Université Paris Diderot - Paris 7

Stage de L3 Informatique

Etude du langage PCF à travers les réseaux de preuve de la logique linéaire

Sambo Boris Eng

supervisé par
Delia Kesner
Michele Pagani

Juin 2017 – Juillet 2017

Contents

Introduction

La correspondance preuve-programme, découverte et grandement explorée durant la fin du XX$^{\text{ème}}$ siècle énonce une association formelle entre les preuves mathématiques et les programmes informatiques. De cette correspondance a émergé de très nombreux résultats à la croisée de la logique et de l'informatique.

L'aspect procédural de la logique. Les preuves mathématiques, définies formellement, comme cela a été fait par *Gerhard Gentzen* au début du XX$^{\text{ème}}$ siècle, peuvent contenir des "raccourcis" correspondant à l'utilisation de lemmes, refuges de la créativité. Gentzen lui même a constaté que ces raccourcis que l'on appelle *coupures* pouvaient être éliminés pour donner une preuve dîte "sans coupure" qui est un enchaînement mécanique sans profondeur de règles logiques, une explicitation de la preuve.

Deux points essentiels sont que : (1) la procédure peut faire accroître la preuve de façon exponentielle et atteindre des dimensions parfois absurdes, (2) la correspondance preuve-programme met en relation cette procédure avec l'exécution des programmes et les preuves sans coupures avec les résultats des programmes. C'est en particulier le second point qui nous intéressera, le premier occupant plutôt l'étude de la complexité et la recherche de preuves.

Logique Linéaire. La logique linéaire, introduite par *Jean-Yves Girard* [14], propose une vision plus fine que les logiques précédentes : on y contrôle explicitement l'effacement et la duplication de certaines formules afin de les déconnecter de leur potentiel infini. Cette considération est loin d'être anodine puisqu'elle explicite les connecteurs logiques : on se retrouve avec deux conjonctions $(\otimes, \&)$ et deux disjonctions $(⅋, \oplus)$. Cette logique est particulièrement expressive puisqu'elle peut encoder la logique classique et intuitionniste.

Réseaux de preuve. Conjointement, la logique linéaire est accompagnée d'un nouveau formalisme de preuve appelé *réseau de preuve* qui donne une présentation canonique et "parallèle" du calcul des séquent (qui est "séquentiel") et permet une étonnante étude de la procédure d'élimination des coupures sous un paradigme *d'interaction* entre formules. Les réseaux ont été si expressifs qu'ils ont pu être utilisés en s'abstrayant de toute interprétation logique pour se focaliser seulement sur leur potentiel calculatoire. Ainsi on a pu obtenir des modèles intéressants de parallélisme et encoder différentes variantes du λ-calcul pour mieux les étudier : deux λ-termes syntaxiquement différents mais ayant le même comportement opérationnel peuvent être représentés par un même réseau. La pertinence des réseaux pour l'étude du λ-calcul a été initialement étudiée par *Vincent Danos* [6] et *Laurent Regnier* [27].

Substitutions explicites. Initialement motivé par l'implémentation des langages de programmation fonctionnelle, les informaticiens ont proposé un formalisme où la substitution n'est plus une procédure externe de réécriture mais partie intégrante de la syntaxe d'un langage [1]. L'opération banale de β-réduction des λ-termes cache, en effet, de nombreuses subtilités (comme la duplication de variables), chose que les substitutions explicites révèlent. Il se trouve que la logique linéaire les réseaux de preuve sont particulièrement adéquats à l'étude des substitutions explicites [10, 9].

Dans ce document, nous étudions le langage PCF à travers une traduction dans les réseaux de preuve de la logique linéaire intuitionniste et une preuve de simulation des réductions de PCF. La traduction utilise le formalisme des réseaux d'interaction de *Yves Lafont* [19] et les boîtes additives [5, 15].

Nous terminons sur une extension de PCF aux substitutions explicites par le *Linear Substitution Calculus* abrégé en λ_{lsub} [2] qui généralise le caclul de *Robin Milner* [23].

1 Langage PCF

Le langage PCF, initialement introduit par *G.D Plotkin* [26, 30] est une extension du λ-calcul simplement typé où l'on dispose d'expressions booléennes, d'un traitement minimal des entiers naturels et de la récursion (par le biais d'un combinateur de point fixe : le combinateur Y). C'est un λ-calcul minimal pouvant faire émerger la Turing-complétude.

1.1 Types

Les types de PCF sont donnés par la grammaire suivante :

$$A, B ::= \texttt{bool} \mid \texttt{nat} \mid A \to B$$

1.2 Syntaxe

Les **variables** sont w, x, y, z avec éventuellement des indices (w_i, x_i, y_i, z_i pour $i \in \mathbb{N}$) et les **termes** (possiblement indicés aussi) sont donnés par la grammaire suivante :

$$s, t, u, s ::= x \mid \lambda x^A.t \mid t\ u \mid \texttt{zero} \mid \texttt{succ}\ t \mid \texttt{true} \mid \texttt{false} \mid \texttt{Y}_A \mid$$
$$\texttt{pred}\ t \mid \texttt{iszero}\ t \mid \texttt{if}\ t\ \texttt{then}\ u_1\ \texttt{else}\ u_2$$

1.3 Règles de typage

Les termes sont typés par un contexte Γ qui associe à chaque variable libre un type de PCF dans la syntaxe des séquents : $\Gamma, x : A \vdash t : A$. Premièrement, les règles qui permettent d'inférer le type des termes est donné par les règles usuelles du λ-calcul simplement typé :

$$\frac{}{\Gamma, x : A \vdash x : A}\ (variable) \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x^A.t : A \to B}\ (abstraction)$$

$$\frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash u : A}{\Gamma \vdash (t\ u) : B}\ (application)$$

qu'on étend ensuite aux termes de PCF :

$$\frac{}{\Gamma \vdash \texttt{true} : \texttt{bool}}\ (true) \qquad \frac{}{\Gamma \vdash \texttt{false} : \texttt{bool}}\ (true)$$

$$\frac{}{\Gamma \vdash \texttt{zero} : \texttt{nat}}\ (zero) \qquad \frac{\Gamma \vdash t : \texttt{nat}}{\Gamma \vdash \texttt{succ}\ t : \texttt{nat}}\ (succ)$$

$$\frac{}{\Gamma \vdash \mathtt{Y}_A : (A \to A) \to A}\ (recursion) \qquad \frac{\Gamma \vdash t : \mathtt{nat}}{\Gamma \vdash \mathtt{pred}\ t : \mathtt{nat}}\ (pred)$$

$$\frac{\Gamma \vdash t : \mathtt{nat}}{\Gamma \vdash \mathtt{iszero}\ t : \mathtt{bool}}\ (iszero)$$

$$\frac{\Gamma \vdash t : \mathtt{bool} \quad \Gamma \vdash u_1 : A \quad \Gamma \vdash u_2 : A}{\Gamma \vdash \mathtt{if}\ t\ \mathtt{then}\ u_1\ \mathtt{else}\ u_2 : A}\ (condition)$$

On a fait le choix arbitraire d'utiliser des présentations additives mais le choix de présentations multiplicatives, pour les expressions conditionnelles par exemple, aurait pu être adapté dans le cas d'une correspondance avec une logique de pertinence (*relevance logic*).

Anecdotiquement, on rappelle que l'introduction de la récursion introduit l'inconsistance logique du système de type : le combinateur `Y` étant paramétré, son type énonce $\vdash \forall A.(A \to A) \to A$ c'est à dire que toute formule est prouvable. Cette inconsistance est nécessaire pour avoir la Turing-complétude.

1.4 Evaluation

Les règles d'évaluation sont décrites dans une sémantique opérationnelle small-step avec une stratégie d'évaluation quelconque (full β-reduction [25]) :

Règles de réduction de rédex

$$\frac{}{\mathtt{pred\ zero} \rightsquigarrow \mathtt{zero}}\ (pred_0) \qquad \frac{}{\mathtt{pred}\ (\mathtt{succ}\ t) \rightsquigarrow t}\ (pred_S)$$

$$\frac{}{\mathtt{if\ true\ then}\ t_1\ \mathtt{else}\ t_2 \rightsquigarrow t_1}\ (if_{true}) \qquad \frac{}{\mathtt{if\ false\ then}\ t_1\ \mathtt{else}\ t_2 \rightsquigarrow t_2}\ (if_{false})$$

$$\frac{}{\mathtt{iszero\ zero} \rightsquigarrow \mathtt{true}}\ (iszero_0) \qquad \frac{}{\mathtt{iszero}\ (\mathtt{succ}\ t) \rightsquigarrow \mathtt{false}}\ (iszero_S)$$

$$\frac{}{(\lambda x^A.t)\ u \rightsquigarrow t[x := u]}\ (\beta) \qquad \frac{}{(\mathtt{Y}_A\ t) \rightsquigarrow t\ (\mathtt{Y}_A\ t)}\ (\mathtt{Y})$$

Règles de réduction contextuelle

$$\frac{t \rightsquigarrow u}{\mathtt{succ}\ t \rightsquigarrow \mathtt{succ}\ u}\ (succ) \qquad \frac{t \rightsquigarrow u}{\lambda x^A.t \rightsquigarrow \lambda x^A.u}\ (abs)$$

$$\frac{t_1 \rightsquigarrow t_2}{t_1\ u \rightsquigarrow t_2\ u}\ (app_L) \qquad \frac{u_1 \rightsquigarrow u_2}{t\ u_1 \rightsquigarrow t\ u_2}\ (app_R)$$

$$\frac{t \rightsquigarrow u}{(\mathtt{Y}_A\ t) \rightsquigarrow (\mathtt{Y}_A\ u)}\ (rec) \qquad \frac{t \rightsquigarrow u}{\mathtt{pred}\ t \rightsquigarrow \mathtt{pred}\ u}\ (pred)$$

$$\frac{t \rightsquigarrow u}{\mathtt{iszero}\ t \rightsquigarrow \mathtt{iszero}\ u}\ (iszero)$$

$$\frac{t \rightsquigarrow u}{\mathtt{if}\ t\ \mathtt{then}\ t_1\ \mathtt{else}\ t_2 \rightsquigarrow \mathtt{if}\ u\ \mathtt{then}\ t_1\ \mathtt{else}\ t_2}\ (if_C)$$

$$\frac{t_1 \rightsquigarrow u_1}{\mathtt{if}\ t\ \mathtt{then}\ t_1\ \mathtt{else}\ t_2 \rightsquigarrow \mathtt{if}\ t\ \mathtt{then}\ u_1\ \mathtt{else}\ t_2}\ (if_1)$$

$$\frac{t_2 \rightsquigarrow u_2}{\mathtt{if}\ t\ \mathtt{then}\ t_1\ \mathtt{else}\ t_2 \rightsquigarrow \mathtt{if}\ t\ \mathtt{then}\ t_1\ \mathtt{else}\ u_2}\ (if_1)$$

1.5 Substitution

La *substitution* (implicite) de termes est donnée par une procédure de réécriture externe à PCF. Les *constantes* sont $\texttt{zero}, \texttt{true}, \texttt{false}$ et $\texttt{Y}_A$.

$$
\begin{array}{lcll}
x[x := u] & = & \text{u} & \\
y[x := u] & = & \text{y} & \text{si } y \neq x \text{ ou } y \text{ est une constante} \\
(\lambda y^A.t)[x := u] & = & \lambda y^A.t[x := u] & \text{si } y \neq x \\
(t_1\ t_2)[x := u] & = & (t_1[x := u]\ t_2[x := u]) & \\
(\texttt{succ}\ t)[x := u] & = & \texttt{succ}\ t[x := u] & \\
(\texttt{pred}\ t)[x := u] & = & \texttt{pred}\ t[x := u] & \\
(\texttt{iszero}\ t)[x := u] & = & \texttt{iszero}\ t[x := u] & \\
(\texttt{if}\ t\ \texttt{then}\ u_1\ \texttt{else}\ u_2)[x := u] & = & \texttt{if}\ t[x := u]\ \texttt{then}\ u_1[x := u]\ \texttt{else}\ u_2[x := u] &
\end{array}
$$

2 Réseaux de preuve

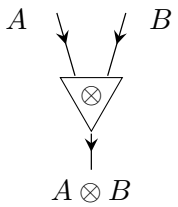
Nous utilisons le formalisme des réseaux d'interaction de *Lafont* [17] pour encoder les réseaux de preuve de *Girard* [19]. Les réseaux d'interaction sont nés du potentiel calculatoire des réseaux de preuve qu'ils généralisent en oubliant toute interprétation logique. L'essence de ce nouveau modèle de calcul réside dans le paradigme de l'*interaction* [18], une nouvelle manière d'interpréter le calcul.

Pour des présentations plus exhaustives on peut se référer à [13] ou [11, 12, 22] pour plus de formalité. Dans notre cas on se contentera d'une observation informelle accompagné de quelques comparaisons avec les réseaux de preuves habituels.

2.1 Cellules et ports

Les réseaux de preuve sont des graphes (éventuellement non connectés, c'est à dire avec des arrêtes libres) contenant des noeuds pour représenter les connecteurs logiques et des arrêtes orientées étiquetées pour les formules.

Les noeuds des réseaux de preuve sont représentés par des triangles appelés **cellules**.

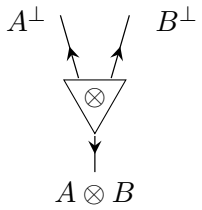


Les arrêtes sont appelées **fils** ou **câbles** et relient des points de contacts appelés **ports** qui sont attachés aux cellules.

- La pointe du triangle est le **port principal** de la cellule représentant la conclusion d'une règle logique.
- Les ports opposés au port principal sont les **ports auxiliaires** représentant les prémisses d'une règle.
- Dans le cas où un port n'est pas connecté à un autre on dit que c'est un **port libre** (ci-dessus on a 3 ports libres).

2.2 Typage des réseaux

On donne un type A aux fils orientés tel que la direction opposée est typée par le type dual $A^\perp$. On peut donc voir chaque fil dans les deux directions avec des typages duaux. Le réseau suivant est équivalent au précédent :

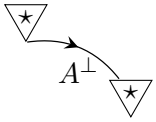


Ce phénomène est analogue au passage gauche-droite de la formulation bilatère à la formulation monolatère du calcul des séquents linéaire.

