

Rapport de Stage de M2

Des Systèmes de Types pour la Terminaison en Concurrency

Romain Demangeon

1 Domaine du Stage: le Terminaison

Ce stage porte sur la question de la terminaison en programmation concurrente en général et en π -calcul en particulier. Son point de départ est l'article [13] qui propose des systèmes de types (numérotés de 1 à 4) pour le π -calcul assurant la terminaison (Le π -calcul sera expliqué en détail plus loin dans le rapport). Ce rapport présente une version volontairement accessible du travail accompli au cours de ce stage, des documents de travail sont disponibles [3], contenant, entre autres, les preuves dans leur intégralité et l'introduction formelle du symbolisme qui leur est nécessaire.

La première section de ce rapport situe la question de la terminaison dans le paysage de la programmation et la seconde présente le travail préliminaire sur le sujet. La contribution du stage est ensuite développée, d'abord l'inférence des systèmes de types existants, puis l'introduction d'une analyse plus fine, et enfin le travail relatif à des systèmes de types plus expressifs.

1.1 La Terminaison dans les systèmes de réécriture

La terminaison est un problème récurrent en programmation. On veut pouvoir s'assurer qu'un programme donné termine, car la divergence est souvent un syndrome de non-correction. Le formalisme des systèmes de réécriture donne lieu à une définition de la terminaison (aussi appelée normalisation forte): il n'existe pas de dérivation infinie.

Le problème de la terminaison d'un système de réécriture (savoir si un système de réécriture donné contient une dérivation infinie) dans le cas général est indécidable [12], donc il n'existe pas d'algorithme général pouvant déceler les programmes qui ne terminent pas.

Des techniques ont été développées pour s'assurer de la terminaison de certains systèmes de réécriture. On notera l'existence de techniques 'd'ordre de réduction' [5] qui utilisent un ordre bien-fondé (noethérien). Une fois trouvé un tel ordre $>$ pour un système de réécriture S , il suffit de prouver que chaque terme de S est strictement plus grand pour $>$ que chaque terme auquel il se réduit. La terminaison de l'ordre force ainsi la terminaison du système de réécriture.

Les ordres polynomiaux qui associent à chaque symboles de fonction un polynôme entier en ses arguments (et donc des naturels pour les symboles de constante) sont des ordres bien-fondés et sont utilisés pour prouver la terminaison de certains systèmes simples.

Les ordres de simplifications (chaque terme est strictement supérieur à n'importe lequel de ses sous-termes), comme l'enchâssement, les RPO (Recursive Path Ordering), les ordres multi-ensemble ou les ordres de Knuth-Bendix sont utilisés dans l'automatisation des preuves de terminaison.

Des outils ont été développés pour tester la terminaison. Citons [9] pour les systèmes de réécriture, [2] pour Prolog et, pour les programmes séquentiels "bas niveau" (typiquement: du code qui tourne dans un système d'exploitation), les travaux de B. Cook et ses coauteurs sur l'outil Terminator [4]

1.2 Application en théorie de la démonstration

Le λ -calcul est un contexte dans lequel le problème de la terminaison a été beaucoup étudié, notamment du fait des liens avec la théorie de la démonstration. Le λ -calcul, étant un modèle de calcul Turing-complet,

peut exprimer des divergences (par exemple $\Omega = (\lambda x.xx)(\lambda x.xx)$ se réduit sur lui même en une étape). On décore les termes du λ -calcul avec des types, d'une part pour indiquer l'usage d'une fonction (par exemple: `float -> int`), d'autre part pour s'assurer de certaines propriétés sur ces termes.

Ainsi le λ -calcul simplement typé est terminant, c'est à dire que tout terme simplement typable termine. Par contre il existe des termes terminants qui ne sont pas simplement typables. Le λ -calcul simplement typé est en isomorphisme de Curry-Howard avec la logique minimale.

De nouvelles techniques de démonstration de terminaison s'appuyant sur la notion de type ont été introduites, comme les candidats de réductibilité dans [6]: la réductibilité d'un terme se définit par induction sur les types: le terme t de type $\Pi X.T$ est réductible si pour tout type U , tU est réductible, on montre ensuite que tous les termes du système sont réductibles, puis que tous les termes réductibles sont terminants. Cette technique peut être utilisée pour prouver la terminaison du *lambda*-calcul simplement typé, donc la cohérence de la logique minimale (on ne peut y prouver une proposition fausse). Des extensions de ce calcul, plus expressives, comme le système F de Girard [6] (une extension contenant la quantification universelle de type), ou le Calcul des Constructions, donnent des correspondances de Curry-Howard vers des logiques plus riches. A chaque fois, la cohérence de la logique est assurée par la terminaison du calcul associé et la technique des candidats de réductibilité est adaptée.

1.3 Terminaison et Programmes Parallèles

Des outils ont été implémentés pour vérifier la terminaison de programmes, en analysant statiquement le code et produisant une preuve mathématique de sa terminaison (comme [4]).

