Programmation et Logique

Christophe Fouqueré

Laboratoire d'Informatique de Paris-Nord
Université de Paris 13 - CNRS 7030

Objectifs / Problèmes / Contexte

Après avoir été vue comme un système de calcul (algèbre du calcul propositionnel de Boole, 1847), la logique est maintenant (aussi) considérée comme un paradigme de programmation à part entière (1965, méthode de résolution de Robinson), (Prolog de Colmerauer, 1973). Nous nous intéresserons dans ce cours à comprendre dans quelle mesure nous pouvons répondre aux questions suivantes :

- comment peut-on interpréter l'exécution de programmes d'un point de vue logique ?
- Comment extraire de la logique un langage de programmation ?
- Les propriétés d'un tel langage peuvent-elles aider à la compréhension de la logique ?
- Quelles sont les particularités de la programmation logique vs d'autres paradigmes ?

La théorie de la preuve permet en effet deux modes opératoires :

- **Réduction de preuves : programmation fonctionnelle**
  - programme == preuve avec coupures
  - calcul == exécution du programme
  - == normalisation de la preuve
  - == élimination des coupures
- **Recherche de preuves : programmation logique**
  - programme == ensemble de formules
  - calcul == construction (montante) de preuves sans coupure
  - séquent == état d'un calcul

Nous verrons comment ces deux perspectives éclairent logique et programmation sous des aspects complémentaires. L'intérêt pour la programmation logique s'est en effet trouvé relancé depuis une dizaine d'années après la mise à jour de propriétés structurelles des preuves en logique linéaire (polarisation, focalisation).

Plan du cours

A Théorie de la démonstration

- Calcul des séquents intuitionniste : Normalisation, Elimination des coupures et Interprétation algorithmique des preuves.
- Logique linéaire : Calcul des séquents.
- Représentation géométrique des preuves : réseaux de preuves, critères de correction, théorème de séquentialisation.

B Programmation par preuves

- Résolution et programmation logique (PROLOG).
- Programmation en logique linéaire (commutative et non-commutative).
- Focalisation et polarisation des preuves : application à la programmation concurrente.

BIBLIOGRAPHIE

B. COURCELLE : Logique et Informatique, une introduction (INRIA, Collection Didactique, 1991).
J.Y. GIRARD : Proof theory and logical complexity (Bibliopolis, Napoli, 1987).
J.Y. GIRARD, Y. LAFONT & P. TAYLOR : Proofs and Types (Cambridge Tracts in Theoretical Computer Science 7, Cambridge University Press, 1989).
A. S. TROELSTRA, H. SCHWICHTENBERG, Basic Proof Theory (Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, Second, revised edition 2000).
A.S. TROELSTRA, Lectures in linear logic, (CSLI Stanford, Lecture Notes Series nr. 29, 1992).

I. Mathématiques = Logique + Calcul ?

- La logique est réductible au calcul : Descartes (géométrie analytique), Leibniz (langage universel)
- La logique vue comme un système de calcul : Boole (algèbre du calcul propositionnel, 1847)
- La logique comme outil de formalisation mathématique : Frege (logique du premier ordre, 1879), Hilbert et Ackerman (1928)
  - **théorie de la preuve** : une dérivation formelle est une suite de symboles satisfaisant certaines règles et menant à une conclusion.
  - **théorie des modèles** : vérité logique et sémantique de la logique
- La logique comme paradigme de programmation : Robinson (1965, méthode de résolution), Colmerauer (1973)

Un peu d'histoire

- Aristote (-384, -322) : syllogisme et déduction formelle : "Tous les hommes sont mortels, les Athéniens sont des hommes, donc ils sont mortels"
- Leibniz (1646, 1716) : *Characteristica Universalis* et *Calculus Ratiocinator*
- Frege (1848, 1925) *Ecriture de concepts* : syntaxe graphique ($\forall$, $\exists$, ...)
- Russell (1872, 1970) et Whitehead (1861, 1947) : *Principia Mathemetica* (1910) : formules, déductions, axiomes, hypothèses et conclusions

... séisme !

- Hilbert (1862, 1943) : recherche d'une mécanisation absolue des raisonnements
- Gödel (1906, 1978) : résultats d'incomplétude (1930)
  - **incomplétude** : il n'existe pas d'axiomatisation finie de système mathématique intégrant l'arithmétique des entiers naturels
  - **indécidabilité** : il n'existe pas de procédure automatisable permettant de savoir dans le cas général si un énoncé est vrai ou non.
- Brouwer (1881, 1966) : **école intuitionniste** : contre la logique classique qui ne permet pas nécessairement de construire une solution, mais seulement d'en montrer l'existence.

Intuitionnisme et constructivisme

N'est pas intuitionniste une démonstration de l'existence d'irrationnels a et b tels que a^b soit rationnel ($\sqrt{2}^{\sqrt{2}}$ sachant que $\sqrt{2}$ est irrationnel)
Une telle démonstration passe par l'utilisation du tiers-exclu $A \vee \neg A$.
La *vérité* n'est plus seulement le fondement de la logique, il faut qu'il y ait *preuve*. Une preuve est constructive par exemple si :

- *Propriété d'existence* : Si une formule $\exists x A(x)$ est prouvable, la preuve doit permettre d'exhiber un x_0 tel que $A(x_0)$ est aussi prouvable.
- *Propriété de disjonction* : Si une formule $A \vee B$ est prouvable alors soit A est prouvable, soit B est prouvable et la preuve de $A \vee B$ permet de savoir lequel des deux l'est.

Exemple de théorème démontrable en logique classique mais non en logique intuitionniste (Girard) : $\exists x(\forall y A(x) \Rightarrow A(y))$.

Sémantique des preuves intuitionnistes

Heyting (1898, 1980) et Kolmogorov (1903, 1987)
La sémantique des preuves est définie inductivement comme :

- la preuve d'une formule atomique est une méthode automatisable permettant de vérifier que la formule est vraie (p.e. la donnée d'un algorithme de calcul).
- la preuve d'une conjonction $A \wedge B$ est formée d'une preuve de A *et* d'une preuve de B.
- la preuve d'une disjonction $A \vee B$ est formée d'une preuve de A *ou* d'une preuve de B.
- la preuve d'une implication $A \Rightarrow B$ est un processus effectivement calculable transformant toute preuve de A en une preuve de B.
- la preuve d'une négation $\neg A$ est un processus effectivement calculable qui transforme toute preuve de A en une preuve d'une contradiction.

- la preuve d'une quantification existentielle $\exists x A(x)$ est la donnée d'un élément x_0 du domaine de quantification et d'une preuve de $A(x_0)$.
- la preuve d'une quantification universelle $\forall x A(x)$ est la donnée d'un processus effectivement calculable qui transforme tout élément x_0 du domaine de quantification en une preuve de $A(x_0)$.