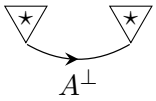
2.3 Axiomes et coupures implicites

L'utilisation de réseaux d'interaction induit une modification de la nature des règles d'axiome et de coupure. Originellement représenté par des noeuds, les règles identité prennent la forme de fils :

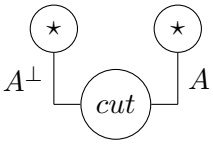
- Un fil de type A entre un port principal et un port auxiliaire est un lien habituel de type A dans les réseaux de preuve



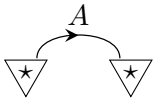
- Un fil de type A entre deux ports principaux cache une coupure entre A et $A^\perp$. C'est à dire que :



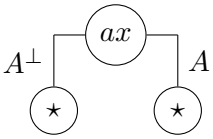
  correspond, dans les réseaux habituels, à :



- De la même manière que pour la coupure, un fil de type A entre deux ports auxiliaires cache un axiome entre A et $A^\perp$:



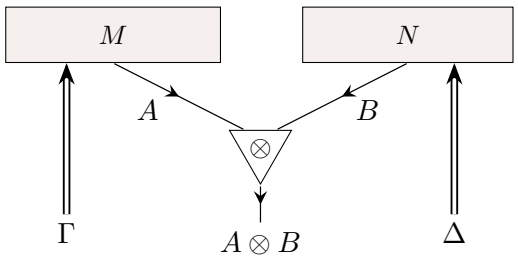
  correspondant à :



2.4 Réseaux multiplicatifs

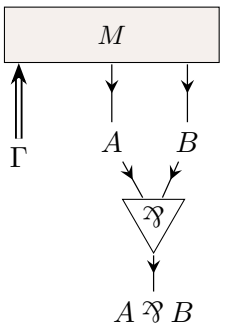
Chaque construction représente une règle du calcul des séquents de la logique linéaire. On dit qu'un réseau possède l'**interface** $\vdash A_1, ..., A_n$ quand il a n ports libres de types $A_1, ..., A_n$. L'interface représente en fait la conclusion d'une règle logique.

Tenseur ($\otimes$)



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

Par ($\parr$)

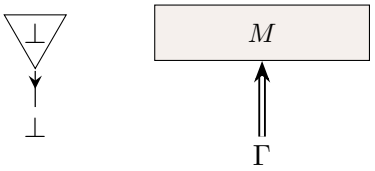


$$\frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B} \ (\parr)$$

Un (1)

$$\frac{}{\vdash 1} \ (1)$$

Bottom ($\bot$)



$$\frac{\vdash \Gamma}{\vdash \Gamma, \bot} \ (\bot)$$

2.5 Réseaux exponentiels

Dans le contexte des réseaux d'interaction, les modalités exponentielles introduisent le concept de **port actif**. Un port actif est un port qui peut-être sujet à une procédure d'élimination des coupures. Pour les connecteurs multiplicatifs par exemple, le port principal est le port actif.

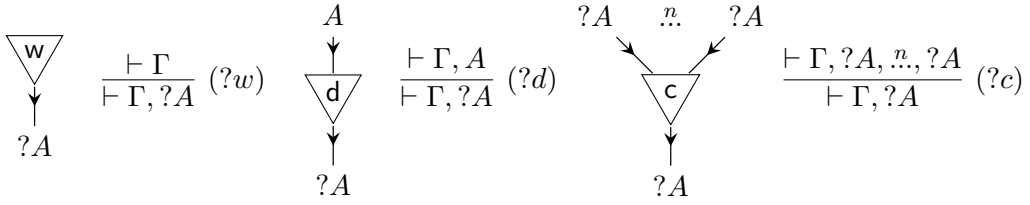
Dans le cas des **boîtes exponentielles** ou **boîte de promotion** symbolisant la règle de promotion, les ports auxiliaires sont *aussi* des ports actifs.

Il existe deux approches équivalentes pour traiter les exponentielles :

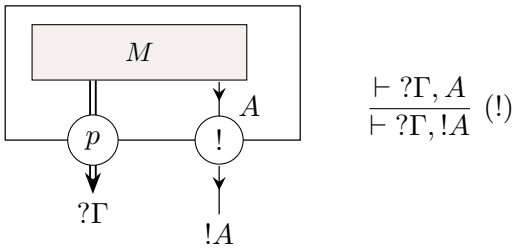
- Généralisation des cellules "pourquoi pas" de façon similaire aux noeuds "pourquoi pas" de *Danos-Regnier* [27, 20] (Réseaux de *Danos-Regnier*).
- Constructions explicites c, d, w comme dans [7] où la contraction est binaire uniquement. Cette approche tient compte de la communativité, l'associativité et la neutralité des liaisons exponentielles et offre une vision plus précise et affinée des réseaux (Réseaux de *Girard*).

Nous faisons le choix d'utiliser les réseaux de *Girard* avec contractions n-aires et des opérations de contrôle des contractions. On garde un contrôle fin tout en s'abstrayant des détails non essentiels pour travailler modulo associativité et commutativité de la contraction [9].

Affaiblissement, Déréliction et contraction n-aire ($?$/"pourquoi pas")



Boîte de promotion (!/"bien sûr")

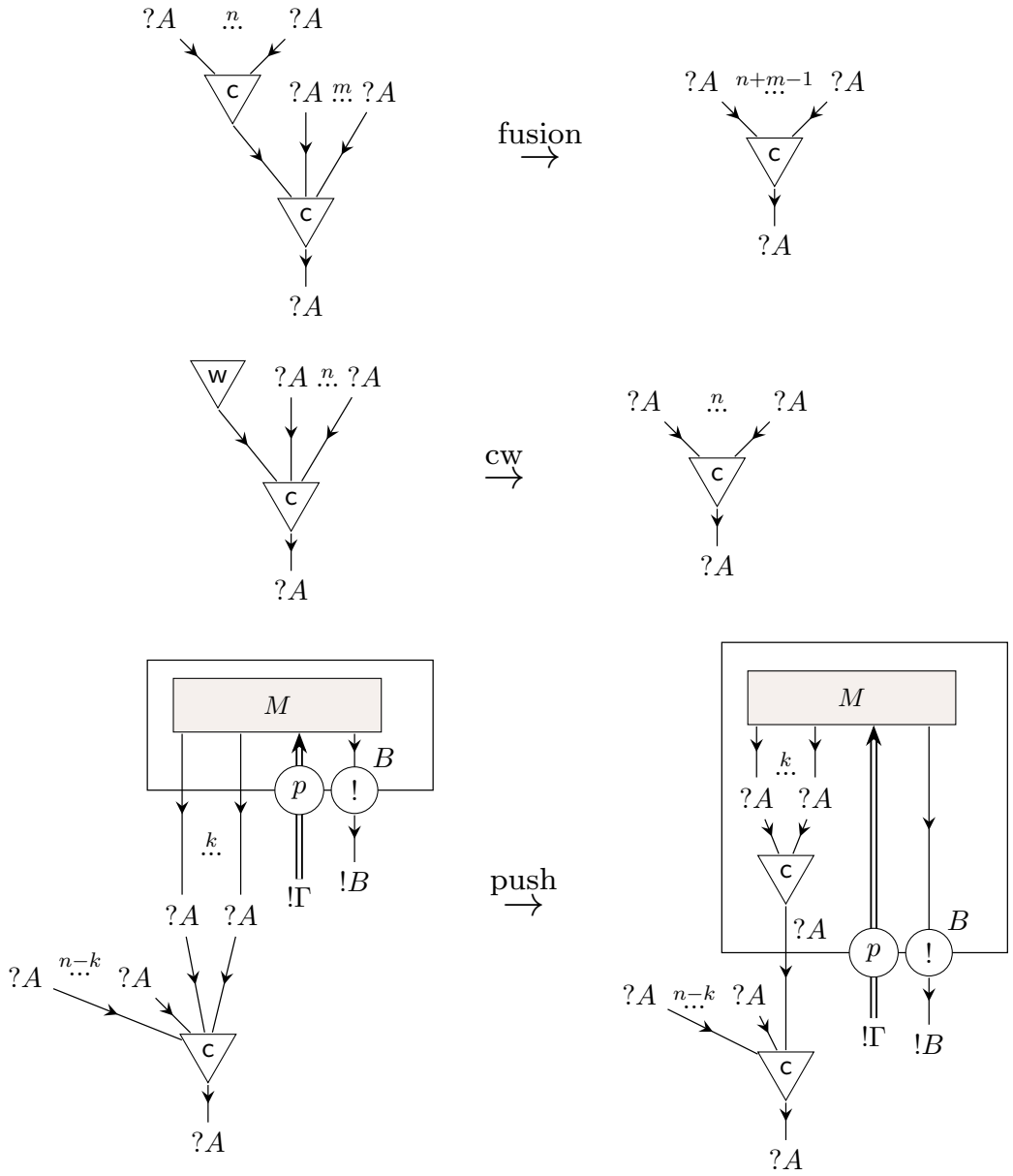


On utilisera un fil double pour représenter tous les fils $\gamma \in \Gamma$ à la fois. On se permet aussi de passer aux noeuds de contraction n-aire des fils doubles pour représenter le partage des variables de plusieurs contextes Γ en prémisse.

Le noeud labellé avec le symbole p représente un port auxiliaire de la boîte. Pour tout noeud p, son fil prémisse et conclusion ont le même type. Appliqué à un contexte Γ de taille n, il représente en fait n ports auxiliaires.

Si $\Gamma = \{\gamma_1, ..., \gamma_n\}$, on utilisera souvent les notations $\Gamma^\perp = \{\gamma_1^\perp, ..., \gamma_n^\perp\}$, $!\Gamma = \{!\gamma_1, ..., !\gamma_n\}$ et $?\Gamma = \{?\gamma_1, ..., ?\gamma_n\}$.

Opérations de contrôle de la contraction

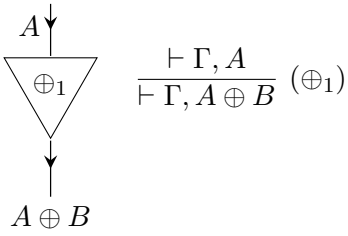


Soit $\overset{?}{\to} = \overset{\text{push}}{\to} \cup \overset{\text{fusion}}{\to} \cup \overset{\text{cw}}{\to}$ la relation de réduction de réseau induite par les trois réductions ci-dessus et $\overset{?}{\twoheadrightarrow}$ sa clôture réflexive transitive. On dit que M' est la ?-forme normale de M (aussi noté $\downarrow_? M$) si et seulement si $M \overset{?}{\twoheadrightarrow} M'$ et M' ne se réduit pas.

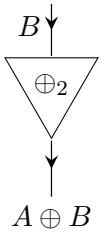
2.6 Réseaux additifs

Nous utilisons les boîtes additives pour représenter le fragment additif de la logique linéaire dans les réseaux de preuve mais d'autres choix peuvent être considérés (tranches additives et pondérations introduites par *Girard* [5, 15]).

Plus $(\oplus_1/\oplus_2)$

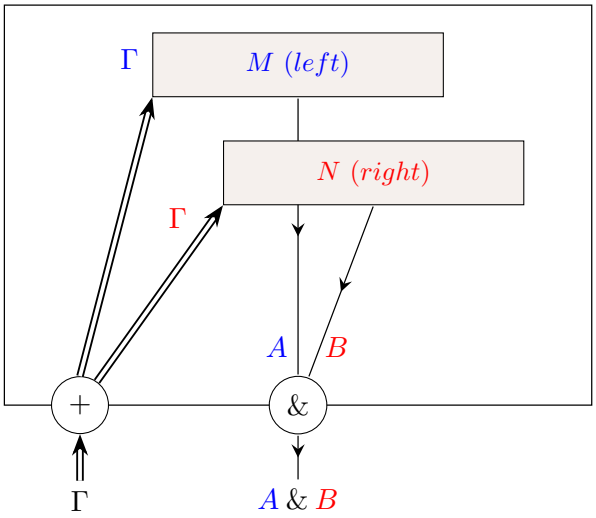


$$\frac{\vdash \Gamma, A}{\vdash \Gamma, A \oplus B}\ (\oplus_1)$$



$$\frac{\vdash \Gamma, B}{\vdash \Gamma, A \oplus B}\ (\oplus_2)$$

Avec (&)



$$\frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \,\&\, B}\ (\&)$$

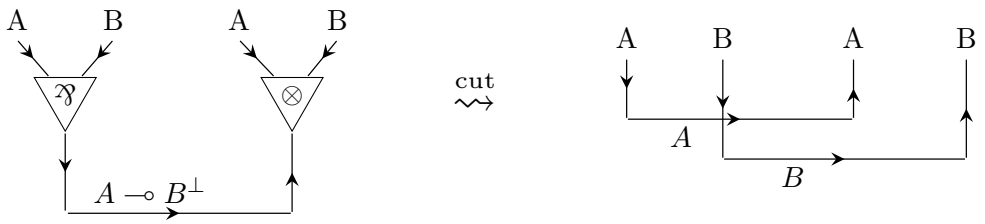
Pour le connecteur & on utilise une boîte additive. On peut apercevoir que l'on a des **contractions additives** représentées par un contracteur + qui permet de simuler la contraction des contextes Γ dans la règle associée.

2.7 Elimination des coupures multiplicatives

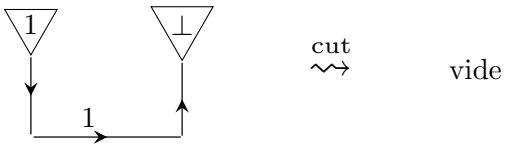
L'élimination des coupures est une procédure de réécriture de graphe qui fait interagir des connecteurs duaux entre eux pour permettre une réduction des réseaux.

On appelle **coupure** une paire de ports actifs reliés par un fil.

Par - Tenseur



Un - Bottom

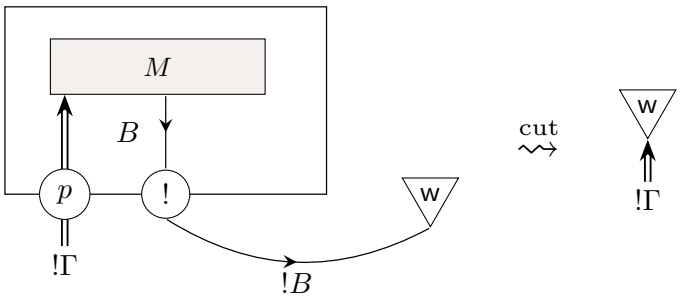


2.8 Elimination des coupures exponentielles

Dans le cas des réseaux exponentiels, la procédure d'élimination des coupures confronte la boîte de promotion aux autres cellules exponentielles.

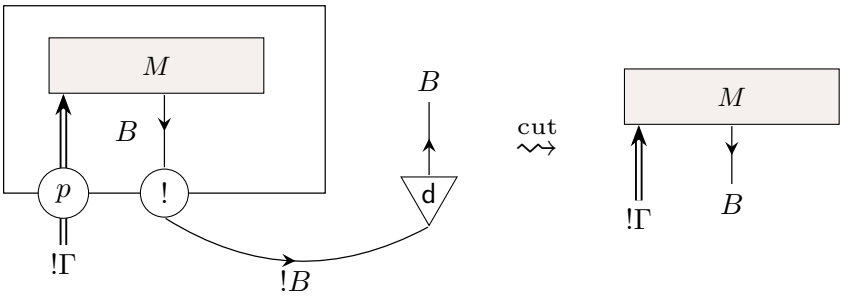
Affaiblissement - Promotion

L'affaiblissement fait disparaître la boîte et laisse des résidus correspondant au contexte Γ.