Dans le cas de la programmation parallèle, la terminaison prend un sens un peu différent. Lorsque l'on modélise un serveur qui peut recevoir des requêtes de la part de clients, il n'est pas essentiellement gênant que le programme du serveur ne soit pas terminant (il continue de s'exécuter pour attendre une requête). Toutefois, le partage de ressource encourage là encore le développement d'outils en rapport avec la terminaison: la coexistence de plusieurs processus dont la terminaison n'est pas assurée peut entraîner des phénomènes de famine, et de ce fait, garantir la terminaison permet d'éviter des dénis de service.

[1] constitue un bon exemple de travail déjà existant dans lequel de fortes hypothèses sont faites sur le modèle de parallélisme. Un programme pour prouver la terminaison de threads tournants en parallèle est développé, prouvant ainsi la correction de certains drivers de Windows Vista.

Le π -calcul se veut un modèle mathématique du parallélisme. Il est suffisamment expressif pour que l'on puisse y encoder la programmation fonctionnelle, la programmation impérative ou la programmation orientée objet. Les découvertes théoriques liées au π -calcul se retrouvent dans des langages de programmation récents. Ainsi *Polyphonic C \sharp* , une extension du C \sharp de Microsoft, se base sur le join-calcul, une variante du π -calcul à laquelle les résultats présentés ici s'étendent naturellement.

Dans [11], la terminaison d'un sous-ensemble restreint du π -calcul est prouvée en se basant sur l'encodage du λ -calcul. [13], le travail sur lequel ce stage est fondé, propose un système de types pour le π -calcul qui ne se focalise pas sur l'encodage du λ -calcul, mais s'attache à définir des critères pour la terminaison 'propres au π -calcul'.

Dans [8], Naoki Kobayashi propose un système de types pour éviter les famines en programmation concurrente (*lock-freedom*). Pour écarter les programmes amenant à des interblocages (*livelock*), il est nécessaire de s'assurer de la terminaison simultanée de plusieurs sous-processus. Une implémentation de ce système de types a été réalisée: TyPiCal, un analyseur de type statique pour le π -calcul [7]. Cela motive non seulement la recherche de systèmes de types larges (typant un nombre élevé de programmes usuels), mais force aussi à s'intéresser à la complexité de l'inférence de ces systèmes.

2 Un Système de Types pour la Terminaison en Concurrency

2.1 CCS, un modèle simple

CCS (Calculus of Communicating Systems) de Milner [10], est le modèle simple et originel de la concurrence dans lequel les agents se synchronisent sur des canaux, représentés par des noms. On s'efforcera, dans ce rapport, de présenter les résultats, obtenus en π -calcul, en *CCS* à chaque fois que c'est possible, afin d'en simplifier la présentation.

2 actions sont possibles sur un canal a : la réception (a) et l'émission (\bar{a}). Les processus (agents) sont construits avec comme outil la séquence d'action, la mise en parallèle et la réplication (symbolisés respectivement par $.$, $|$ et $!$). 2 processus en parallèle peuvent se synchroniser s'il peuvent effectuer des actions correspondantes sur le même canal, i.e. si l'un des deux a comme préfixe a et l'autre \bar{a} . Par exemple: $a.P | \bar{a}.Q$ se réduit en $P | Q$. Cette opération symbolise une communication de type *handshake*, caractérisant la simplicité de *CCS* face au π -calcul, où des informations peuvent être échangées dans ces communications.

La réplication permet de créer un nombre arbitraire d'instances d'un même agent. C'est ce dernier constructeur qui induit la notion de divergence, *CCS* sans la réplication étant trivialement terminant (on ne peut que réduire la taille de l'ensemble des termes à chaque étape de réduction). Un processus de la forme $!a.P$ dénote donc une réplication initialisée par une réception.

Les expressions du langage sont interprétées comme un système de transition étiqueté (LTS): les actions étiquettent les transitions du système de réécriture. La communication devient une règle: Si $P \xrightarrow{a} P'$ et $Q \xrightarrow{\bar{a}} Q'$ alors $P | Q \xrightarrow{\tau} P' | Q'$. L'étiquette τ représente une transition interne au processus. On notera parfois τ_a dans ce rapport pour stipuler que c'est sur a que la synchronisation a lieu.

2.2 Divergence

Puisque seuls les processus s'initialisant par un réception peuvent être répliqués, la règle pour la réplication peut s'écrire $!a.P \xrightarrow{a} P | !a.P$, $!a.P$ jouant le rôle d'un nombre 'arbitrairement grand' de copies de P en parallèle.

On a vu que sans réplication, pas de divergence. Prenons pour exemple de processus divergent $P = \bar{a} | !a.\bar{a}$. On a la transition $P \xrightarrow{\tau_a} P$. De tels processus doivent être rejetés par les systèmes de type que l'on construit.

Dans $!a.\bar{a}$, on comprend qu'en 'consommant' un a (la réception), on 'produit' un a (l'émission).

De la même manière, considérons, $Q = !a.\bar{b} | !b.\bar{a} | \bar{a}$. L'agent Q diverge car $Q \xrightarrow{\tau_a} Q \xrightarrow{\tau_b} Q$: on consomme un a pour produire un b , on consomme ce b pour produire un a et on revient à la situation de départ.