Remarques :

- processus effectivement calculable ?
- montrer $\neg A$ c'est montrer $A \Rightarrow \bot$

Ordres logiques

- Logique d'ordre 0 : logique des propositions. Combinaison de propositions indépendantes
- Logique d'ordre 1 : logique des prédicats. Quantification sur des domaines d'éléments, fonctions sur le domaine. Les prédicats sont des propositions sur des termes, i.e. applications de fonctions sur des variables parcourant le domaine.
- Logique d'ordre 2 : variables sur prédicats (p.e. axiome de récursivité)
- etc

Langage de la logique (1)

- vocabulaire :
  - ensemble dénombrable de constantes $a, b, c, \ldots$
  - ensemble dénombrable de fonctions avec arité $f, g, \ldots$
  - ensemble dénombrable de variables d'individus $x, y, \ldots$
  - ensemble dénombrable de prédicats avec arité $p, q, \ldots$
  - ensemble dénombrable de variables propositionnelles $A, B, \ldots$
  - connecteurs $\vee, \wedge, \neg, \Rightarrow$
  - quantificateurs $\forall, \exists$

Langage de la logique (2)

- *termes* construits inductivement sur les fonctions, les variables d'individus et les constantes
- *formules propositionnelles* construites inductivement sur les variables propositionnelles et les connecteurs
- *formules du premier ordre* construites inductivement sur les variables propositionnelles, les prédicats et termes, les connecteurs, les quantificateurs

Variables libres et liées, renommage et substitution

II Système de preuves : présentation

Exemples :

- système hilbertien
- système de déduction naturelle
- calcul des séquents

Une théorie est la donnée :

- d'un ensemble d'axiomes ou de schémas d'axiomes
- d'un ensemble de règles d'inférence

Démontrer un théorème, c'est construire l'arbre de la preuve où la racine est le théorème démontré, où les feuilles sont les axiomes ou des instances d'un schéma d'axiomes et où on passe des fils d'un noeud au noeud père par utilisation d'une des règles d'inférence.
Les objets que l'on peut chercher à prouver sont des *jugements*.

Extension d'une théorie

... par ajout de règles ou extension du langage à une théorie donnée

- **Extension dérivable** : l'ensemble des théorèmes n'est pas modifié (la nouvelle règle est une preuve partielle générique)
- **Extension consistante** : l'ajout conserve le fait que toute formule n'est pas un théorème
- **Extension conservative** : toute formule de la théorie non étendue démontrable dans la théorie étendue l'est aussi dans la théorie non étendue

III. Logique classique

- Logique propositionnelle :
  - Variables propositionnelles interprétées vrai/faux (0/1)
  - Connecteurs $\wedge$ et $\vee$ (et/ou, multiplicatif/additif)
  - Négation $\neg$ exprimant la dualité des valeurs et connecteurs binaires.
  - Contraction et affaiblissement présentes
  - Propriétés algébriques (commutativité, associativité, distributivité, éléments neutres)
- Logique des prédicats du premier ordre :
  - Prédicats dont les arguments sont des termes avec variables et symboles fonctionnels
  - Quantifications universelle $\forall$ et existentielle $\exists$

III.1 Logique classique : Système hilbertien

Hilbert(1862,1943)
Un système hilbertien ne comporte que des axiomes et une seule règle d'inférence (le *modus ponens*) :

$$\frac{A \quad A \Rightarrow B}{B}$$

Les axiomes du système hilbertien pour la logique des propositions classique sont :

- **Conjonction**

$$A \Rightarrow B \Rightarrow A \wedge B \quad A \wedge B \Rightarrow A \quad A \wedge B \Rightarrow B$$

- **Disjonction**

$$A \Rightarrow A \vee B \quad B \Rightarrow A \vee B \quad A \vee B \Rightarrow (A \Rightarrow C) \Rightarrow (A \Rightarrow C) \Rightarrow C$$

- **Implication**

$$A \Rightarrow B \Rightarrow A \quad (A \Rightarrow (B \Rightarrow C)) \Rightarrow (A \Rightarrow B) \Rightarrow A \Rightarrow C$$

- **Négation**

$$A \vee \neg A \quad A \Rightarrow \neg A \Rightarrow B$$

III.2 Logique classique : Déduction naturelle

(Lukasiewicz 1878, 1956) et (Gentzen 1909, 1945)
Une preuve est une déduction composée de formules positionnées en arbre. Une formule est une conséquence des formules qui sont au-dessus d'elles. Les formules en haut des déductions peuvent être des axiomes, ou bien des hypothèses (notées entre crochets).
Chaque connecteur est pourvu de plusieurs règles d'*introduction* et de règles d'*élimination*.

Système de déduction naturelle pour la logique des propositions :

- **Conjonction**

$$\frac{\begin{array}{cc}\vdots & \vdots \\ A & B\end{array}}{A \wedge B} \qquad \frac{\vdots}{\dfrac{A \wedge B}{A}} \qquad \frac{\vdots}{\dfrac{A \wedge B}{B}}$$

- **Disjonction**

$$\frac{\vdots}{\dfrac{A}{A \vee B}} \qquad \frac{\vdots}{\dfrac{B}{A \vee B}} \qquad \frac{\begin{array}{ccc} & [A] & [B] \\ \vdots & \vdots & \vdots \\ A \vee B & C & C\end{array}}{C}$$

- **Implication**

$$\frac{\begin{array}{c}[A] \\ \vdots \\ B\end{array}}{A \Rightarrow B} \qquad \frac{\begin{array}{cc}\vdots & \vdots \\ A & A \Rightarrow B\end{array}}{B}$$

- **Négation**

$$\frac{}{A \vee \neg A} \qquad \frac{\begin{array}{cc}[A] & [A] \\ \vdots & \vdots \\ B & \neg B\end{array}}{\neg A} \qquad \frac{\begin{array}{cc}\vdots & \vdots \\ A & \neg A\end{array}}{B}$$

III.3 Logique classique : Calcul des séquents (Gentzen 1934)

$$\frac{}{A \vdash A}\ (axiom) \qquad \frac{\Gamma_1 \vdash \Gamma_2, A \quad A, \Delta_1 \vdash \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\ (cut)$$

$$\frac{\Gamma \vdash \Delta, A, B, \Pi}{\Gamma \vdash \Delta, B, A, \Pi}\ (r-exchange) \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, A}\ (r-weakening) \qquad \frac{\Gamma \vdash \Delta, A, A}{\Gamma \vdash \Delta, A}\ (r-contraction)$$

$$\frac{\Gamma, A \vdash \Delta}{\Gamma \vdash \Delta, \neg A}\ (negation) \qquad \frac{A, \Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \to B}\ (implies)$$

$$\frac{\Gamma_1 \vdash \Delta_1, A \quad \Gamma_2 \vdash \Delta_2, B}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2, A \wedge B}\ (and) \qquad \frac{\Gamma \vdash \Delta, A}{\Gamma \vdash \Delta, A \vee B}\ (or_1) \qquad \frac{\Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \vee B}\ (or_2)$$

$$\frac{\Gamma \vdash \Delta, p[y]}{\Gamma \vdash \Delta, \forall x p}\ (\forall) \qquad \frac{\Gamma \vdash \Delta, p[t/x]}{\Gamma \vdash \Delta, \exists x p}\ (\exists)$$

où y n'est pas libre dans Γ et Δ, et t est un terme arbitraire.
... et les règles pour introduction gauche des connecteurs et quantificateurs