Déréliction - Promotion

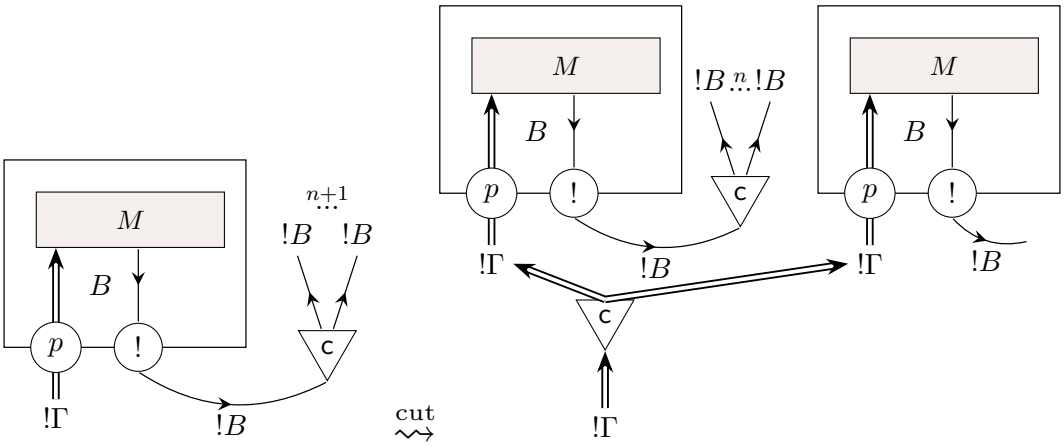
La déréliction ouvre la boîte.



Contraction - Promotion

L'élimination d'une coupure contraction n-aire avec une boîte promotion duplique la boîte pour un des n fils pour laisse les $n-1$ fils restants contractés.

Par défaut, on appliquera n élimination de coupure contraction-promotion quand la granularité de la réduction n'est pas nécessaire.



Commutations exponentielles

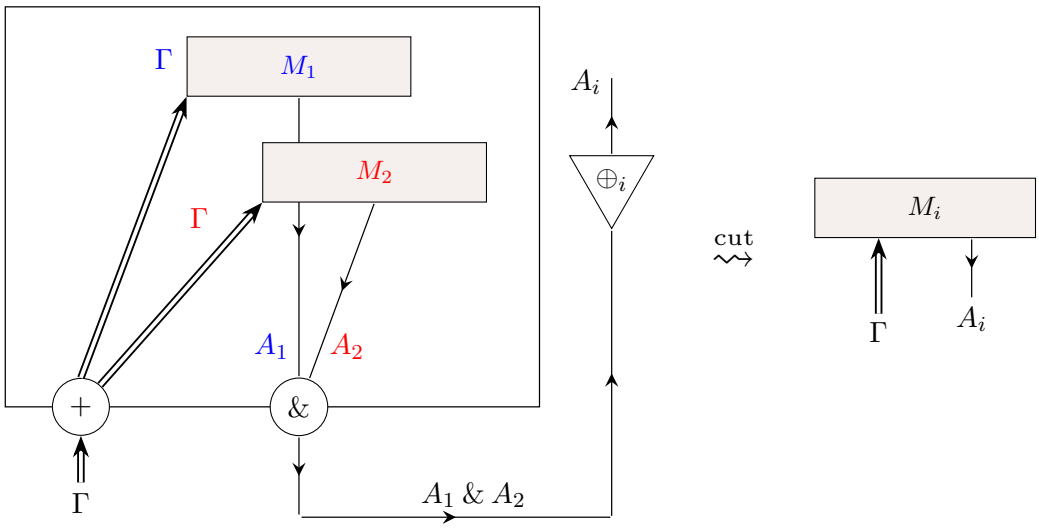
Si une boîte exponentielle est connectée à l'un des ports auxiliaires d'une autre boîte exponentielle, on fait rentrer la première à l'intérieur de la seconde.

2.9 Elimination des coupures additives

Une coupure additive met en relation une boîte additive & et le connecteur $\oplus_1$ ou $\oplus_2$. L'élimination des coupures va choisir une partie de la boîte (gauche ou droite) et effacer l'autre partie. Cette procédure simule un choix.

On distinguera les deux parties de la boîte par des couleurs et/ou une barre de séparation et/ou leur position pour minimiser les ambiguïtés.

Avec - Plus



Commutations additives

Soit N une boîte additive avec les sous-réseaux N_1 (gauche) et N_2 (droite) et P une boîte de promotion.

Quand on a une coupure entre un fil issu d'une contraction additive de N et P, on duplique deux fois P et on fait rentrer chaque copie dans N avec l'une d'elles connectée à N_1 et l'autre à N_2.

3 Traduction de PCF dans les réseaux de preuve

Ci-dessous, on donne une correspondance entre un jugement de la forme $\Gamma \vdash t : A$ et sa pré-traduction $[\Gamma \vdash t : A]$ qui contient éventuellement des coupures multiplicatives avec un fil conclusion $?B^\perp$ pour chaque $(\gamma : B) \in \Gamma$.

La **traduction complète** ou juste **traduction** $[\![\Gamma \vdash t : A]\!]$ est donnée en éliminant toutes les coupures multiplicatives de $\downarrow_? [\Gamma \vdash t : A]$.

L'élimination des coupures multiplicatives dans la traduction permet d'avoir le même réseau pour deux termes σ-équivalents. On rappelle ci-dessous les règles de la σ-réduction $\to_\sigma$ introduites par *Laurent Regnier* [28, 21] :

$$\text{Infiltration :} \qquad ((\lambda u.t)u)v \quad \to_\sigma \quad (\lambda y.(t\ v))u \qquad \text{sachant } y \notin v \tag{1}$$

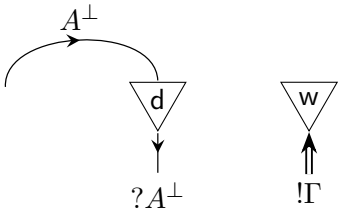
$$\text{Capture :} \qquad (\lambda y.\lambda x.t)u \quad \to_\sigma \quad \lambda x.((\lambda y.t)u) \qquad \text{sachant } x \notin u \tag{2}$$

La σ-équivalence $=_\sigma$ est la relation d'équivalence induite par $\to_\sigma$.

La traduction d'une variante de λ-calcul se base essentiellement sur une traduction de la logique intuitionniste dans la logique linéaire.

3.1 Variable

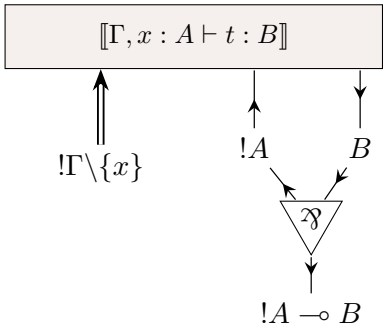
$[\Gamma, x : A \vdash x : A] =$



3.2 Abstraction

La traduction de l'abstraction est fondée sur l'interprétation de la logique intuitionniste dans la logique linéaire proposée par *Girard*. Le type $A \to B$ est interprêté comme $!A \multimap B$.

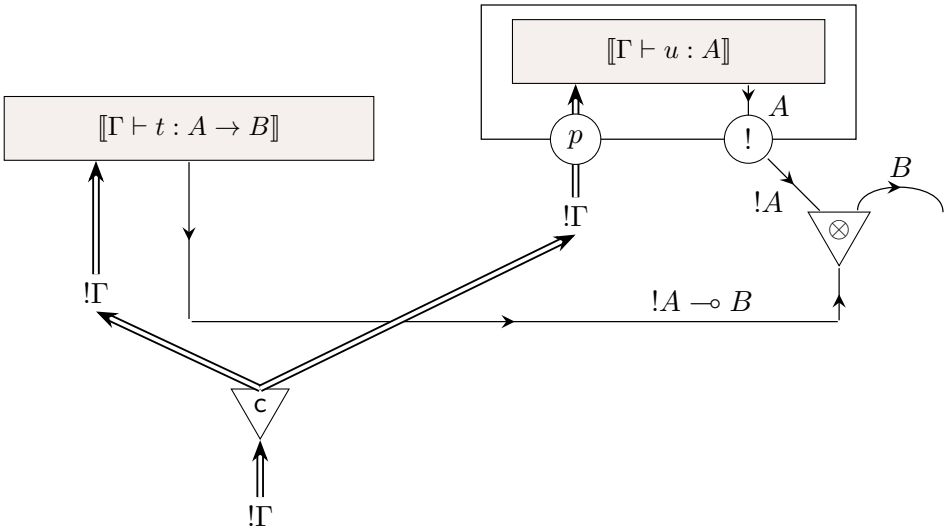
$[\Gamma \vdash \lambda x^A.t : A \to B] =$



La pré-traduction ajoute une cellule d'affaiblissement au bout du lien de type $!A$ si x n'est pas libre dans t. Cela permet de simuler l'absence d'usage d'une variable liée.

3.3 Application

$$[\Gamma \vdash (t\ u) : B] =$$



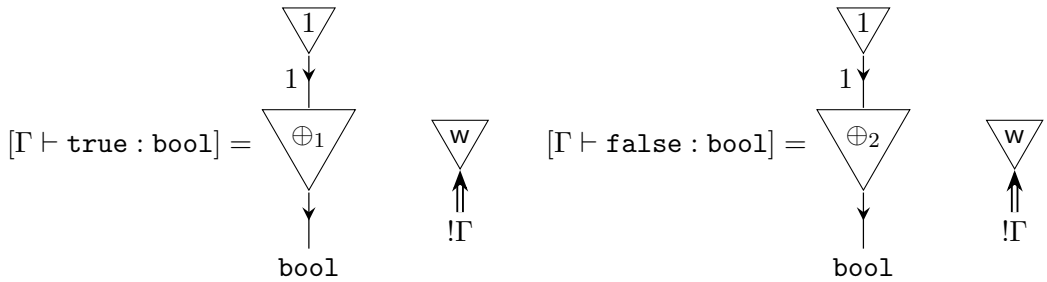
Il faudra éliminer la coupure multiplicative présente afin d'atteindre la traduction $[\![\Gamma \vdash (t\ u) : B]\!]$ que l'on a définit. De plus :

- il faut ajouter des cellules d'affaiblissement pour chaque variable libre de u qui est liée dans t (et vice-versa ; pour chaque variable libre de t qui est liée dans u). Similairement au cas de l'abstraction cela permet de tenir compte du comportement opérationnel des termes selon l'absence ou la présence d'une variable liée. Par exemple $(\lambda x.t)u$ se comporte de deux façons différentes selon si la variable x est libre ou liée dans t.

- chaque occurrence d'une même variable libre doit être prémisse d'une cellule de contraction. On aura donc autant de cellule de contraction que de variables libres différentes.

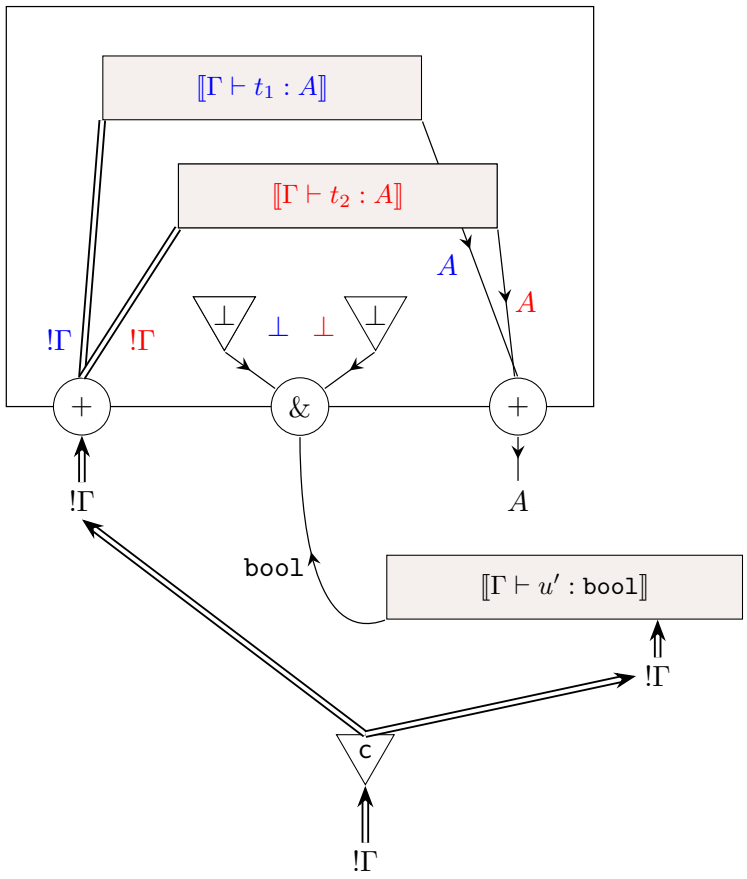
3.4 Booléens

On utilise l'encodage $\texttt{bool} = 1 \oplus 1$ pour les booléens simulant un type somme à deux constructeurs nullaires.



3.5 Conditions

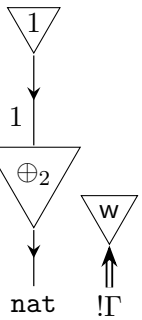
$[\Gamma \vdash \texttt{if}\ u'\ \texttt{then}\ t_1\ \texttt{else}\ t_2 : A] =$



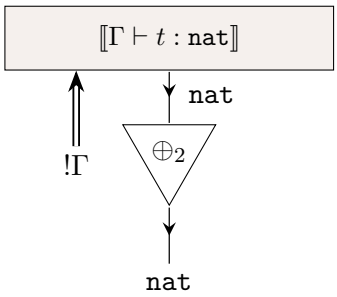
3.6 Entiers naturels

On utilise l'encodage des types inductifs $\texttt{nat} = \mu X.1 \oplus X$.

