conclusion et perspectives

Nous avons étudié des connections entre la logique et les programmes concurrents par contraintes (`cc`) dans le paradigme *programme - formule* et *calcul - recherche de preuve*, et cela nous a conduits à introduire une version mixte de la logique linéaire, combinant des connecteurs commutatifs et des connecteurs non-commutatifs. Cette logique mixte permet de caractériser des observations fines: les suspensions d'un agent `cc`.

Précisément nous avons été amenés à distinguer deux types de contraintes dans ces langages: d'une part des contraintes complexes (construites avec conjonction et quantification existentielle), habituelles dans les applications de la programmation logique par contraintes à la résolution de problèmes combinatoires, d'autre part des contraintes de synchronisation, adaptées aux problèmes de spécification de protocoles. Cette distinction reflète bien les deux styles de programmes concurrents par contraintes: d'un côté les applications de `cc` à la résolution de problèmes, de l'autre les applications de `cc` à la spécification de protocoles dans des systèmes réactifs. Dans le premier cas, le système de contraintes a une structure riche avec implications entre contraintes atomiques. Dans le second cas en revanche, les systèmes de contraintes de synchronisation ont une structure beaucoup plus simple, la notion de suspension est fondamentale dans les programmes de protocoles, et la correspondance que nous proposons montre que ces suspensions sont caractérisables logiquement.

Un développement naturel de ce travail est l'étude sémantique de la logique linéaire mixte en liaison avec le développement d'outils d'analyse de programmes. Dans la Section 4.2.2, nous avons donné un exemple de preuve de propriété de sûreté de programme `cc` en utilisant la sémantique des phases de la logique linéaire commutative. Il serait intéressant de poursuivre cette étude avec la sémantique des phases de la logique mixte, qui fournirait en particulier une nouvelle sémantique dénotationnelle des `cc` qui caractérise des observables plus fins que ceux traités jusqu'à présent [52, 23, 49].

Dans une perspective plus large, le lien, à la suite de [35, 25], entre la sémantique logique et les sémantiques usuelles de la concurrence (en particulier la bisimulation) à la lumière de la théorie de la démonstration, fournit une autre perspective de recherche.

Du point de vue logique, les deux questions évidentes et fondamentales sont:
– la sémantique des preuves: sémantiques catégoriques à la suite de [53, 4], jeux, sémantique cohérente [17, 44]...
– et les réseaux: critère de correction géométrique [36, 15], liens avec les jeux...

Notons enfin que la logique mixte apparaît comme une approche possible du "dilemme logique" dans le processus d'élimination des coupures dans le calcul des séquents classique (conflit qui conduit naturellement à orienter les connecteurs [10]).

Bibliography

[1] M. Abrusci. Phase semantics and sequent calculus for pure non-commutative classical linear logic. *Journal of Symbolic Logic*, 56(4), 1991.

[2] M. Abrusci. Non-commutative proof-nets. In *Cambridge University Press, LMSLNS 222*, 1995.

[3] J.M. Andreoli and R. Pareschi. Linear objects: logical processes with built-in inheritance. *New Generation Computing*, 9, 1991.

[4] M. Barr. $*$-autonomous categories. *Springer Lecture Notes in Mathematics*, 752, 1979.

[5] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96, 1992.

[6] E. Best, F.S. de Boer, and C. Palamidessi. Concurrent constraint programming with information retrieval. Esprit project ACCLAIM final report, 1994.

[7] G. Boudol. Asynchrony and the π-calculus. Technical Report RR 1702, INRIA, 1992.

[8] Ph. Codognet and F. Rossi. NMCC programming: constraint enforcement and retraction in CC programming. In *Proceedings ICLP'95, Int. Conf. on Logic Programming*, 1995.

[9] V. Danos. *La logique linéaire appliquée à l'étude de divers processus de normalisation (principalement du λ-calcul)*. PhD thesis, Université Paris 7, 1990.

[10] V. Danos, J.B. Joinet, and H. Schellinx. A new deconstructive logic: linear logic. In *Preprint 936, Dept. of Mathematics, University of Utrecht, To appear in the J. of Symbolic Logic*, 1995.

[11] Ph. de Groote. Partially commutative linear logic: sequent calculus and phase semantics. In *Proofs and Linguistic Categories, Proceedings 1996 Roma Workshop*. In V. M. Abrusci and C. Casadio, eds. Cooperativa Libraria Universitaria Editrice Bologna, 1996.

[12] Endberg and G. Winskel. Linear logic on petri nets. *Springer Lecture Notes in Computer Science*, 803, 1994.

[13] F. Fages. *Programmation logique par contraintes.* Ellipses, 1996.

[14] F. Fages. Constructive negation by pruning. *J. of Logic Programming*, 32(2), 1997.

[15] A. Fleury. *La règle d'échange: logique linéaire multiplicative tressée.* PhD thesis, Université Paris 7, Novembre 1996.

[16] F.S. de Boer, M. Gabbrielli, and C. Palamidessi. Proving correctness of constraint logic programming with dynamic scheduling. In *Proceedings of SAS'96, Springer LNCS 1145*, 1996.

[17] J.Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1), 1987.

[18] J.Y. Girard. Towards a geometry of interaction. *Contemporary Mathematics*, 92:69–108, 1989.

[19] J.Y. Girard. A new constructive logic: classical logic. *Mathematical Structures in Computer Science*, 1(3):255–296, 1991.

[20] J.Y. Girard. *Linear logic: its syntax and semantics.* Cambridge University Press, LMSLNS 222, 1995.