Exemples

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A} \quad \dfrac{}{B \vdash B}}{A \to B, A \vdash B} \quad \dfrac{\dfrac{}{A \vdash A} \quad \dfrac{}{C \vdash C}}{A \to C, A \vdash C}}{A \to B, A, A \to C \vdash B \wedge C}}{A, A \to B, A \to C \vdash B \wedge C}}{A \to B, A \to C \vdash A \to B \wedge C} \quad \dfrac{\dfrac{\dfrac{}{A \vdash A} \quad \dfrac{\dfrac{}{B \wedge C \vdash B \wedge C} \quad \dfrac{}{D \vdash D}}{B \wedge C, B \wedge C \to D \vdash D}}{A, A \to B \wedge C, B \wedge C \to D \vdash D}}{A \to B \wedge C, B \wedge C \to D \vdash A \to D} \ (cut)$$

$$\dfrac{\dfrac{\dfrac{}{p(a) \vdash p(a)}}{p(a) \vdash \exists x p(x)} \quad \dfrac{\dfrac{}{p(a) \vdash p(a)}}{p(b) \vdash \exists p(x)}}{p(a) \vee p(b) \vdash \exists x p(x)} \ (I - or)$$

Propriétés fondamentales

- Théorème d'élimination de la coupure (Hauptsatz, Gentzen 1934)
- La logique intuitionniste (i.e. au plus une conclusion) est constructive : p.e. si on peut prouver $\vdash \exists x p(x)$ alors on peut exhiber un terme t qui convient.
- Forte normalisation de la déduction naturelle pour la logique intuitionniste (Prawitz 1965) : l'ordre d'élimination des coupures n'a pas d'importance, le nombre d'étapes est fini.
- Correspondance de Curry-Howard : l'évaluation en lambda-calcul typé est la normalisation.
- théorème de la déduction et principe de résolution;

IV. Logique linéaire : Girard 1987

Remarques :

- la logique intuitionniste est constructive (correspondance preuve - algorithme).
  la logique classique ne l'est pas : non déterminisme de l'élimination des coupures : gestion ensembliste des formules.
- la caractérisation des fonctions séquentielles (i.e. des morphismes entre ensembles d'états) met à jour une factorisation de ceux-ci :
  l'application d'une fonction à un argument suppose une possible duplication de celui-ci avant une application réelle linéaire.

L'idée était donc de séparer ces deux étapes, mais induit aussi une distinction sur la disjonction et la conjonction.

Structure générale :

- Opérations linéaires : connecteurs additifs, multiplicatifs
- Opérations de contrôle de la multiplicité : connecteurs exponentiels
- $A \Longrightarrow B \equiv \ !A \multimap B$: la LL "décompose" la logique classique

Calculs et représentations sémantiques :

- Calcul des séquents : élimination des coupures, ...
- Sémantique des "phases" : se substitue au modèle booléen
- Sémantique des espaces "cohérents" : sémantique des preuves
- Réseaux de preuve : critère global de preuve

IV.1 Logique linéaire : Calcul des séquents

$$\frac{}{A \vdash A}\ (axiom) \qquad \frac{\vdash \Gamma, A \quad \vdash A^{\perp}, \Delta}{\vdash \Gamma, \Delta}\ (cut)$$

$$\frac{\vdash \Delta, A, B, \Pi}{\vdash \Delta, B, A, \Pi}\ (r-exchange) \qquad \frac{\vdash \Delta}{\vdash \Delta, ?A}\ (r-weakening) \qquad \frac{\vdash \Delta, ?A, ?A}{\vdash \Delta, ?A}\ (r-contraction)$$

$$\frac{\vdash \Delta, A}{\vdash \Delta, ?A}\ (r-dereliction) \qquad \frac{\vdash ?\Delta, A}{\vdash ?\Delta, !A}\ (r-!)$$

$$\frac{\vdash \Delta_1, A \quad \vdash \Delta_2, B}{\vdash \Delta_1, \Delta_2, A \otimes B}\ (and \otimes) \qquad \frac{\Gamma \vdash \Delta, A}{\Gamma \vdash \Delta, A \oplus B}\ (or \oplus_1) \qquad \frac{\Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \oplus B}\ (or \oplus_2)$$

$$\frac{\vdash \Delta, A, B}{\vdash \Delta, A \parr B}\ (or\ \parr) \qquad \frac{\vdash \Delta, A \quad \vdash \Delta, B}{\vdash \Delta, A \,\&\, B}\ (and\ \&)$$

V. Programmation

- Comment extraire de la logique un langage de programmation ?
- Les propriétés d'un tel langage peuvent-elles aider à la compréhension de la logique ?
- Quelles sont les particularités de la programmation logique vs d'autres paradigmes ?

Quelques pointeurs :

LIF, (Colmerauer), IML (Andreoli) Marseille

Imperial College, Londres (Kowalski)

Simon Fraser University, Canada (Dahl)

Amsterdam (Apt)

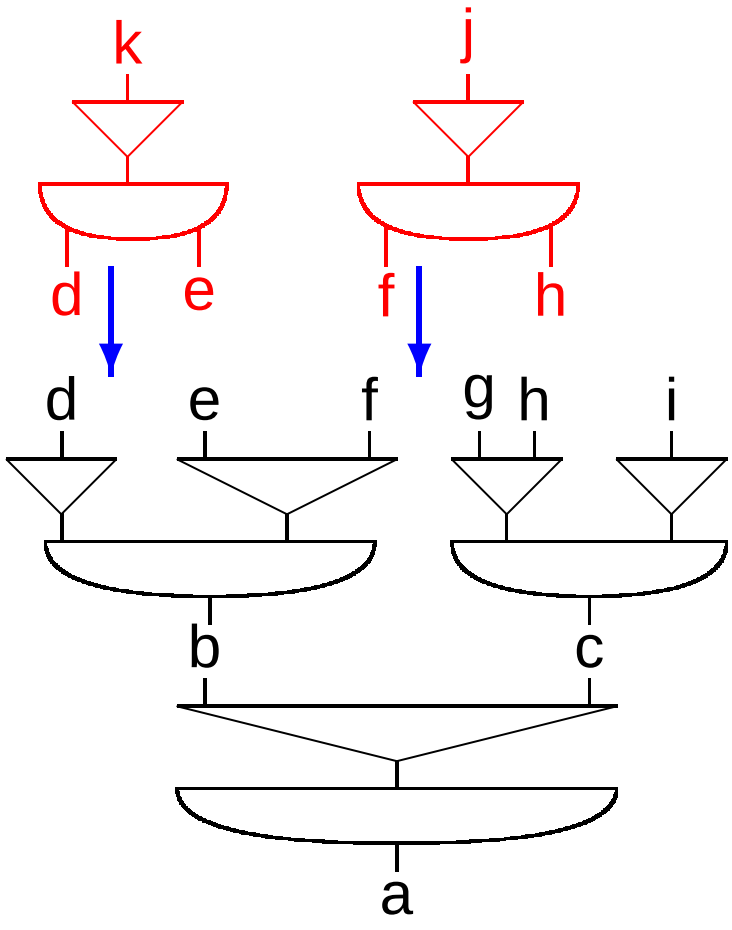
New York (Warren)

Pise (...)

LIX, Palaiseau (Miller)

...

Exemples de programmation logique



clauses

mortel(X) :- homme(X).
homme(marcus).
pompeien(marcus).
ne(marcus,40).
mort(X,79) :- pompeien(X).
mort(X,T) :- *bound(T)*, mortel(X),
ne(X,TT), (T - TT)>150.
mort(X,T) :- *bound(T)*, mort(X,TT),
T > TT.

goal : mort(marcus,1989).
result : $\longrightarrow$ true.