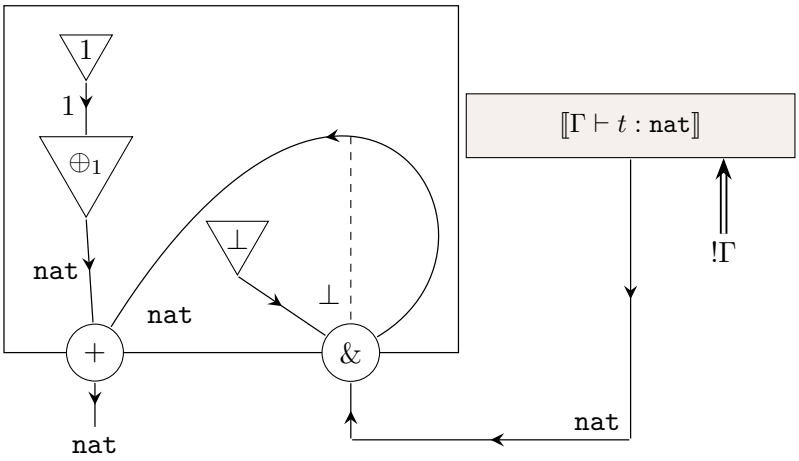
$[\Gamma \vdash \texttt{zero} : \texttt{nat}] =$



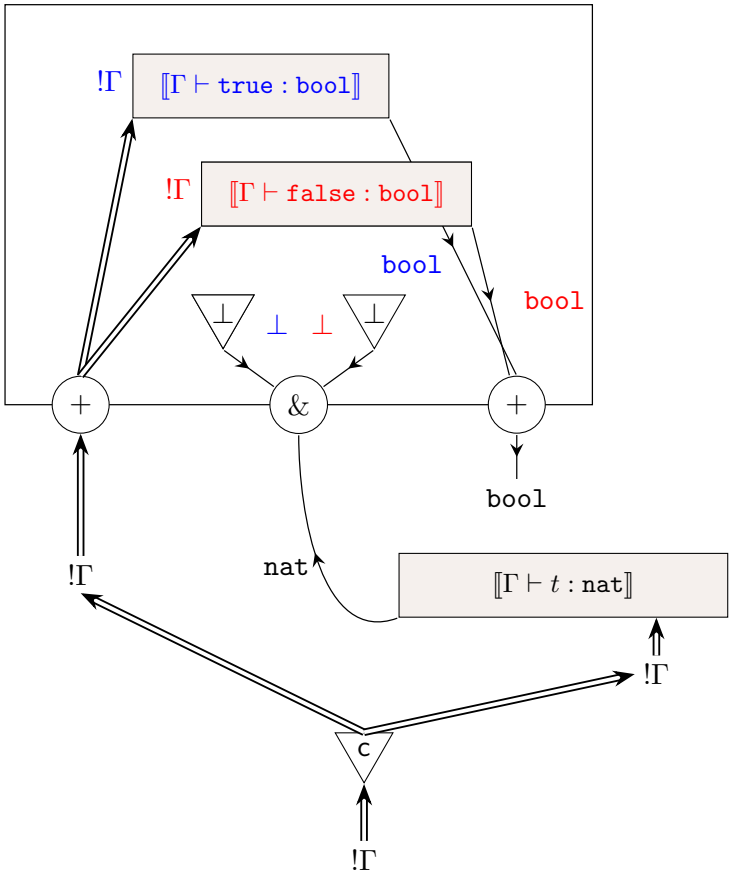
$[\Gamma \vdash \texttt{succ}\ t : \texttt{nat}] =$



$[\Gamma \vdash \texttt{pred}\ t : \texttt{nat}] =$



$[\Gamma \vdash \texttt{iszero}\ t : \texttt{bool}] =$

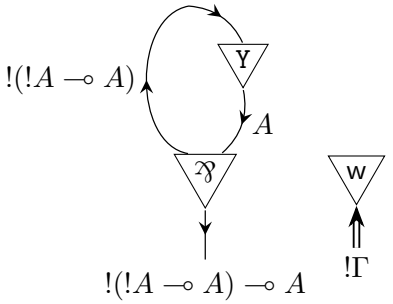


3.7 Récursion

Pour représenter le combinateur `Y` dans les réseaux on utilise la solution la plus naïve et simple : on établit une construction à partir d'une nouvelle cellule dont l'interface correspond à sa règle de typage.

L'interface de la cellule sera représentée par le séquent $\vdash !\Gamma^{\perp}, !(!A \multimap A) \multimap A$ de la logique linéaire. Graphiquement, on représente la nouvelle construction ainsi :

$[\Gamma \vdash \mathtt{Y}_A : (A \to A) \to A] =$

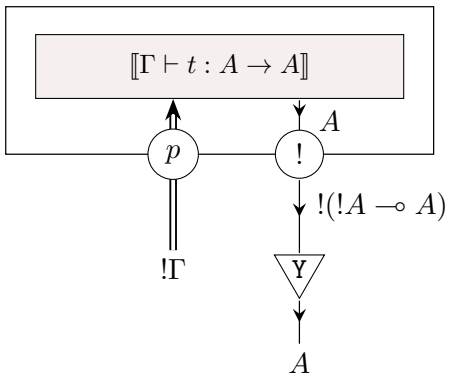


On peut noter qu'il est possible d'expliciter la cellule `Y` par un réseau mais qui introduit un cycle et fait perdre la cohérence logique (non satisfaction du critère de correction). On s'intéressera surtout au contenu calculatoire des réseaux plutôt qu'à leur interprétation logique.

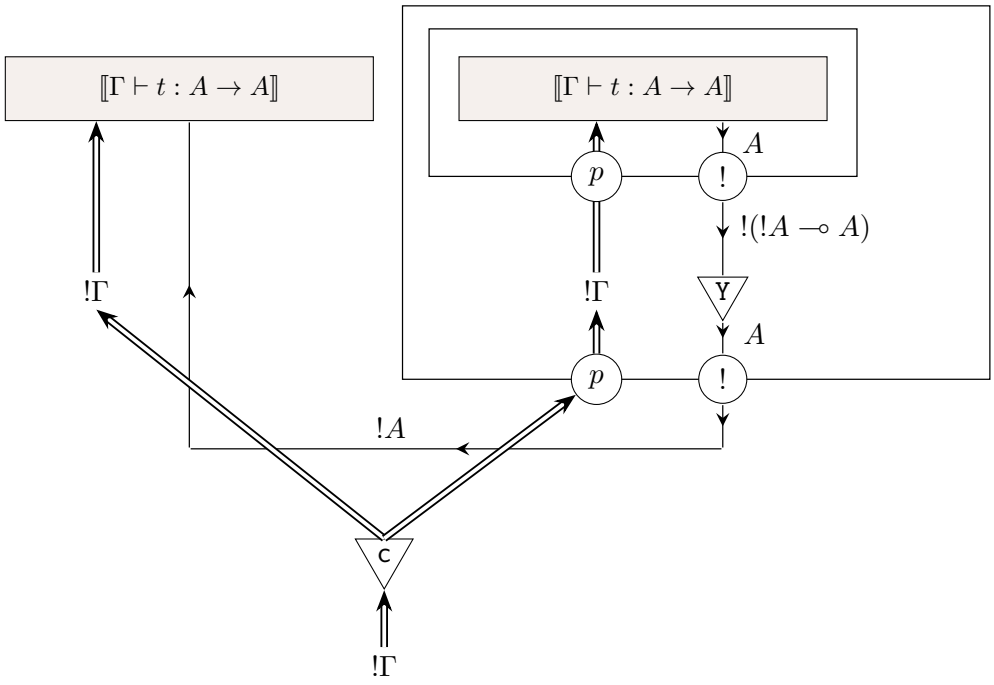
3.8 Réduction des réseaux

Les réseaux sont réduits par les étapes d'élimination des coupures multiplicatives, additives et exponentielles adéquates puis mise en ?-forme normale.

Seule la cellule `Y` profite d'un traitement particulier par la réduction suivante :



$\mathtt{Y}/! \rightsquigarrow$



Cette procédure permet de simuler la règle de réduction $\dfrac{}{\mathtt{Y}\ t \rightsquigarrow t\ (\mathtt{Y}\ t)}\ ^{\mathtt{Y}}$ correspondant à `Y`. En réduisant successivement le réseau on finira par tomber sur un affaiblissement de la boîte exponentielle la plus interne qui va terminer la récursion.

4 Théorème de simulation

Le théorème de simulation énonce qu'une réduction dans PCF peut être transportée en une réduction de réseaux par la traduction énoncée précédemment.

Théorème (Préservation de type)

Soit t et u deux termes de PCF typés sous un contexte Γ.
Si $\Gamma \vdash t : A$ et $t \rightsquigarrow u$ alors $\Gamma \vdash u : A$.

Comme PCF possède la propriété de *préservation de type* on s'autorisera à utiliser le raccourci $\Gamma \vdash t : A \rightsquigarrow \Gamma \vdash u : A$ pour $\Gamma \vdash t : A$, $\Gamma \vdash u : A$ et $t \rightsquigarrow u$.

Propriété (Simulation de la dynamique d'exécution)

Pour tout terme t de PCF, $\Gamma \vdash t : A$ est un rédex si et seulement si $\llbracket \Gamma \vdash t : A \rrbracket$ contient une coupure.

Démonstration.
On procède par induction sur t.

- ↠ Si t est une variable, un booléen, $\mathtt{Y}_A$ ou `zero` alors t n'est pas un rédex : hypothèse incohérente.

- ↠ Si $t = (\lambda x.u)$ on a forcément un rédex dans u. Par hypothèse d'induction la traduction de u contient une coupure et la traduction d'une abstraction n'introduit pas de coupure en dehors de celles de u donc $(\lambda x.u)$ est un rédex ssi $[\![\Gamma \vdash (\lambda x.u) : A]\!]$ contient une coupure.
- ↠ Si t est `succ` u, `pred` u, `iszero` u, `if` u `then` t_1 `else` t_2 le cas est similaire au précédent.
- ↠ Si $t = (t_1\ t_2)$:
  - Si $t = (\lambda x.u)\ t_2$. On a une coupure exponentielle dans sa traduction.
  - Si $t = (\mathtt{Y}_A\ t)$. On a une coupure exponentielle dans sa traduction.
  - Sinon soit t_1 ou t_2 est un rédex. Sans perte de généralité, supposons que t_1 soit un rédex. Par hypothèse d'induction t_1 est un rédex ssi sa traduction contient une coupure. On a donc $(t_1\ t_2)$ est un rédex ssi $[\![\Gamma \vdash (t_1\ t_2) : A]\!]$ contient une coupure (la coupure est conservée par application).

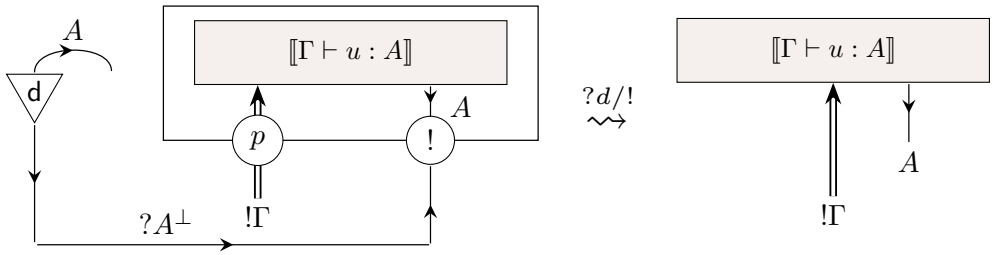
□

Lemme (Simulation de la β-reduction)

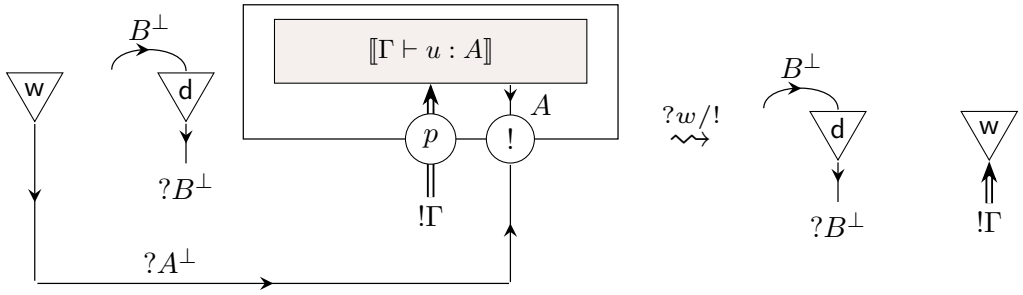
Pour tout terme t de PCF, $[\![\Gamma \vdash (\lambda x^A.t)\ u : B]\!] \overset{\text{cut}}{\rightsquigarrow}{}^* [\![\Gamma \vdash t[x := u] : B]\!]$

Démonstration.
On procède par induction sur t.

▶ Si t est la variable x (c'est à dire qu'on a $B = A$). On élimine la coupure exponentielle déréliction-promotion du rédex pour obtenir le réseau $[\![\Gamma \vdash u : A]\!]$ c'est à dire $[\![\Gamma \vdash x[x := u] : A]\!]$.

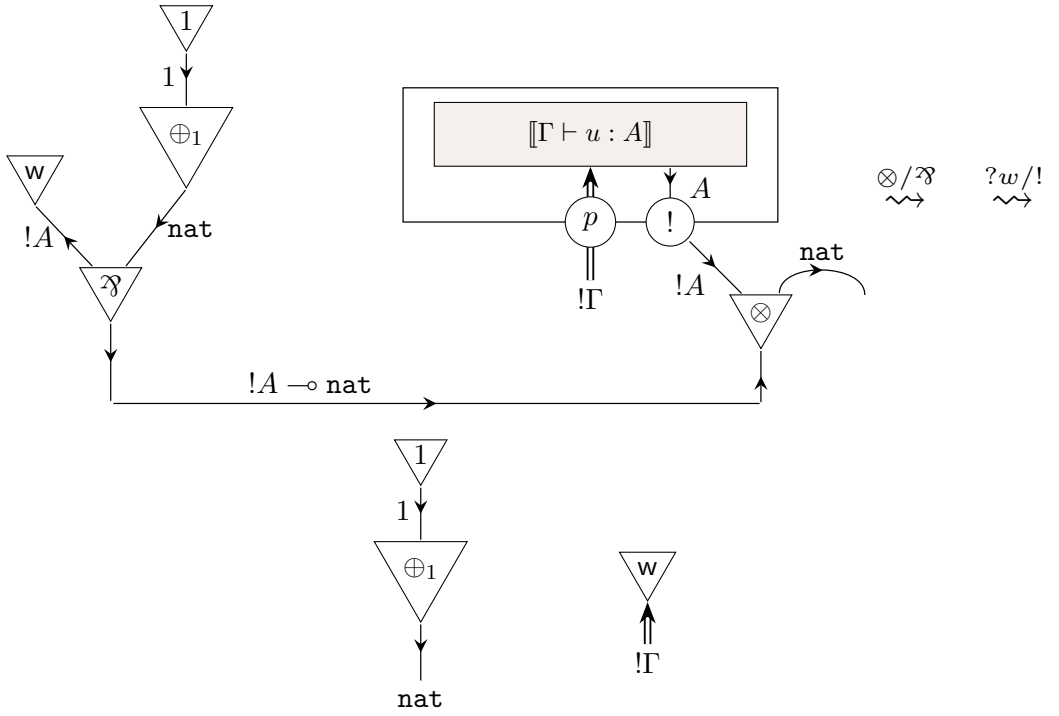


▶ Si t est une variable y tel que $x \neq y$. Considérons le cas où $y \notin \mathtt{fv}(u)$. On élimine la coupure exponentielle affaiblissement-promotion de la traduction du rédex pour obtenir le réseau $[\![\Gamma \vdash y : B]\!]$ accompagné de cellules d'affaiblissement pour chaque variable libre de u.