[21] P. Van Hentenryck, V. Saraswat, and Y. Deville. Design, implementation and evaluation of the constraint language `cc(fd)`. In *Constraint Programming: Basics and Trends*, pages 293–316, 1995.

[22] J. Jaffar and J-L. Lassez. Constraint logic programming. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages, Munich, Germany*, pages 111–119. ACM, January 1987.

[23] R. Jagadeesan, V. Shanbhogue, and V.A. Saraswat. Angelic nondeterminism in concurrent constraint programming. Technical report, Xerox Parc, 1991.

[24] J. Jourdan. *Concurrence et coopération de modèles multiples dans les langages CLP et CC: vers une méthodologie de programmation par modélisation.* PhD thesis, Université Paris 7, 1995.

[25] N. Kobayashi and A. Yonezawa. Logical, testing and observation equivalence for processes in a linear logic programming. Technical Report 93-4, Department of Computer Science, University of Tokyo, 1993.

[26] N. Kobayashi and A. Yonezawa. Asynchronous communication model based on linear logic. *Formal Aspects of Computing*, 3, 1994.

[27] R.A. Kowalski. Predicate logic as a programming language. In *Information Processing*, pages 569–574, North-Holland, 1974.

[28] Y. Lafont. Interaction nets. In *Proc. International Conference on Principles of Programming Languages*, ACM, 1990.

[29] Y. Lafont. The finite model property for various fragments of linear logic. *To appear in the J. of Symbolic Logic*, 1995.

[30] J. Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65(3):154–170, 1958.

[31] P. Lincoln and V.A. Saraswat. Proofs as concurrent processes. Parc Xerox Tech. Report, 1991.

[32] M. Maher and J. Jaffar. Constraint logic programming: a survey. *Journal of Logic Programming*, 19-20, 1994.

[33] M.J. Maher. Logic semantics for a class of committed-choice programs. In *Proceedings of ICLP'87, International Conference on Logic Programming*, 1987.

[34] N. Marti-Oliet and J. Meseguer. From petri nets to linear logic. In *Proceedings of Category Theory and Computer Science*, pages 313–340, Springer LNCS 389, 1989.

[35] N. Mendler, P. Panangaden, P.J. Scott, and R.A.G. Seely. A logical view of concurrent constraint programming. *Nordic J. of Computing*, 2, 1995.

[36] F. Métayer. *Une étude homologique des réseaux*. PhD thesis, Université Paris 7, 1994.

[37] D. Miller. The π-calculus as a theory in linear logic: preliminary results. In *Proceedings Workshop on Extensions of Logic Programming*, Springer LNCS 660, 1992.

[38] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1), 1992.

[39] R. Mohring. Computationally tractable classes of ordered sets. *NATO ASI Series (I.Rival, ed)*, 255, 1989.

[40] V. Novák. Cyclically ordered sets. *Czechoslovak Math. Journal*, 32(107):460–473, 1982.

[41] M. Okada. Girard's phase semantics and a higher-order cut-elimination proof, 1994.

[42] G. Perrier. Concurrent programming in linear logic. Technical Report CRIN 95-R-052, INRIA-Lorraine, 1995.

[43] A. Podelski and G. Smolka. Operational semantics of constraint logic programming with coroutining. In *Proceedings of ICLP'95, International Conference on Logic Programming*, Tokyo, 1995.

[44] Ch. Retoré. *Réseaux et séquents ordonnés.* PhD thesis, Université Paris 7, 1993.

[45] J.A. Robinson. A machine-oriented logic based on the resolution principle. *Machine Intelligence*, 12(1):23–41, 1965.

[46] P. Ruet. Logical semantics of concurrent constraint programming. In *Proceedings of CP'96, 2^{nd} International Conference on Constraint Programming, Cambridge, MA, Springer LNCS 1118*, 1996.

[47] P. Ruet. Non-commutative linear logic with mobilities. *Presented at the Logic Colloquium'96, San Sebastian, Spain*, Bulletin of Symbolic Logic 3-2:274–275, Jan. 1997.

[48] P. Ruet and F. Fages. Concurrent constraint programming and non-commutative logic. *Computer Science Logic'97*, `http://www.dmi.ens.fr/~ruet`, 1997.

[49] V.A. Saraswat. *Concurrent constraint programming.* ACM Doctoral Dissertation Awards. MIT Press, 1993.

[50] V.A. Saraswat and P. Lincoln. Higher-order linear concurrent constraint programming. Parc Xerox Technical Report, 1992.

[51] V.A. Saraswat and M. Rinard. Concurrent constraint programming. In *Proceedings 17th ACM Symposium on Principles of Programming Languages*, pages 232–245, 1990.

[52] V.A. Saraswat, M. Rinard, and P. Panangaden. Semantic foundations of concurrent constraint programming. In *POPL'91: Proceedings 18th ACM Symposium on Principles of Programming Languages*, 1991.

[53] R.A.G. Seely. Linear logic, $*$-autonomous categories and cofree coalgebras. *Contemporary Mathematics*, 92, 1989.

[54] P.J. Stuckey. Constructive negation for constraint logic programming. *Information and Computation*, 118(1), 1995.

[55] C. Tse. The design and implementation of an actor language based on linear logic. Master Thesis, MIT, 1994.

[56] J. Valdes, R.E. Tarjan, and E.L. Lawler. The recognition of series-parallel digraphs. *Journal of Computing*, 11(2), May 1982.

[57] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming.* Logic Programming Series. MIT Press, Cambridge (MA), 1989.

[58] D.N. Yetter. Quantales and (non-commutative) linear logic. *J. of Symbolic Logic*, 55(1), 1990.

Contents