V.1 Programmation logique : cadre (1)

La théorie de la preuve permet deux modes opératoires :

- **Réduction de preuves : programmation fonctionnelle**
  - programme == preuve avec coupures
  - calcul == exécution du programme
  - == normalisation de la preuve
  - == élimination des coupures
- **Recherche de preuves : programmation logique**
  - programme == ensemble de formules
  - calcul == construction (montante) de preuves sans coupure
  - séquent == état d'un calcul

V.1 Programmation logique : cadre (2)

Séquent $\Delta \vdash \Gamma$

programme $\vdash$ *but*

clauses donnant le sens des constantes non logiques

$\vdash$ formules correspondant à la conclusion cherchée

On cherche à éviter le non-déterminisme non computationnellement important :

- associativité, commutativité des connecteurs
- calcul sur les clauses du programme (et qui ne fait pas intervenir le but)

La recherche de preuves est **orientée par le but** : Une preuve **uniforme** est une preuve sans coupure où tout séquent ayant une partie droite non atomique est la conclusion d'une règle d'introduction à droite.

Un **langage de programmation logique abstrait** est un système de preuves tel que chaque séquent prouvable admet une preuve uniforme.

V.2 Programmation logique et clauses de Horn

Le calcul des séquents de la logique classique n'est pas un langage de programmation logique abstrait (cf disjonction). Toutefois, il existe des fragments avec un système de preuves complet.

V.2 Programmation logique et clauses de Horn (2)

Cas du langage Prolog (Colmerauer 1973, Kowalski 1974) : une *clause de Horn* est une formule clausale ($\forall x (\bigvee_i L_i)$), universellement fermée (pas de variables libres), avec au plus un littéral positif.

$\forall x\ p$	fait	$p.$
$\forall x\ (p_1 \vee \neg p_1 \vee \cdots \vee \neg p_n)$	clause définie (ou **règle**)	$p :- p_1, \ldots, p_n.$
$\forall x\ (\neg p_1 \vee \cdots \vee p_q)$	clause négative (ou **but**)	$? - p_1, \ldots, p_q.$

Un **programme défini** P est un ensemble de clauses définies C_j.

la réponse au but R sur le programme P

est l'ensemble des substitutions σ tel que

$\bigwedge_j C_j \vdash R\sigma$.

Prolog : exemples

$nat(0).$
$nat(s(X)) :- nat(X).$
$?- nat(s(0)).$
$\longrightarrow true.$
$?- nat(s(X)).$
$\longrightarrow X = 0.$
$\longrightarrow X = s(0).$
$\longrightarrow X = s(s(0)). \ldots$
$\longrightarrow \ldots$

$add(X, 0, X).$
$add(X, s(Y), s(Z)) :- add(X, Y, Z).$
$?- add(s(0), s(0), Z)$
$\longrightarrow Z = s(s(0)).$
$?- add(X, Y, s(s(s(s(0))))), add(X, s(s(0)), Y)$
$\longrightarrow X = s(0), Y = s(s(s(0))).$

$?- f(X, g(X), Y) = f(h(Z), T, Z).$
$\longrightarrow X = h(Z), Y = Z, T = g(h(Z)).$

Prolog : considérations sémantiques

- constructivité du fragment de Horn (stabilité par produit des modèles)
- sémantique déclarative : intersection des modèles de Herbrand (algèbre de générateurs les faits)
- sémantique opérationnelle induite par le principe de résolution

Prolog : SLD-résolution

1. Principe de résolution :

   $$S \cup \{\bigvee L_i \vee A, \bigvee L'_j \vee \neg A\} \rightarrow S \cup \{\bigvee L_i \vee A, \bigvee L'_j \vee \neg A, \bigvee L_i \bigvee L'_j\}$$

   (cas général : substitution sur les littéraux)

Theorem (Robinson 1965)

Un ensemble de clauses S n'est pas satisfaisable ssi la clause vide est dérivable par résolution en un nombre fini d'étapes.

2. SLD-résolution : (Selection, Linear, Definite)
   - Une des clauses parentes est une clause définie (i.e. du programme)
   - l'autre clause parente est la résolvante la plus récente obtenue d'un but (linéarité)
   - sélection déterministe de cette clause

Theorem

La SLD-résolution est correcte et complète pour le langage des clauses de Horn.

V.3 Harrop : un peu plus que Horn

- une formule de Harrop héréditaire est une formule construite sur *true*, $\wedge$, $\forall$, $\Longrightarrow$ (avec contrainte sur les emboîtements d'implications)
- le calcul est intuitionniste.
- Un séquent s'écrit : *programme : règle* $\vdash$ *but*.
- Preuves normalisées :
  - Décomposition du but :
    $$\frac{P : \vdash G_1 \quad P : \vdash G_2}{P : \vdash G_1 \wedge G_2} \qquad \frac{P \cup \{R\} : \vdash G}{P : \vdash R \Longrightarrow G} \qquad \frac{P : \vdash G[c/x]}{P : \vdash \forall x G}$$
  - Sélection de la règle : $\dfrac{\{R\} \cup P : R \vdash A}{\{R\} \cup P : \vdash A}$
  - Décomposition/application de la règle :
    $$\frac{}{P : \vdash true} \qquad \frac{}{P : A \vdash A}$$
    $$\frac{P : R_i \vdash A}{P : R_1 \wedge R_2 \vdash A} \qquad \frac{P : \vdash G \quad P : R \vdash A}{P : G \Longrightarrow R \vdash A} \qquad \frac{P : R[t/x] \vdash A}{P : \forall x R \vdash A}$$

V.4 Programmation logique en logique linéaire : Andreoli 1992

Andreoli montre en 1992 que la logique linéaire en totalité peut donner lieu à un langage de programmation logique abstrait. Deux propriétés essentielles :

- **Polarisation** : la dualité entre connecteurs (e.g. et/ou) exprime une dualité calculatoire.
- **Focalisation** : les connecteurs sont essentiellement n-aires.

V.4 Programmation logique en logique linéaire : Polarisation (1)

La normalisation des preuves passe par l'étude des équivalences structurelles entre preuves d'un même séquent : les permutations entre applications de connecteurs sont non computationnellement signifiants.
On montre que certains connecteurs peuvent toujours être immédiatement décomposés : connecteurs **asynchrones**

Asynchrones	$\bot, \parr, ?, \top, \&, \forall$
Synchrones	$1, \otimes, !, 0, \oplus, \exists$

Exemple :

$$\dfrac{\dfrac{\dfrac{\dots}{\vdash \Gamma, A, B, C}}{\vdash \Gamma, A \parr B, C} \quad \dfrac{\dots}{\vdash \Delta, D}}{\vdash \Gamma, \Delta, A \parr B, C \otimes D} \quad \rightarrow \quad \dfrac{\dfrac{\dots}{\vdash \Gamma, A, B, C} \quad \dfrac{\dots}{\vdash \Delta, D}}{\vdash \Gamma, \Delta, A, B, C \otimes D}}{\vdash \Gamma, \Delta, A \parr B, C \otimes D}$$

V.4 Programmation logique en logique linéaire : Polarisation (2)

- les règles asynchrones sont déterministes : elles ne modifient pas le contexte
- la notion de connecteurs asynchrones est implicitement présente dans le calcul sur formules de Horn ou de Harrop
- asynchronie + distributivité : les formules alternent des couches asynchrones / synchrones (positives / négatives)

V.4 Programmation logique en logique linéaire : Focalisation

Theorem

Si toutes les formules d'un séquent prouvable en LL sont synchrones (i.e. leur connecteur principal), alors il existe au moins une formule dont tous les connecteurs synchrones de plus haut niveau sont décomposables.