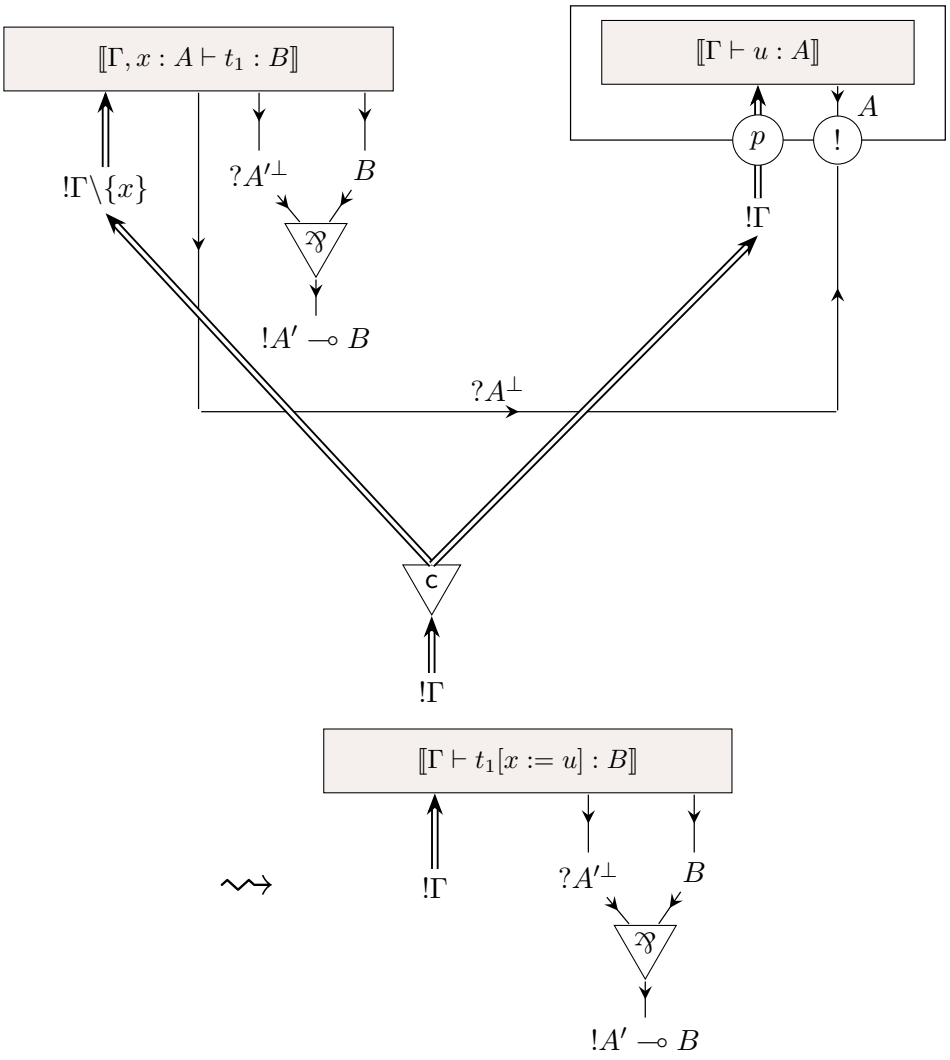


Si $y \in \mathtt{fv}(u)$, on a un lien du contexte $!\Gamma$ de u contracté avec la déréliction représentant la variable y. Cependant, l'élimination de la coupure exponentielle termine sur le même réseau.

▶ Si t est une valeur $\mathtt{zero}, \mathtt{true}$, $\mathtt{false}$ ou la constante $\mathtt{Y}_A$. On a forcément $x \notin \mathtt{fv}(t)$ ce qui détruit la boîte exponentielle. La preuve est similaire au cas précédent. Illustrons en partant, par exemple, de $[\Gamma \vdash \mathtt{zero} : \mathtt{nat}]$:

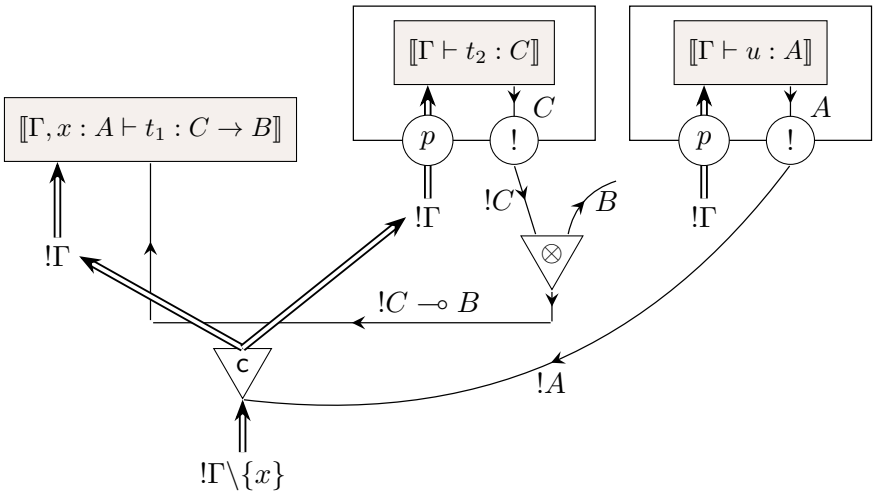


▶ Si $t = \lambda y^{A'}.t_1$. On veut montrer $[\![\Gamma \vdash (\lambda x^A y^{A'}.t_1)\ u : B]\!] \overset{\text{cut}}{\rightsquigarrow}{}^* [\![\Gamma \vdash (\lambda y^{A'}.t_1)[x := u] : B]\!]$. Par hypothèse d'induction $[\![\Gamma \vdash (\lambda x^A.t_1)\ u : B]\!] \overset{\text{cut}}{\rightsquigarrow}{}^* [\![\Gamma \vdash t_1[x := u] : B]\!]$. Cela nous permet de réécrire partiellement le réseau ainsi :



▶ Si $t = (t_1\ t_2)$. On part de $[\Gamma \vdash (\lambda x^A.t_1\ t_2)\ u : B]$.

↠ Si $x \in \mathtt{fv}(t_1)$ et $x \in \mathtt{fv}(t_2)$, on a un lien sortant de t_1 et t_2 connecté à une cellule de contraction représentant la variable x. La cellule est connectée à la boîte de u :



On met le terme en ?-forme normale et on élimine les éventuelles coupures $⅋/\otimes$. En éliminant la coupure exponentielle contraction-promotion on duplique deux fois la boîte de u. Une des copies se connecte à la cellule de contraction dans t_1 représentant la variable x et l'autre à celle de t_2 en rentrant dans sa boîte par commutation exponentielle. Dans cette boîte on retrouve une autre cellule qui contracte les occurrences de x dans t_2 (comme nous sommes en ?-forme normale). On élimine les éventuelles coupures $⅋/\otimes$ introduites.

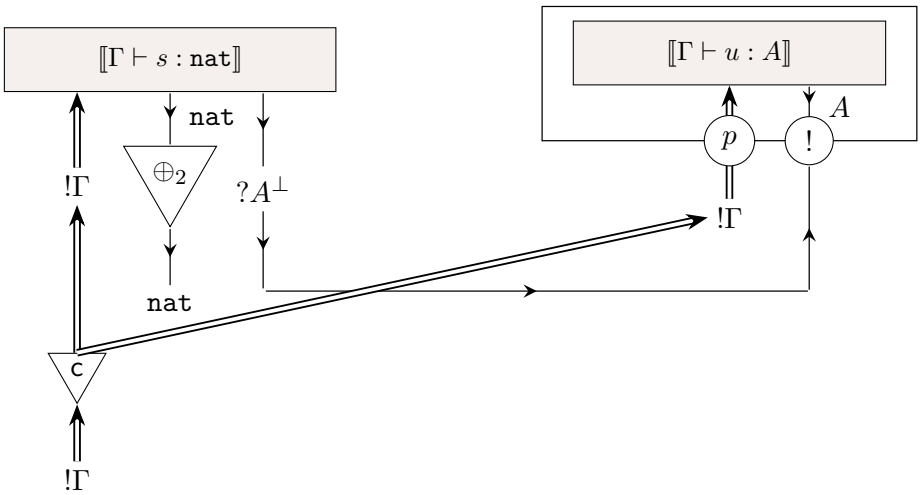
On peut ensuite utiliser les deux hypothèses d'induction pour réécrire respectivement $[\![\Gamma \vdash (\lambda x.t_1)\ u : B]\!]$ en $[\![\Gamma \vdash t_1[x := u] : B]\!]$ et $[\![\Gamma \vdash (\lambda x.t_2)\ u : B]\!]$ en $[\![\Gamma \vdash t_2[x := u] : B]\!]$. On obtient $[\![\Gamma \vdash (t_1[x := u]\ t_2[x := u]) : B]\!]$ c'est à dire $[\![\Gamma \vdash (t_1\ t_2)[x := u] : B]\!]$.

↠ Si $t_1 = \mathtt{Y}$, après ?-normalisation et élimination de coupure $⅋/\otimes$, soit $x \in \mathtt{fv}(t_2)$ et on élimine la coupure déréliction-promotion pour ouvrir la boîte et appliquer l'hypothèse d'induction afin d'obtenir $[\![\Gamma \vdash \mathtt{Y}\ t_2[x := u] : B]\!] = [\![\Gamma \vdash (\mathtt{Y}\ t_2)[x := u] : B]\!]$ ou alors $x \notin \mathtt{fv}(t_2)$ et la boîte disparaît par affaiblissement et on obtient $[\![\Gamma \vdash (\mathtt{Y}\ t_2) : B]\!] = [\![\Gamma \vdash (\mathtt{Y}\ t_2)[x := u] : B]\!]$. On élimine les éventuelles coupures $⅋/\otimes$ introduites.

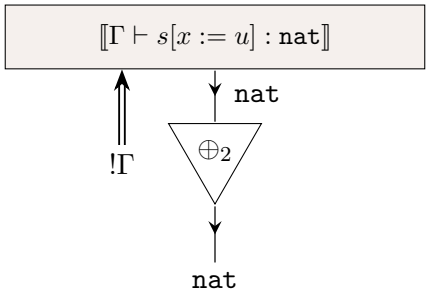
↠ Si $x \notin \mathtt{fv}(t_1)$ ou $x \notin \mathtt{fv}(t_2)$, atteignons une traduction complète. Dans le cas où les deux n'ont pas d'occurrence libre de x, la boîte est affaiblie et on obtient $[\![\Gamma \vdash (t_1\ t_2) : B]\!]$. On élimine les éventuelles coupures $⅋/\otimes$ introduites.

Sinon, seul un des deux réseaux sera connecté à une cellule de déréliction. On élimine la coupure exponentielle déréliction-promotion en ouvrant la boîte de u puis on applique l'hypothèse d'induction adéquate et on élimine les éventuelles coupures $⅋/\otimes$ introduites.

▶ Si $t = \mathtt{succ}\ s$. On a $[\![\Gamma \vdash (\lambda x.\mathtt{succ}\ s)\ u : \mathtt{nat}]\!]$:



On utilise l'hypothèse d'induction pour réécrire le réseau $[\![\Gamma \vdash (\lambda x.s)\ u : \mathtt{nat}]\!]$ en $[\![\Gamma \vdash s[x := u] : \mathtt{nat}]\!]$ pour obtenir $[\![\Gamma \vdash \mathtt{succ}\ (s[x := u]) : \mathtt{nat}]\!]$ c'est à dire $[\![\Gamma \vdash (\mathtt{succ}\ s)[x := u] : \mathtt{nat}]\!]$.



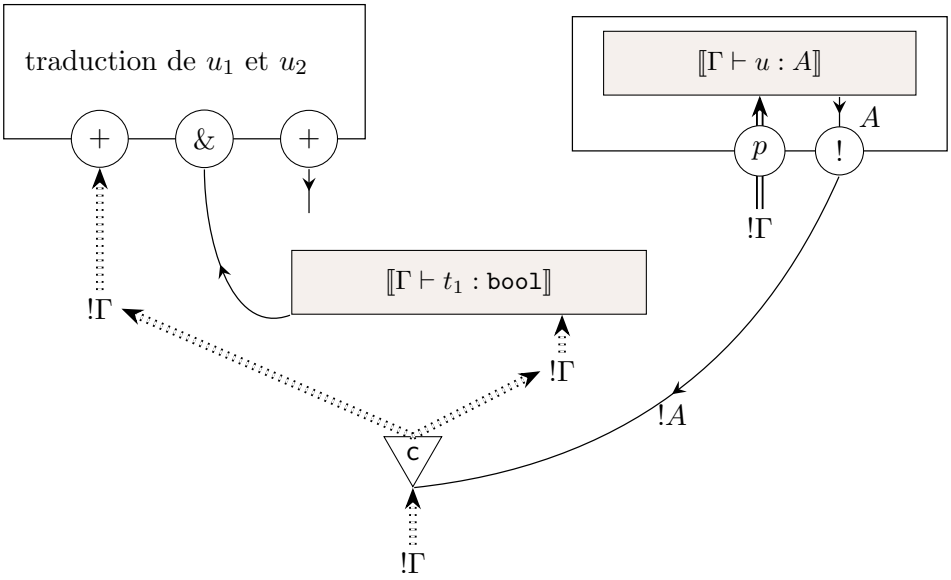
▶ Si $t =$ `pred` s. On procède de la même manière que le cas précédent en appliquant l'hypothèse d'induction.

▶ Si $t =$ `iszero` s. Même procédé.

▶ Si $t =$ `if` t_1 `then` u_1 `else` u_2. La preuve est similaire au cas de l'application. Une particularité de ce cas est qu'il faut prendre en compte que le contenu de la boîte additive (&) correspondant aux expressions conditionnelles peut contenir une occurrence libre de x.

↠ Si x n'est libre dans aucun des sous-termes, la boîte de u est affaiblie et on obtient $⟦Γ ⊢$ `if` t_1 `then` u_1 `else` $u_2 : A⟧ = ⟦Γ ⊢$ (`if` t_1 `then` u_1 `else` $u_2)[x := u] : A⟧$.

↠ Sinon, on a une éventuelle contraction qui partage x entre t_1 et la boîte additive des conclusions u_1 et u_2 :



Au moins un des chemins prémisses de la cellule de contraction doit exister. On élimine la coupure exponentielle entre la contraction et la boîte de u pour la partager.