En conséquence, la recherche de preuves peut être normalisée en itérant sur les étapes :

- décomposition des formules asynchrones
- sélection d'une formule synchrone
- décomposition de la formule sélectionnée jusqu'aux couches asynchrones

V.4 Programmation logique en logique linéaire : Essentiel du lan

But :

$$G := g \mid !g \mid G \otimes G \mid G \oplus G \mid \exists x G$$

$$g := X \mid ?X \mid g \parr g \mid g \,\&\, g \mid \forall x G$$

Méthode (clause) : $\forall x (G \multimap X_1 \parr \ldots \parr X_n)$

Un programme est un ensemble de méthodes dont l'exécution sur un but est régie par la **règle de progression** :

$$\frac{P, G \multimap V_1 \parr V_2 \vdash W, V_1 : Z \Downarrow G}{P, G \multimap V_1 \parr V_2 \vdash W, V_1 : V_2, Z \Uparrow}$$

Theorem

Le calcul est un langage de programmation logique abstrait correct et complet pour la logique linéaire entière.

V.4 Programmation logique en logique linéaire : Un exemple

Simulation de la synchronisation entre 2 processus P et Q :

Comportements des processus :

$P \circ\!\!- C(m) \circ\!\!- \mathrm{ack} \circ\!\!- P$

$Q \circ\!\!- \bot \circ\!\!- C(x) \circ\!\!- (Q \parr \mathrm{ack})$

Méthodes correspondantes :

$p_1 : [u \parr C(m)] \multimap P$

$p_2 : P \multimap (\mathrm{ack} \parr u)$

$q_1 : (v \parr \bot) \multimap Q$

$q_2 : (Q \parr \mathrm{ack}) \multimap (v \parr C(x))$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P, Q \Uparrow}{Q \Downarrow P}}{u, Q, \mathrm{ack} \Uparrow}\, p_2}{u \Downarrow Q \parr \mathrm{ack}}\, q_2}{u, v, C(m) \Uparrow}}{u, C(m) \Downarrow v \parr \bot}\, q_1}{Q, u, C(m) \Uparrow}}{Q \Downarrow u \parr C(m)}\, p_1}{P, Q \Uparrow}$$

V.5. Programmation concurrente en LL : Réseaux de preuve (2)

A tout lien l est associé un ensemble $\mathcal{S}(l)$ de graphes appelés *positions de switch*. $\mathcal{S}(l) = \{l\}$sauf pour le $\parr$-link. $\mathcal{S}(\parr\text{-link})$ est défini par les switches :

Un *switching* s d'une structure de preuve Θ est une fonction qui associe une position de switch $s(l) \in \mathcal{S}(l)$ à tout lien l de Θ. La *structure de preuve switché* $s(\Theta)$ est le graphe dont les sommets sont les formules étiquetant Θ, et dont les arêtes sont celles données par la fonction de switch s.

V.5. Programmation concurrente en LL : Réseaux de preuve (2)

Un *réseau de preuve* est une structure de preuve tel que toutes les structures de preuves switchées sont des arbres, i.e. des graphes acycliques et connexes.

Theorem

Une structure de preuve Θ est un réseau de preuves de conclusions Γ ssi il existe une preuve π du séquent $\vdash \Gamma$ en logique linéaire multiplicative. Ceci définit une correspondance entre preuves et réseaux de preuves.

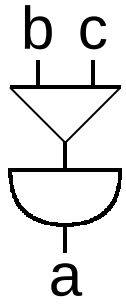
V.6 Programmation concurrente en LL

Remarques :

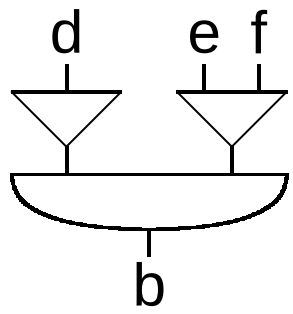
- Une méthode peut être interprétée comme un graphe orienté avec arêtes pendantes (les variables).
- La règle de progression correspond au branchement d'un graphe sur un autre graphe.
- L'environnement de calcul est aussi un graphe orienté.
- La sélection non-déterministe d'une méthode est une exécution concurrente des méthodes.
- Les preuves sont partielles : simulation de processus sans terminaison.

Exemple

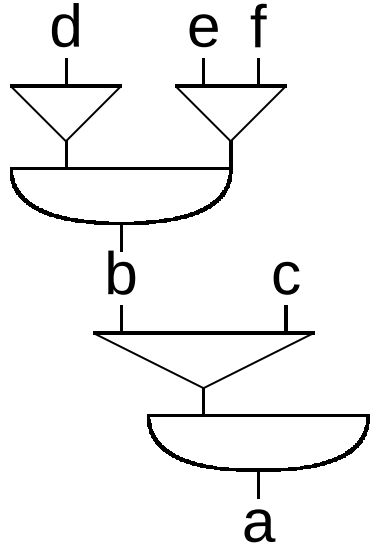
Deux méthodes α et β de types $t(\alpha) = a \multimap (b \otimes c)$ et $t(\beta) = b \multimap (d \parr (e \otimes f))$:

α: 

et

β: 

La composition des méthodes α et β est $\alpha \circ_{\{b\}} \beta$ de type $t(\alpha) \otimes t(\beta)$:

$\alpha \circ_{\{b\}} \beta$: 

Quel est le problème ?

Il faut que le graphe obtenu corresponde à une preuve partielle !

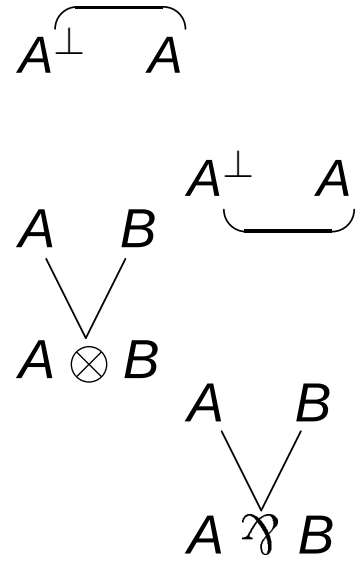
Back to proofnets :

- Girard (1987) a donné une syntaxe parallèle pour la logique linéaire multiplicative.
- Les structures manipulées sont des graphes orientés appelés **réseaux de preuves**.
- Des critères de correction permettent de distinguer ces réseaux de preuves des structures non séquentialisables.

Réseaux de preuve (1)

Une *structure de preuve* est un graphe, dont les sommets sont étiquetés par des formules, construit à partir des liens suivants :

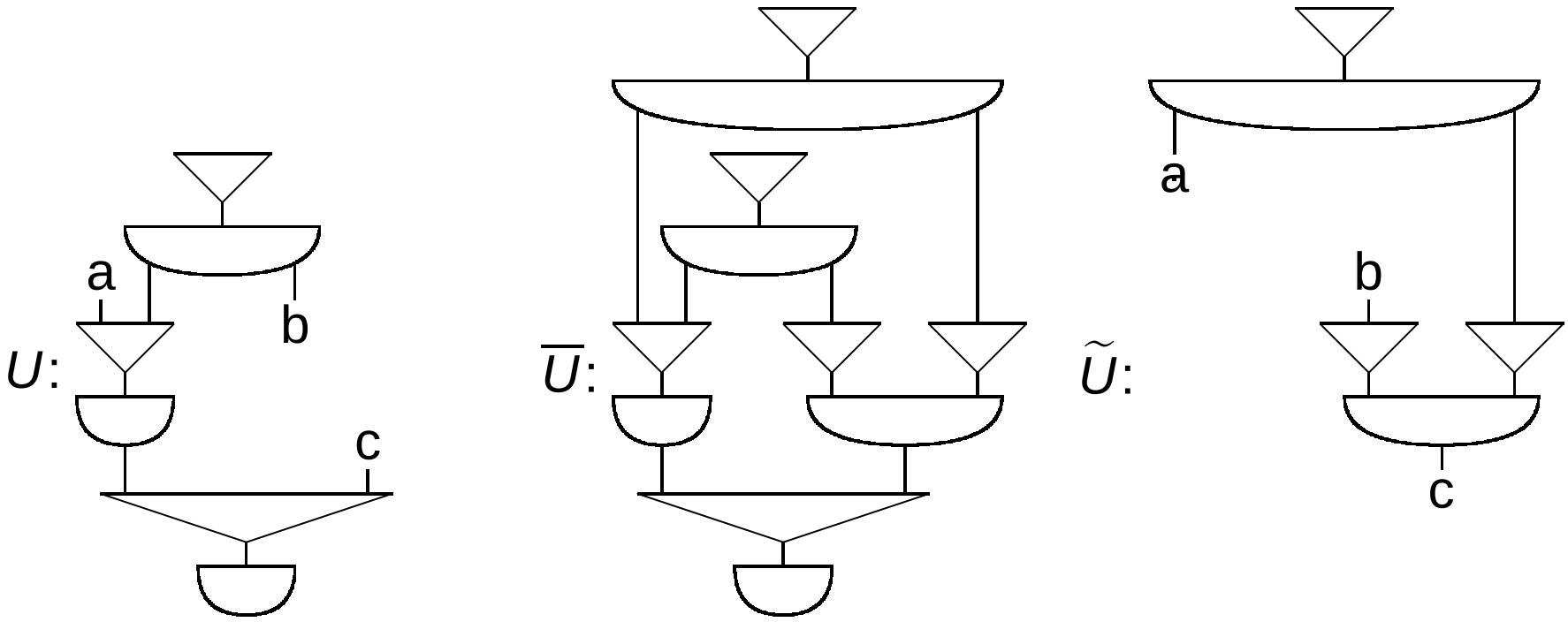
- Axiom-link (two conclusions, no premise)
- Cut-link (two premises, no conclusion)
- $\otimes$-link (two premises, one conclusion)
- $\parr$-link (two premises, one conclusion)



et toute occurence de formule est la prémisse d'au plus un lien et la conclusion d'exactement un lien.

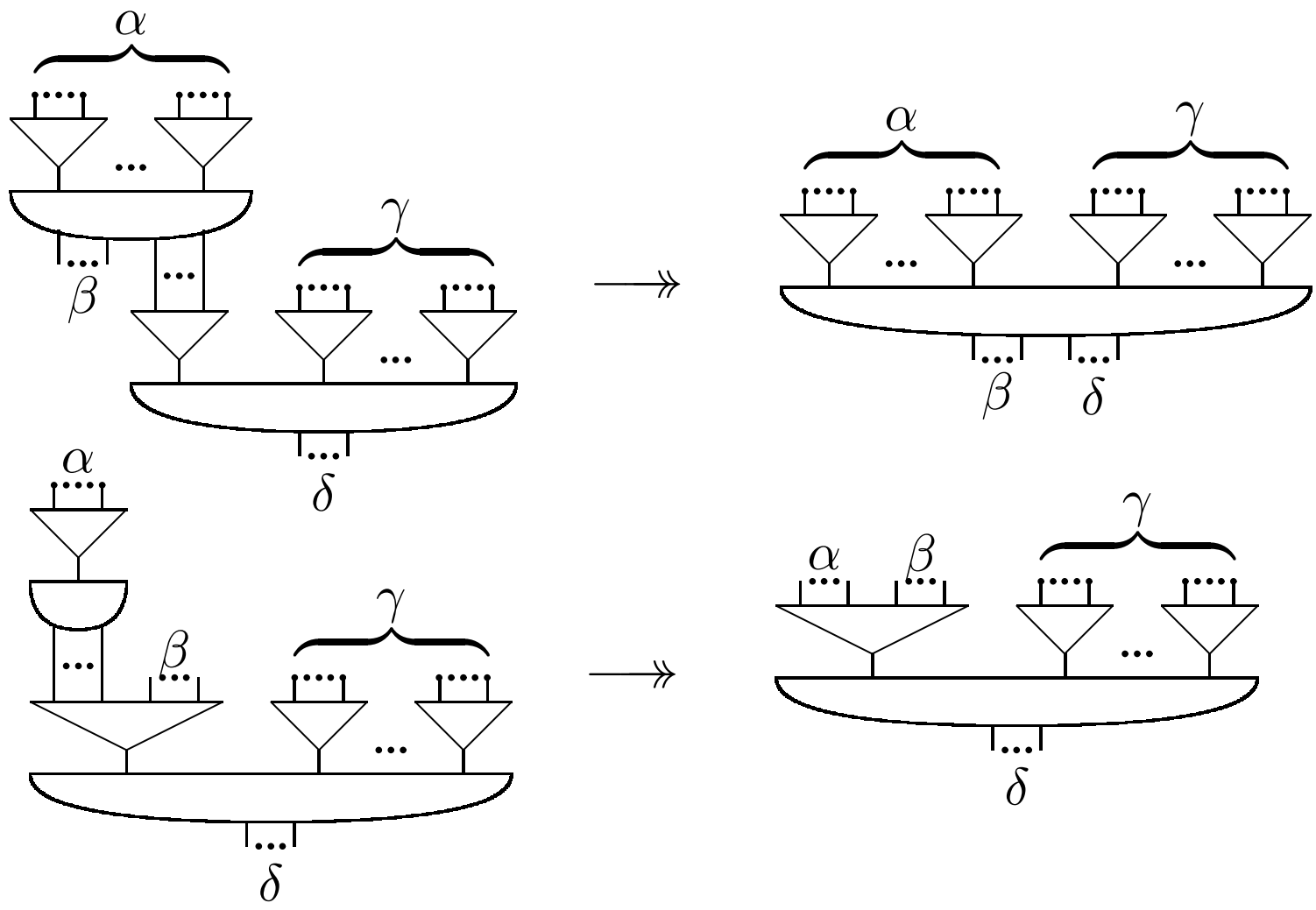
critère de correction

Un module ouvert U est *o-correct* ssi il existe une clôture de U correcte.



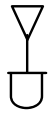
Theorem

Un o-module U est o-correct ssi U est acyclique et connectable.