Pour montrer que $⟦Γ ⊢$ `if` t_1 `then` u_1 `else` $u_2 : A⟧$ se réduit en $⟦Γ ⊢$ (`if` t_1 `then` u_1 `else` $u_2)[x := u] : A⟧$ il faut montrer que peu importe la localisation d'une instance libre de x dans t, elle sera substituée par u :

– Si $x \in \mathtt{fv}(t_1)$, par hypothèse d'induction, on peut réécrire $[\![\Gamma \vdash t_1 : \mathtt{bool}]\!]$ en $[\![\Gamma \vdash t_1[x := u] : \mathtt{bool}]\!]$.
– Si $x \in \mathtt{fv}(u_1)$ ou $x \in \mathtt{fv}(u_2)$. On applique l'élimination des commutations additives pour faire rentrer la boîte de u dans la boîte additive et on applique les hypothèses d'induction nécessaires.

□

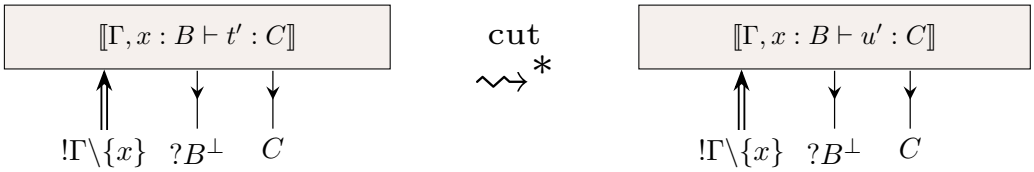
Théorème (Simulation de l'évaluation de PCF)

Soit t et u deux termes de PCF typés sous un contexte Γ.
Si $\Gamma \vdash t : A \rightsquigarrow \Gamma \vdash u : A$ alors $[\![\Gamma \vdash t : A]\!] \overset{\text{cut}}{\rightsquigarrow}{}^* [\![\Gamma \vdash u : A]\!]$.

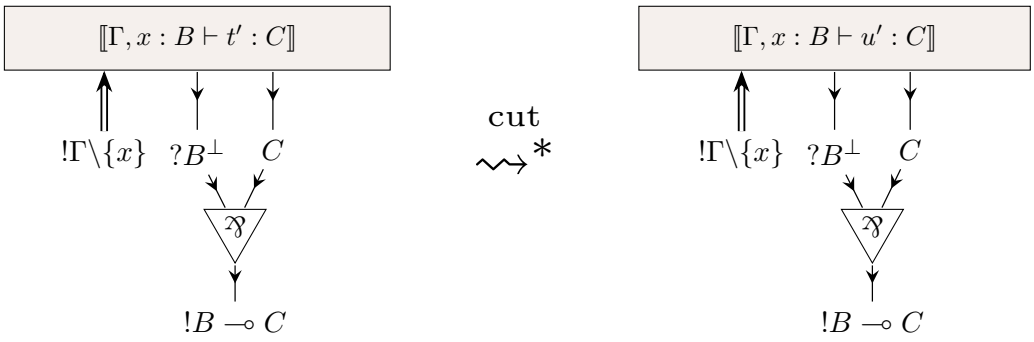
Démonstration.
On procède par induction sur le terme t sur une stratégie de réduction quelconque (full β-reduction). L'hypothèse d'induction nous dira que si un terme se réduit en un autre, alors cette réduction est conservée dans leur traduction.

▶ Si t est une variable, `zero`, `true`, `false` ou `Y`. Trivial par incohérence de l'hypothèse.

▶ Si $t = \lambda x^B.t'$ et $A = B \to C$. On a forcément $u = \lambda x^B.u'$ et $\Gamma, x : B \vdash t' : C \rightsquigarrow \Gamma, x : B \vdash u' : C$ (c'est le seul cas de réduction possible pour une abstraction). Par hypothèse d'induction on obtient la réduction $[\![\Gamma, x : B \vdash t' : C]\!] \overset{\text{cut}}{\rightsquigarrow}{}^* [\![\Gamma, x : B \vdash u' : C]\!]$ que l'on peut représenter de la manière suivante :



On peut ensuite connecter la conclusion de type $?B^\perp$ (représentant la variable x) et celle de type C (représentant le type de sortie) par un lien $\parr$ qui conserve la réduction des réseaux :



La variable x devient liée et n'est plus capturée par le contexte Γ. On obtient finalement la réduction $[\![\Gamma \vdash \lambda x^B.t' : B \to C]\!] \overset{\text{cut}}{\rightsquigarrow}{}^* [\![\Gamma \vdash \lambda x^B.u' : B \to C]\!]$ comme attendu.

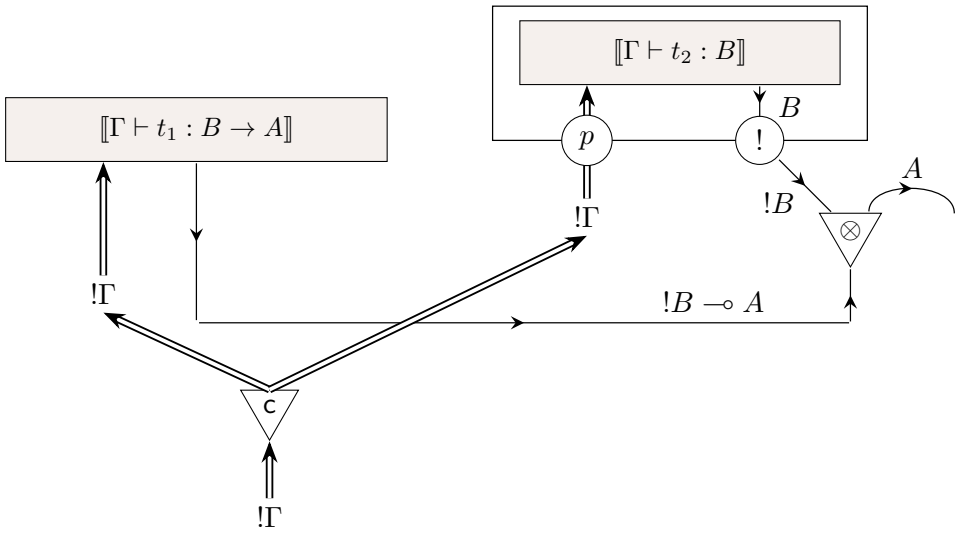
▶ Si $t = t_1\ t_2$ avec $\Gamma \vdash t_1 : B \to A$ et $\Gamma \vdash t_2 : B$.

↠ Si $t = (\mathtt{Y}\ t_2)$ et $B = A \to A$ on utilise la réduction $\mathtt{Y}$-promotion pour obtenir exactement $[\![\Gamma \vdash t_2\ (\mathtt{Y}\ t_2) : A]\!]$.

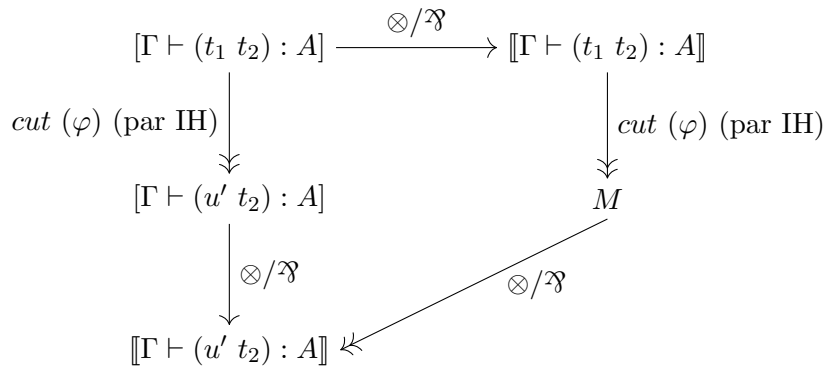
Sinon on a 3 sous-cas possibles :

↠ Réduction à gauche : $u = u'\ t_2$ et t_1 se réduit à u'. Par hypothèse d'induction, $[\![\Gamma \vdash t_1 : B \to A]\!] \overset{\text{cut}}{\rightsquigarrow}{}^* [\![\Gamma \vdash u' : B \to A]\!]$.

La pré-traduction nous donne un réseau $[\Gamma \vdash (t_1\ t_2) : A]$ dont la conclusion de $[\![\Gamma \vdash t_1 : B \to A]\!]$ peut être la conclusion d'un lien $\parr$:



On sait par hypothèse d'induction que le réseau se réduit en $[\Gamma \vdash (u'\ t_2) : A]$ par un certain nombre d'élimination de coupures que l'on regroupe sous l'identifiant φ mais aussi que l'on peut éliminer l'éventuellement coupure entre t_1 et la boîte de t_2 pour avoir $[\![\Gamma \vdash (t_1\ t_2) : A]\!]$. Cette divergence est représentée par le schéma ci-dessous :



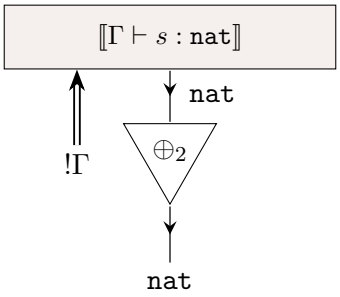
Vers le bas : par définition de la traduction, on peut éliminer l'éventuelle coupure multiplicative introduite par u' pour passer de $[\Gamma \vdash (u'\ t_2) : A]$ à $[\![\Gamma \vdash (u'\ t_2) : A]\!]$.

Vers la droite : le but est de joindre la même conclusion par $[\![\Gamma \vdash (t_1\ t_2) : C]\!]$. Pour cela on réalise les opérations de réduction φ pour atteindre un certain réseau M grâce à l'hypothèse d'induction. On élimine finalement les éventuelles coupures multiplicatives introduites par φ pour obtenir $[\![\Gamma \vdash (u'\ t_2) : A]\!]$ après mise en ?-forme normale.

↠ Réduction à droite : $u = (t_1\ u')$ et $t_2 \rightsquigarrow u'$. On procède dans le même esprit que précédemment. Comme $[\![\Gamma \vdash t_2 : B]\!]$ est emboîté (et donc isolé) on peut le réécrire directement en $[\![\Gamma \vdash u' : B]\!]$ par hypothèse d'induction.

↠ Dans le cas où t est un rédex, on applique le lemme de *simulation de la β-réduction* (section 4).

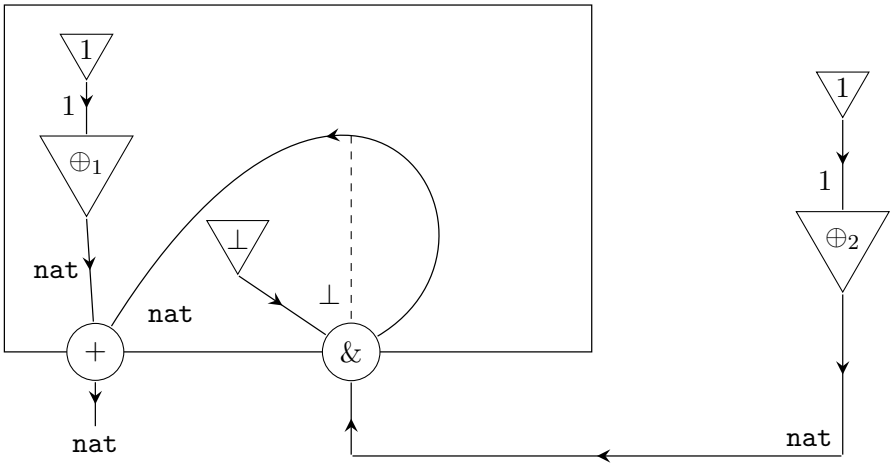
▶ Si $t = \texttt{succ}\ s$. Soit s ne contient pas de rédex et la preuve est triviale, soit $(\Gamma \vdash t : \texttt{nat} \rightsquigarrow \Gamma \vdash \texttt{succ}\ s' : \texttt{nat})$ et $(\Gamma \vdash s : \texttt{nat} \rightsquigarrow \Gamma \vdash s' : \texttt{nat})$. Dans ce dernier cas on obtient le réseau suivant :



On réécrit le réseau $[\![\Gamma \vdash s : \texttt{nat}]\!]$ en $[\![\Gamma \vdash s' : \texttt{nat}]\!]$ en appliquant l'hypothèse d'induction.

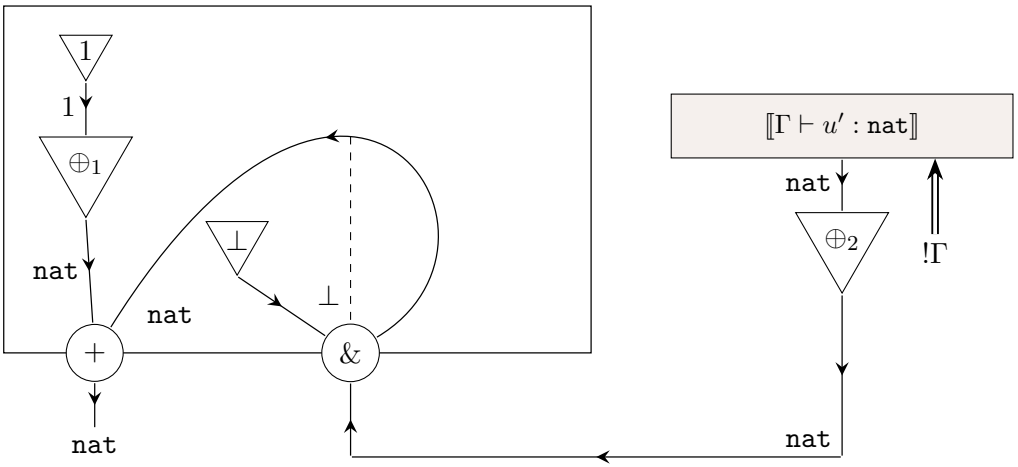
▶ Si $t = \texttt{pred}\ u$. Soit u ne contient pas de rédex et la preuve est triviale, soit on a deux réductions possibles selon la forme de u :

↠ Si $t = \texttt{pred zero}$



On élimine la coupure additive ce qui efface la partie droite de la boîte. Il nous reste le contenu de la partie gauche contenant le réseau $[\![\Gamma \vdash \texttt{zero} : \texttt{nat}]\!]$ ce qui correspond bien au prédécesseur de `zero`.