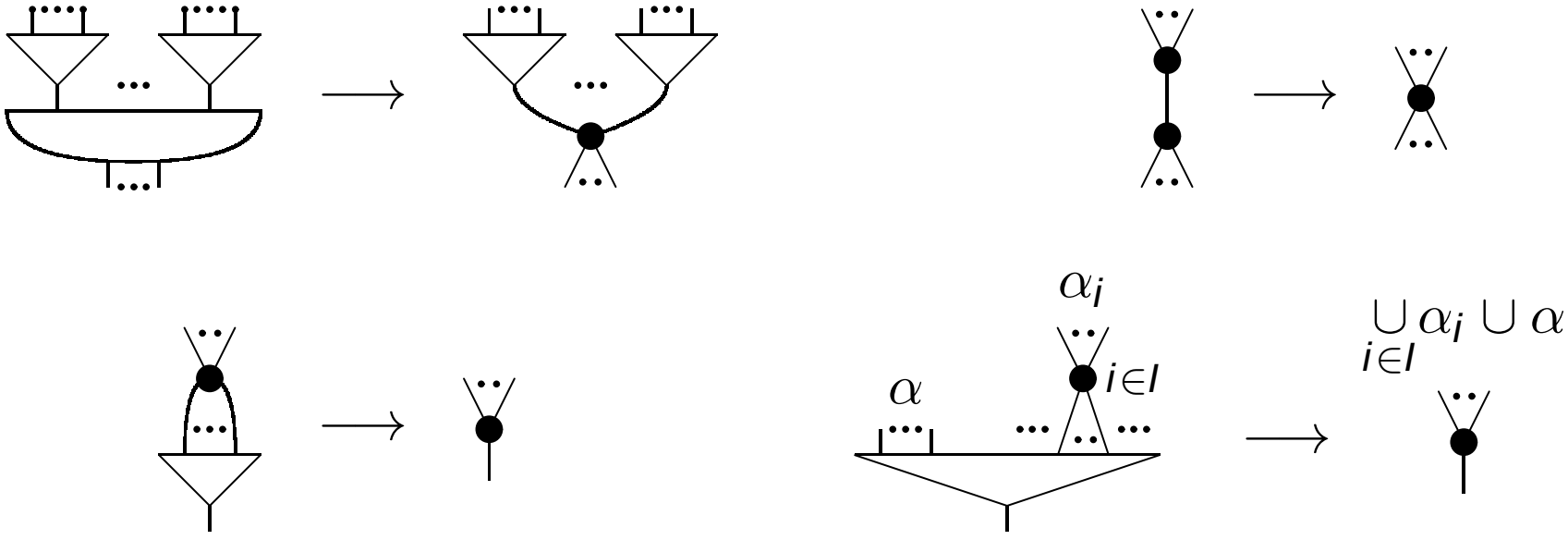


Theorem

Un module fermé (i.e. sans variable libre) M est correct ssi

$M \twoheadrightarrow^*$  .

connectabilité



où α est un ensemble d'arêtes pendantes, et il existe au moins une arête pendante par α_i.

Theorem

Un module ouvert M est connectable ssi M se contracte en un noeud unique ou bien un ensemble de noeuds ayant chacun au moins une arête pendante.

Quoi d'autre ?

- Inductive Logic Programming
- Functional Logic Programming
- Higer-order Logic Programming
- Constraint Logic Programming
- Programmation logique et langage naturel

VI. Structures en logique linéaire : ludique 1(Girard 2000)

Utilisation des propriétés fondamentales "découvertes" en programmation logique pour ré-appréhender la logique :

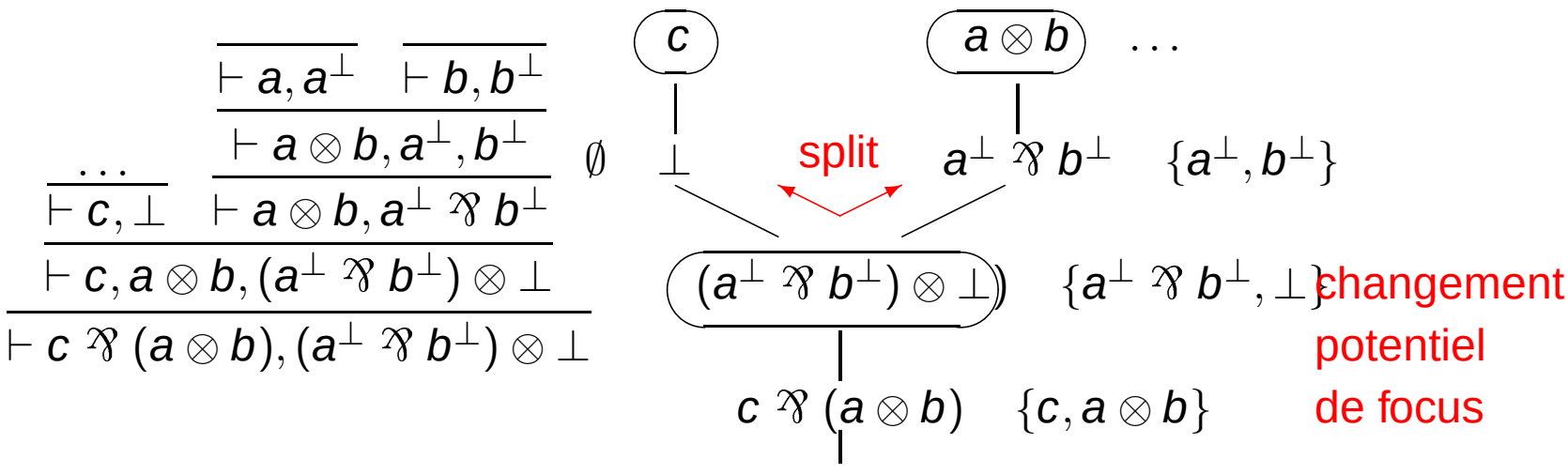
$$\dfrac{\dfrac{\dfrac{\dots}{\vdash c, \bot} \quad \dfrac{\dfrac{\vdash a, a^{\perp} \quad \vdash b, b^{\perp}}{\vdash a \otimes b, a^{\perp}, b^{\perp}}}{\vdash a \otimes b, a^{\perp} \parr b^{\perp}}}{\vdash c, a \otimes b, (a^{\perp} \parr b^{\perp}) \otimes \bot}}{\vdash c \parr (a \otimes b), (a^{\perp} \parr b^{\perp}) \otimes \bot}$$

polarisation

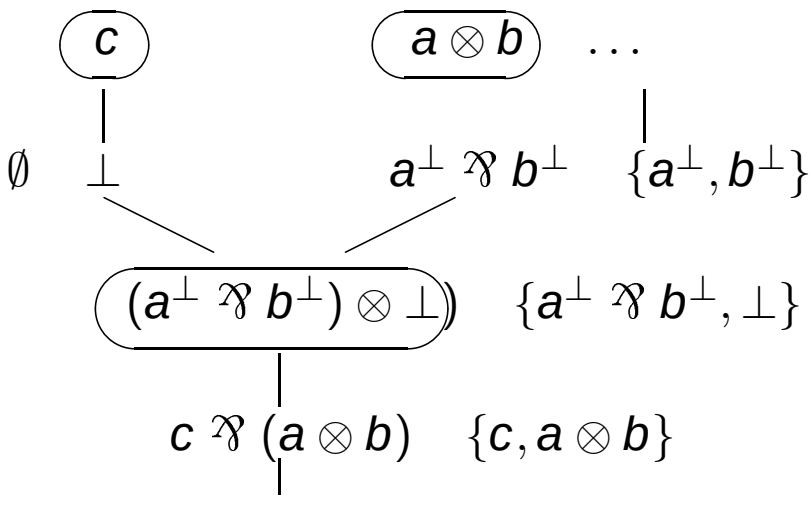
- après +

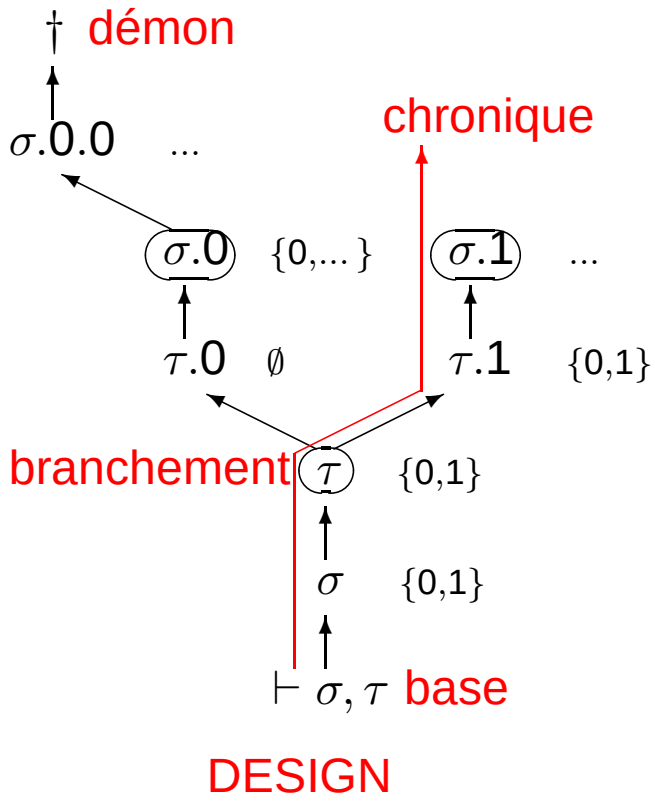
choix sur positif

VI. Structures en logique linéaire : ludique 2



VI. Structures en logique linéaire : ludique 3





DESIGN

VI. Structures en logique linéaire : ludique 4

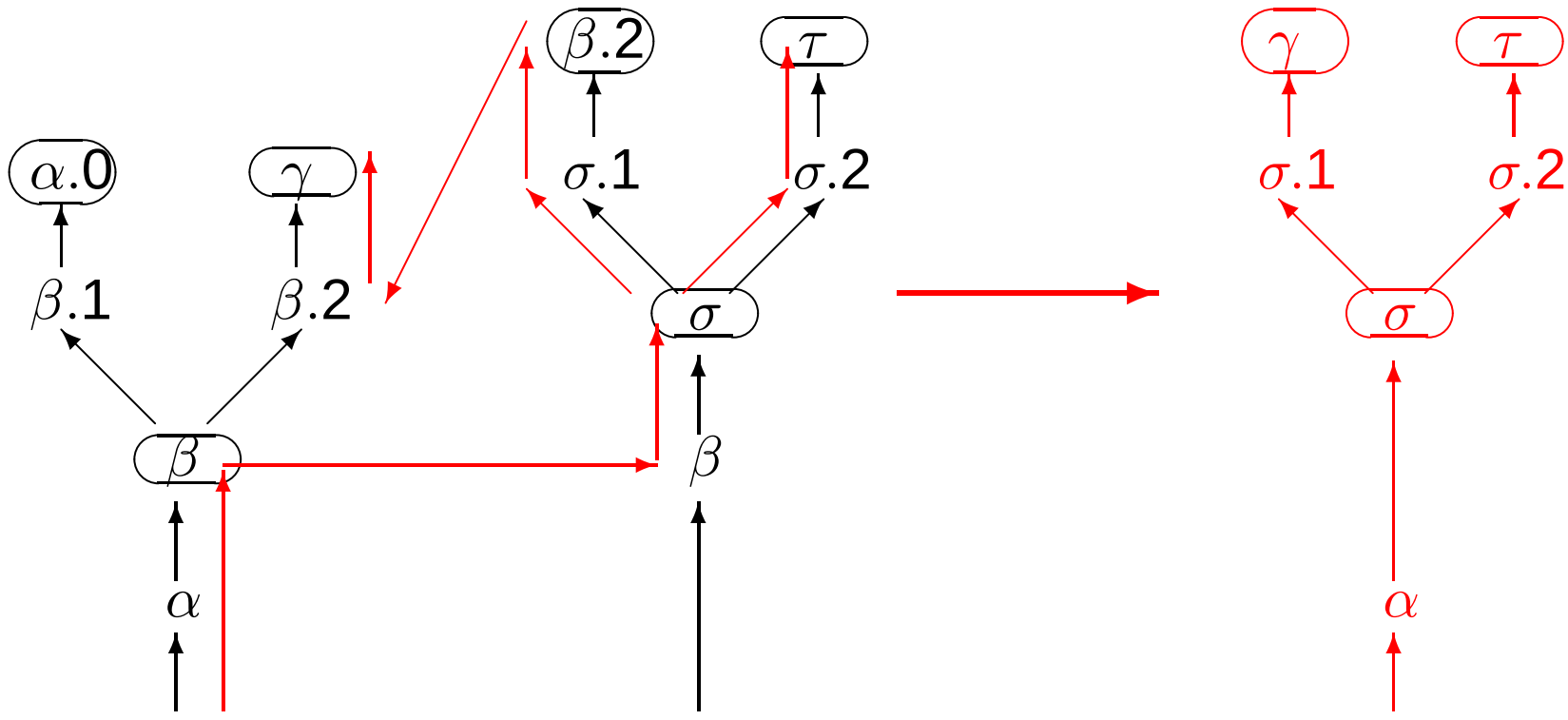
Une **chronique** est une suite d'actions

- alternée
- justifiée (action directe sur négatifs)
- à focus distincts

Un **design** sur base $\Delta \vdash \Gamma$ est un ensemble de chroniques vérifiant

- arborescence (clôture préfixe)
- branchement sur négatif
- propagation (négatifs dictincts sauf cas additif)
- positivité (extension positive)
- totalité (non vide si Δ est vide)

VI. Structures en logique linéaire : ludique (normalisation)



VI. Structures en logique linéaire : ludique (propriétés)

La dualité est exprimée par la finitude de la normalisation entre designs.
La dénotation d'une formule est un **comportement**, i.e. un ensemble de designs égal à son bidual.
Construction de comportements :

$A \oplus B = A \cup B$

$A \otimes B = \{d_1 \otimes d_2 / d_1 \in A, d_2 \in B\}$, par concaténation des 1ères actions positives

Théorèmes :

- séparation : topologie T_0 avec $D < D'$ ssi $D' \in D^{\perp\perp}$
- associativité de la normalisation
- stabilité : la normalisation commute aux intersections
- modèle d'un fragment de la logique linéaire