↠ Si $t = \texttt{pred}\ (\texttt{succ}\ u')$



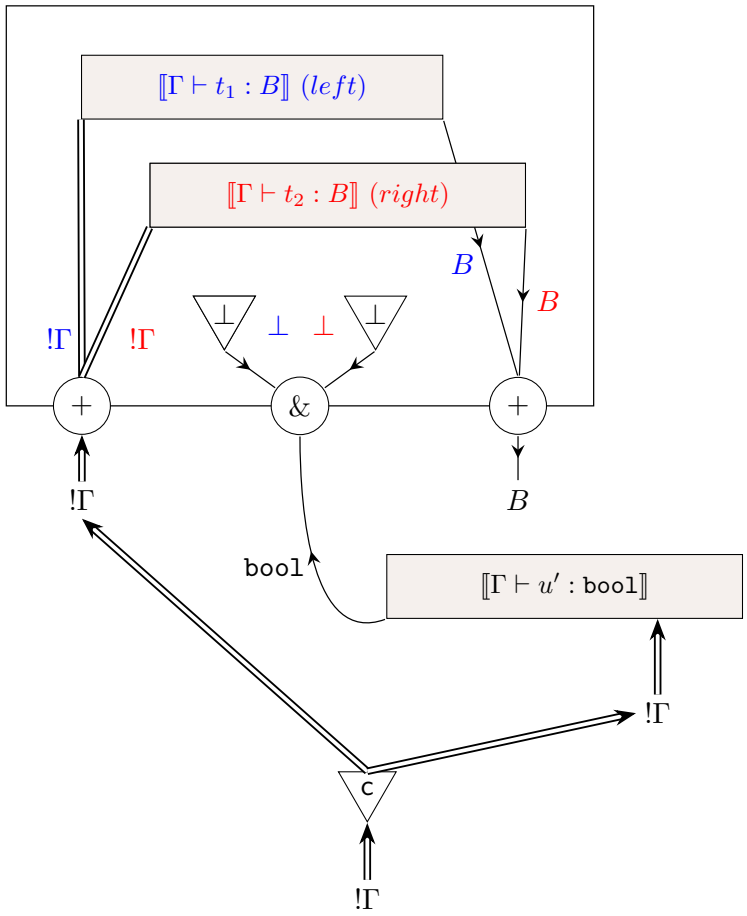
On élimine la coupure additive et on obtient le réseau $[\![\Gamma \vdash u' : \texttt{nat}]\!]$ qui est bien le réseau du prédécesseur de $u = \texttt{succ}\ u'$.

▶ Si $t = \texttt{iszero}\ s$. Si s contient un rédex, la preuve est triviale. Sinon, on raisonne sur la forme de s :

↠ Soit $t = \texttt{iszero zero}$ et on réduit la coupure additive pour avoir $[\![\Gamma \vdash \texttt{true} : \texttt{bool}]\!]$.

↠ Soit $t = \texttt{iszero}\ (\texttt{succ}\ s')$ et on réduit la coupure additive pour avoir $[\![\Gamma \vdash \texttt{false} : \texttt{bool}]\!]$.

▶ Si $t = \texttt{if}\ u'\ \texttt{then}\ t_1\ \texttt{else}\ t_2$ sachant qu'on a $\Gamma \vdash t : B$. On a 5 réductions possibles sur le réseau suivant :



→ Soit u' contient un rédex et dans ce cas on utilise la réécriture de réseau offerte par l'hypothèse d'induction. Le théorème de *préservation de type* (section 4) de PCF nous assure que si u' se réduit en u'' alors $\Gamma \vdash u'' : \texttt{bool}$.

→ Soit t_1 ou t_2 contient un rédex. Dans ces derniers cas, on procède comme précédemment.

→ Soit $u' = \texttt{true}$ et t se réduit en t_1 ou alors $u' = \texttt{false}$ et t se réduit en t_2. Dans ces deux cas on prends le réseaux de t et on élimine la coupure additive pour finalement obtenir soit t_1 ou t_2 : par conséquent la réduction est bien conservée dans les réseaux.

□

5 Extension à λ_{lsub}

Dans l'objectif d'étudier une correspondance exacte entre les substitutions explicites et les réseaux, on fait le choix arbitraire d'étendre PCF au *calcul de substitution linéaire (Linear Substitution Calculus)* abrégé en λ_{lsub} [2], un calcul parmi tant d'autres.

Le calcul λ_{lsub} est une généralisation du calcul λ_m de Milner [23] dont la règle $\mapsto_{db}$ provient du *λ-calcul structurel (structural λ-calculus)* [3, 4].

5.1 Syntaxe

On nomme le calcul obtenu PCF+λ_{lsub} dont la syntaxe des termes est la suivante :

$$s, t, u ::= x \mid \lambda x^A.t \mid t\ u \mid \texttt{zero} \mid \texttt{succ}\ t \mid \texttt{true} \mid \texttt{false} \mid \texttt{Y}_A \mid$$
$$\texttt{pred}\ t \mid \texttt{iszero}\ t \mid \texttt{if}\ t\ \texttt{then}\ u_1\ \texttt{else}\ u_2 \mid t\langle x := u\rangle$$

Où les identifiants peuvent être indicés par un entier naturel i. La substitution n'est plus vue comme une opération de réécriture externe mais comme partie intégrante de la syntaxe. De plus elle est vue comme un *lieur (binder)* de variable au même titre que l'abstraction.

Ensuite, on introduit une nouvelle notion, celle de **λ-contexte**. Un λ-contexte ou simplement *contexte* (quand il n'y a pas d'ambiguïtés) est un terme qui contient exactement *un* trou qui peut être rempli :

$$C ::= \Box_S \mid \lambda x^A.C \mid C\ t \mid t\ C \mid \texttt{succ}\ C \mid \texttt{pred}\ C \mid \texttt{iszero}\ C \mid$$
$$\texttt{if}\ C\ \texttt{then}\ u_1\ \texttt{else}\ u_2 \mid \texttt{if}\ t\ \texttt{then}\ C\ \texttt{else}\ u_2 \mid \texttt{if}\ t\ \texttt{then}\ u_1\ \texttt{else}\ C \mid$$
$$C\langle x := u\rangle \mid t\langle x := C\rangle$$

Les trous sont indexés par un ensemble S de variables. On écrit $C[t]$ pour le remplissage du trou de C par le terme t et $C[\![t]\!]$ pour le remplissage de C par t quand les variables libres de t ne sont pas capturées par C (aucune abstraction ou substitution explicite ne lie x). Quand on insère un terme t dans un trou $\Box_S$, on doit forcément avoir $\texttt{fv}(t) \subseteq S$.

Une particularité des contextes est qu'ils ne sont pas considérés modulo α-conversion et ne sont pas sujets à la β-réduction. Le concept de "trou" est un concept externe au langage, un *meta-concept* au même titre que la substitution implicite.

5.2 Evaluation

Au niveau des réductions, on ne considère plus la règle β et on ajoute quatre nouvelles règles :

$$\frac{}{(\lambda x.t)Lu \rightsquigarrow t\langle x := u\rangle L}\ (db)$$

où L est une liste de substitutions éventuellement vide. Le choix de conserver une liste de substitutions est motivé par l'encodage de la réduction dans les réseaux de preuve [2].

$$\frac{|C[\![x]\!]|_x > 1}{C[\![x]\!]\langle x := u\rangle \rightsquigarrow C[\![u]\!]\langle x := u\rangle}\ (c) \qquad \frac{|C[\![x]\!]|_x = 1}{C[\![x]\!]\langle x := u\rangle \rightsquigarrow C[\![u]\!]}\ (d)$$

Le nombre d'occurrences de x dans C est représenté par $|C[\![x]\!]|_x$. Ces deux règles sont habituellement formulées par une seule règle $C[\![x]\!]\langle x := u\rangle \mapsto_{ls} C[\![u]\!]\langle x := u\rangle$ mais on préfère la séparer en deux pour une meilleure adéquation avec les réseaux.

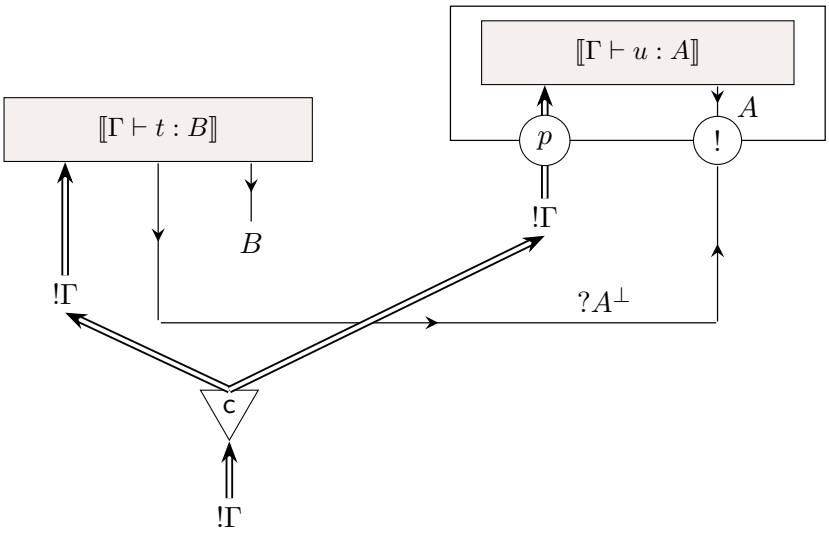
$$\frac{}{t\langle x := u\rangle \rightsquigarrow t}\ (gc) \qquad \text{si } x \notin \texttt{fv}(t)$$

Les identifiants *db*, *ls*, *c*, *d* et *gc* signifient respectivement *distant beta*, *linear substitution*, *contraction*, *dereliction* et *garbage collection*.

5.3 Traduction des substitutions explicites dans les réseaux

Le terme de substitution $t\langle x := u\rangle$ est traduit par un réseau qui relie une boîte de promotion contenant $[\![\Gamma \vdash u : A]\!]$ à un fil correspondant à la variable x sortant du réseau $[\![\Gamma \vdash t : B]\!]$:

$$[\Gamma \vdash t\langle x := u\rangle : B] =$$



Si x n'apparaît pas dans t, on ajoute une cellule d'affaiblissement dans $[\![\Gamma \vdash t : B]\!]$ auquel sera connectée la boîte de promotion.

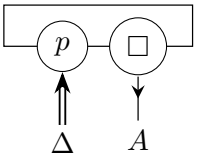
Les substitutions explicites correspondent en fait exactement à la notion de boîte de promotion.

Dans le cadre des substitutions explicites, éliminer les coupures multiplicatives lors de la traduction risque d'identifier le rédex $(\lambda x.t)u$ et le terme $t\langle x := u\rangle$. Pour éviter cela, on évite d'éliminer les coupures multiplicatives. La traduction est donc simplement donnée par la ?-forme normale de la pré-traduction. Il faut noter qu'en échange on perd l'adéquation avec les σ-équivalences.

5.4 Traduction des contextes dans les réseaux

Les termes étant disjoints des contextes, il faut aussi définir une traduction pour les contextes. Pour cela on introduit une nouvelle boîte appelée *hole-box* qui se présente exactement comme une boîte de promotion avec un comportement différent :

$$[\![\Delta \vdash \Box_\Delta : A]\!] =$$



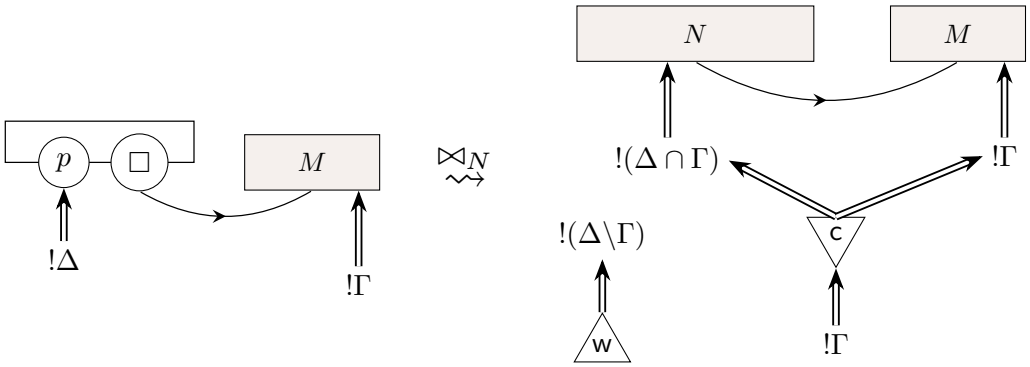
La boîte est paramétrée par un contexte Δ et un type A. La traduction des contextes se base sur la grammaire des λ-contextes, la traduction des λ-termes et la hole-box ci-dessus.

En fait, on ne fait qu'esquisser une idée incomplète et sûrement inexacte. Pour une investigation plus sérieuse, on peut se référer aux travaux de *Beniamino Accattoli*.

On se retrouve finalement avec deux sortes de réseaux :

- Les *réseaux réguliers* comme ceux que l'on a toujours présenté précédemment (sans hole-box).
- Les *réseaux contextuels* qui ont *une unique* occurrence de hole-box (c'est le critère de correction des réseaux contextuels qui simule l'unicité des trous $\Box$ dans les λ-contextes). Les réseaux contextuels sont définis inductivement selon la traduction des contextes.

On introduit une procédure de transformation des réseaux contextuels vers les réseaux réguliers qu'on appelle *plugging* (noté $\bowtie$ ou $\bowtie_N$ pour une injection d'un réseau régulier N) :



La procédure admet les effets de bord suivants :

- On ajoute une cellule d'affaiblissement dans la boîte pour chaque variable libre $\Delta\backslash\Gamma$ qui n'est pas utilisée.
- On contracte les nouvelles variables libres $\Delta \cap \Gamma$ introduites par N.

5.5 Simulation des réductions

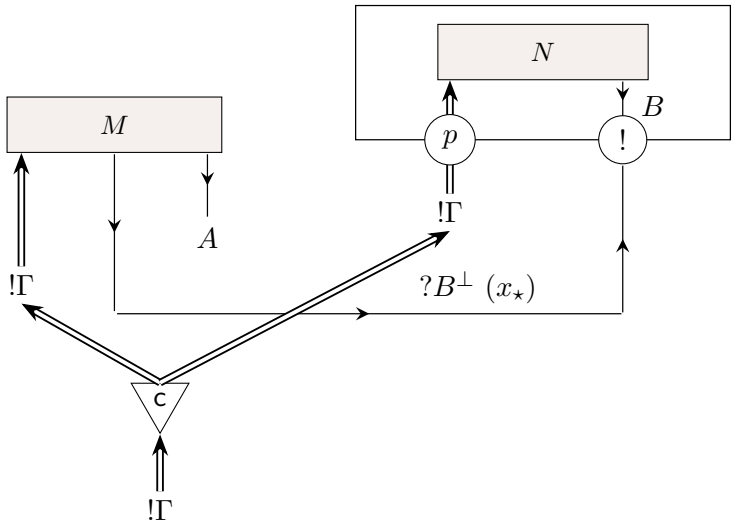
Comme on n'élimine plus les coupures multiplicatives on perd la correspondance exacte entre rédex et réseau sans coupure vu que la traduction d'une application introduit *toujours* une coupure multiplicative $\parr/\otimes$.

On remarque aussi que l'on a plus de simulation pour la β-réduction vu que la règle a disparu.

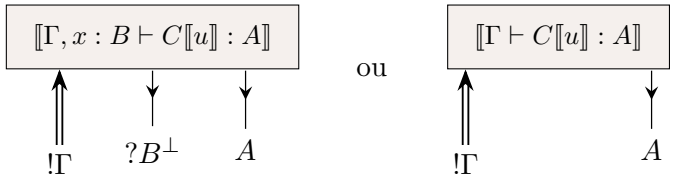
De plus on fait le choix d'omettre le contexte S des trous $\Box_S$ en supposant que dans un séquent $\Gamma \vdash A$, les trous sont paramétrés avec Γ.

Lemme (Substitution linéaire)

Si $M = [\![\Gamma, x : B \vdash C[\![x]\!] : A]\!]$ et si $N = [\![\Gamma \vdash u : B]\!]$ alors le réseau M relié, par une coupure, à une boîte de promotion contenant le réseau N avec éventuellement quelques contractions sur x :



se réduit en $[\![\Gamma, x : B \vdash C[\![u]\!] : A]\!]$ ou $[\![\Gamma \vdash C[\![u]\!] : A]\!]$ selon si x est libre ou non dans $C[\![u]\!]$.



Démonstration
Soit $x_\star$, l'occurrence de x insérée dans C. On procède par induction sur C.

- ↠ Si $C = \square$, dans $[\![\Gamma \vdash \square[\![x]\!]\langle x := u\rangle : B]\!]$ on a une variable représentée par une cellule de déréliction connectée à une boîte de promotion contenant $[\![\Gamma \vdash u : A]\!]$. On élimine la coupure déréliction-promotion pour remplacer $x_\star$ par le contenu de la boîte pour obtenir $[\![\Gamma \vdash \square[\![u]\!] : A]\!]$.

- ↠ Si $C = \lambda y.C'$ avec $y \neq x$, on réécrit partiellement le réseau grâce à l'hypothèse d'induction sans tenir compte de la cellule d'abstraction $\parr$.

- ↠ Si $C = (C'\ t_1)$, le cas est similaire.

- ↠ Si $C = (t_1\ C')$, on élimine la commutation exponentielle pour faire rentrer la boîte de u à l'intérieur de celle de C' et on applique l'hypothèse d'induction.

- ↠ Si C est $(\texttt{succ}\ C'), (\texttt{pred}\ C')$, $(\texttt{iszero}\ C')$ ou $(\texttt{if}\ C'\ \texttt{then}\ t_1\ \texttt{else}\ t_2)$, on procède de manière similaire au cas $C = (C'\ t_1)$.

$\twoheadrightarrow$ Si C est `if` t_1 `then` C' `else` u_2 ou `if` t_1 `then` u_2 `else` C', on procède de manière similaire au cas $C = (t_1\ C)$ sauf qu'on a une commutation additive à la place d'une commutation exponentielle pour faire rentrer la boîte de u dans la boîte additive de condition.

$\twoheadrightarrow$ Si $C = C'\langle y := u_2\rangle$ avec $y \neq x$ et $u_2 \neq u_1$, on procède de manière similaire au cas $C = (C'\ t_1)$.

$\twoheadrightarrow$ Si $C = t_1\langle y := C'\rangle$ avec $y \neq x$, on a un réseau correspondant à t_1 connecté à une boîte de promotion contenant $[\![\Gamma \vdash C'[\![x]\!] : A]\!]$ elle-même connectée à une boîte de promotion contenant $[\![\Gamma \vdash u : B]\!]$. On élimine la commutation exponentielle pour faire rentrer cette dernière boîte dans la première pour avoir $[\![\Gamma \vdash t_1\langle y := C'[\![u]\!]\rangle : A]\!]$ qui est équivalent à $[\![\Gamma \vdash (t_1\langle y := C'\rangle)[\![u]\!] : A]\!]$ puisque le trou de C est localisé dans C'.

□

Théorème (Simulation de l'évaluation de PCF+λ_{lsub})

Soit t et u deux termes de PCF+λ_{lsub} typés sous un contexte Γ.
Si $\Gamma \vdash t : A \rightsquigarrow \Gamma \vdash u : A$ alors $[\![\Gamma \vdash t : A]\!] \overset{\text{cut}}{\rightsquigarrow}{}^{*} [\![\Gamma \vdash u : A]\!]$.

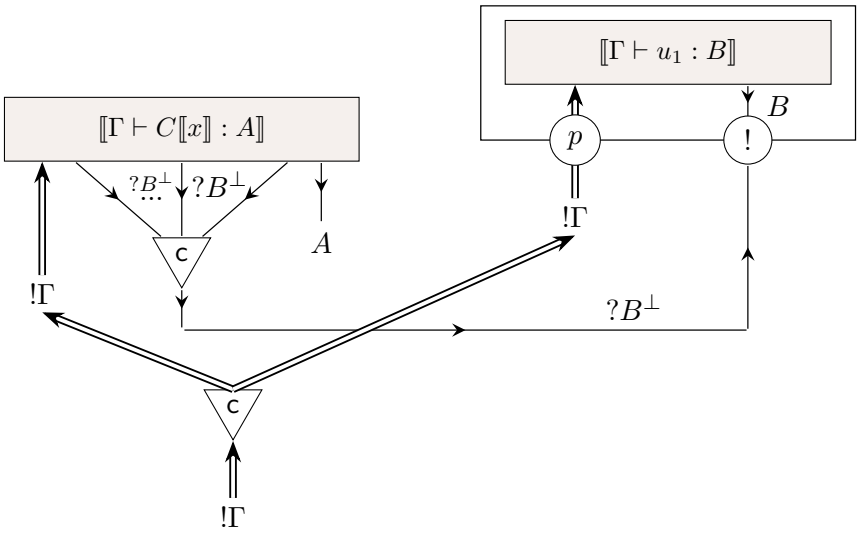
Démonstration.
Par induction sur la relation de réduction $\rightsquigarrow$,

▶ Règle $db : (\lambda x.t_1)Lu_1 \rightsquigarrow t\langle x := u_1\rangle L$.
Dans le cadre des substitutions explicites la preuve est triviale : il suffit d'éliminer la coupure multiplicative $\parr/\otimes$ introduite par l'application de u_1 à $(\lambda x^A.t_1)$ pour exhiber la coupure exponentielle derrière. Comme les boîtes de promotion correspondant à L ne sont pas affectées par la procédure, on a exactement $[\![\Gamma \vdash t_1\langle x := u_1\rangle L]\!]$.

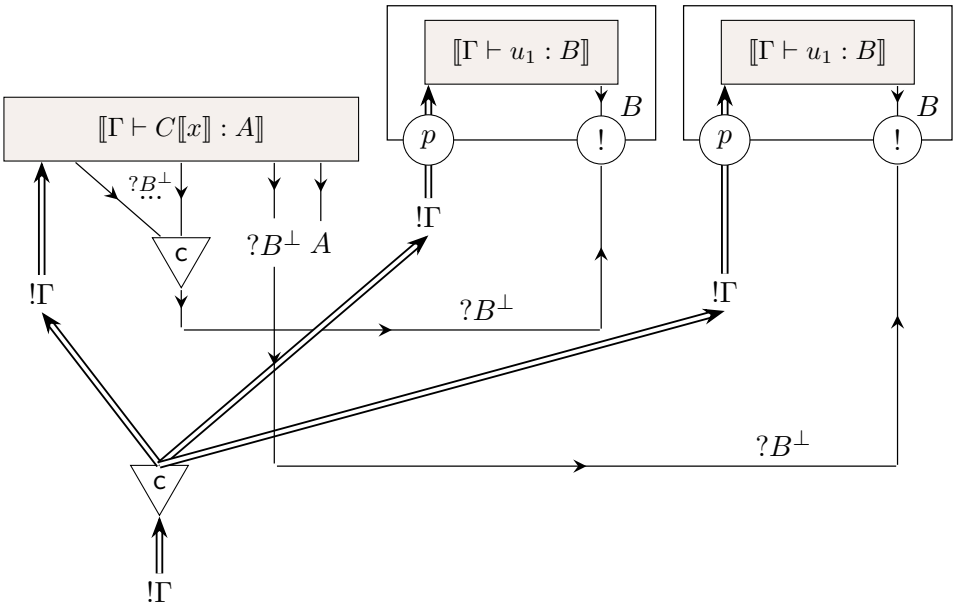
▶ Règle $c : C[\![x]\!]\langle x := u_1\rangle \rightsquigarrow C[\![u]\!]\langle x := u_1\rangle$ avec $|C[\![x]\!]|_x > 1$. On raisonne par cas sur la structure de C.

$\twoheadrightarrow$ Si $C = \Box$, hypothèse contradictoire car on a $|\Box[\![x]\!]|_x = 1$.

$\twoheadrightarrow$ Soit $x_\star$ l'occurrence de x insérée dans le contexte C. Pour les autres cas, comme on a au moins deux occurrences de x dans le contexte, on a un lien de contraction connecté à la boîte de promotion introduite par $\langle x := u_1\rangle$.



Il faut exécuter une étape d'élimination de coupure contraction-promotion pour extraire le fil de $x_\star$ avec une copie de boîte contenant $[\![\Gamma \vdash u_1 : B]\!]$:



On procède ensuite de la même manière que dans le lemme de substitution linéaire que l'on utilise pour conclure.

▶ Règle $d : C[\![x]\!]\langle x := u_1\rangle \rightsquigarrow C[\![u]\!]$ avec $|C[\![x]\!]|_x = 1$. On raisonne par cas sur la structure de C.

↠ Si $C = \square$, on a le réseau $[\![\Gamma \vdash \square[\![x]\!]\langle x := u_1\rangle : A]\!]$. Par le lemme de substitution linéaire, l'occurrence de x insérée dans le trou est remplacée par u_1 et la boîte de promotion qui la contenait disparaît. On obtient finalement $[\![\Gamma \vdash \square[\![u]\!] : A]\!]$.

$\twoheadrightarrow$ Les autres cas sont similaires à la simulation de la règle (c) sauf qu'il n'est pas nécessaire d'éliminer une coupure contraction-promotion. On applique directement le lemme de substitution linéaire (ce qui consomme l'unique boîte de promotion associée à x) après avoir éliminé les commutations exponentielles et additives si nécessaire.

▶ Règle $gc : t_1\langle x := u_1\rangle \rightsquigarrow t_1$ avec $x \notin \mathtt{fv}(t_1)$.

Comme on a $x \notin \mathtt{fv}(t_1)$, la boîte de promotion contenant u_1 est reliée à une cellule d'affaiblissement. En éliminant la coupure affaiblissement-promotion, la boîte est détruite et on termine sur le réseau $[\![\Gamma \vdash t_1 : A]\!]$.

▶ Les autres règles de réduction sont simulées de la même façon que pour PCF.

□

6 Ouvertures

- On aurait pu investiguer plus en profondeur le concept de σ-équivalence pour PCF. Quels sont les termes syntaxiquement différents mais que l'on peut représenter par un même réseau pour capturer la notion d'équivalence opérationnelle ?
- On aurait pu explorer une traduction du Système T de Gödel dans les réseaux de preuve (déjà suggéré par Gimenez dans sa thèse). La grande différence avec PCF se trouve dans la récursion primitive, une forme contrôlée de réduction par rapport au combinateur Y.
- On peut investiguer plus en profondeur la traduction de λ_{lsub} dans les réseaux et vérifier des propriétés et éventuellement proposer une traduction plus intelligente.

References

[1] Martin Abadi, Lucas Cardelli, Pierre-Louis Curien, Jean-Jacques Lévy, *Explicit Substitutions*. 1990.

[2] Beniamino Accattoli, Eduardo Bonelli, Delia Kesner, Carlos Lombardi, *A Nonstandard Standardization Theorem*. 2014.

[3] Beniamino Accattoli, *Jumping around the box: Graphical and operational studies on λ-calculus and Linear Logic (PhD Thesis)*, 2011

[4] Beniamino Accattoli, Delia Kesner, *The structural lambda-calculus*. 2010.

[5] Marc Bagnol, *Réseaux de preuve pour MALL*. 2013.

[6] Vincent Danos, *Une Application de la Logique Linéaire a l'Etude des Processus de Normalisation (principalement du λ-calcul) (Thèse de doctorat)*, 1990

[7] Roberto Di Cosmo, Vincent Danos, *The Linear Logic Primer*, 1996

[8] Roberto Di Cosmo, Stefano Guerrini, *Strong Normalization of Proof Nets Modulo Structural Congruences*. 1999.

[9] Roberto Di Cosmo, Delia Kesner, *Strong Normalization of Explicit Substitutions via Cut Elimination in Proof Nets*. 1996.

[10] Roberto Di Cosmo, Delia Kesner, Emmanuel Polonovski, *Proof Nets and Explicit Substitutions*. 2000.

[11] Marc de Falco, *An Explicit Framework for Interaction Nets*, 2009.

[12] Marc de Falco, *Géométrie de l'Interaction et Réseaux Différentiels (Thèse de doctorat)*. 2009.

[13] Stéphane Gimenez, *Programmer, calculer et raisonner avec les réseaux de la logique linéaire (Thèse de Doctorat)*. 2009.

[14] Jean-Yves Girard, *Linear Logic*. 1986.

[15] Jean-Yves Girard, *Proof-Nets : The Parallel Syntax for Proof Theory*. 2013.

[16] Martin Abadi, Luca Cardelli, Pierre-Louis Curien, Jean-Jacques Lévy, *Explicit Substitutions*. 1990.

[17] Yves Lafont, *Interaction Nets*, 1989

[18] Yves Lafont, *The paradigm of interaction (Short version)*. 1991.

[19] Yves Lafont, *From Proof-Nets to Interaction Nets*. 1994.

[20] Olivier Laurent, *Théorie de la Démonstration (Notes de cours)*, 2011

[21] Olivier Laurent, *An Introduction to Proof Nets*, 2013

[22] Damiano Mazza, *Interaction Nets : Semantics and Concurrent Extensions (Thèse de doctorat)*. 2006.

[23] Robin Milner, *Local Bigraphs and Confluence: Two Conjectures*, 2007

[24] Raphaël Montelatici, *Polarized Proof Nets with Cycles and Fixpoints Semantics*, 2003

[25] Benjamin C. Pierce, *Types and Programming Languages*. The MIT Press, 2002.

[26] G.D Plotkin, *LCF considered as a Programming Language*. 1994.

[27] Laurent Regnier, *Lambda-calcul et réseaux (Thèse de doctorat)*, 1992

[28] Laurent Regnier, *Une équivalence sur les lambda-termes*, 1993

[29] Laurent Regnier, *Les limites de la correspondance preuve/programme*, 2003

[30] Peter Selinger, *Lecture Notes on the Lambda Calculus*. 2008.