Une notion de cycle commence à émerger: si on dispose d'une suite de noms $(a_i)_{i \in [1, n]}$ tel que consommer le nom a_i suffit à produire le nom a_{i+1} (modulo n), i.e. si il existe des sous-agents répliqués $!a_i.P_i$ avec \bar{a}_{i+1} dans P_i , alors le processus risque de diverger (sous condition d'initialisation du cycle: le processus doit contenir au moins un \bar{a}_i).

L'étape suivante est la généralisation aux multiensembles: si on dispose d'une suite de multiensembles de noms $(A_i)_{i \in [1, n]}$ tel que consommer les noms de A_i suffit à produire les noms de A_{i+1} (et si on peut initialiser le cycle), alors le processus va diverger. Prenons par exemple $R = !a.(\bar{b} | \bar{c}) | !b.\bar{d}.\bar{d} | !d.\bar{d}.\bar{a} | \bar{a}$.

Ainsi, lors de la construction de système de types, le point essentiel sera la règle de typage de la réplication. Le but est donc de décider quelles formes de réplication on peut autoriser, et quelles formes de réplication on doit rejeter, en gardant à l'esprit qu'on ne peut pas capturer tous les agents terminants, le problème de la terminaison de *CCS* étant indécidable.

2.3 Quelques Mots sur le π -calcul

Modèle de la concurrence, le π -calcul est plus complexe que *CCS*. Il est nécessaire d'en expliquer quelques différences, car certaines démarches effectuées dans ce stage ne prennent de sens qu'appliquées au π -calcul.

Les noms peuvent transporter d'autres noms: $\bar{p}\langle a \rangle$ signifie que le nom a est envoyé sur le canal p . Ainsi des variables apparaissent dans les réceptions, la règle de réception s'écrit, en sémantique *early*: $a(x).P \xrightarrow{av}$

$P[x \leftarrow v]$ où $[x \leftarrow v]$ est la substitution qui renomme x en v . Par exemple $a(x).x(y) \mid \bar{a}(v).\bar{v}(w)$ se réduit en $v(y) \mid \bar{v}(w)$.

Le π -calcul autorise la création dynamique de nom à l'aide de l'opérateur ν . Les noms créés ne sont connus que dans la continuation du processus, mais peuvent ensuite être envoyés hors de la portée de ce processus. Par exemple si $P_1 = a(x).x(y)$ et $P_2 = (\nu v)\bar{a}(v)$, alors dans $P_1 \mid P_2$ le nom v n'est connu que de P_2 . Ce nom peut être passé à P_1 , puisque $P_1 \mid P_2$ se réduit par τ_a en .

On utilisera aussi dans les exemples la conditionnelle 'if b then P else Q ' dont le rôle est assez intuitif.

2.4 Un premier système de types pour la terminaison

Le point de départ du stage est un article de Davide Sangiorgi [13] qui présente 4 systèmes de type plus ou moins emboîtés qui assurent la terminaison en π -calcul. Les systèmes 1 et 3 peuvent être détaillés en *CCS* sans difficulté.

Ces systèmes se basent sur l'attribution de niveaux aux différents noms utilisés dans un processus. Le système de niveaux est lexicographique: un nom de niveau t 'pèse plus lourd' qu'un nombre arbitraire de noms de niveau u si $t > u$. Quand on a un multiensemble de noms, on peut considérer le vecteur poids de cet ensemble, comme le multiensemble des niveaux de ces noms.

Par exemple si p a un niveau 3, a un niveau 2 et b et c un niveau 1. Le vecteur poids du multiensemble $\{a, a, a, p, b, b, c\}$ est $[1, 3, 3]$. Les vecteurs poids sont comparés de manière lexicographique.

Pour typer une réplication $!a.P$ avec le système 1, on compare le poids de cette réception au vecteur poids du multiensemble des émissions libres (qui ne sont pas sous une réplication) présents dans Q . On type la réplication si le poids de p est strictement supérieur au poids de chacune des émissions de Q . Ainsi on doit consommer un nom qui pèse plus lourd que tous les noms que l'on produit.

Si on considère le vecteur poids des émissions libres d'un processus au cours d'une dérivation, alors chaque utilisation de la règle réplication (et chaque utilisation de la règle de communication) fait décroître ce vecteur pour l'ordre lexicographique. Comme cet ordre est bien fondé, on ne peut pas décroître une infinité de fois et donc chaque terme typé est terminant.

Par exemple $P = !p.(\bar{a} \mid \bar{b}) \mid !a.\bar{b} \mid \bar{p} \mid \bar{p}$ est terminant car typable avec p au niveau 3, a au niveau 2 et c au niveau 1. Dans la première réplication on consomme un nom de niveau 3 pour récupérer des noms de niveaux 2 et 1 dans la deuxième réplication on consomme un nom de niveau 2 pour produire un nom de niveau 1. Si on considère la séquence de réduction $P \rightarrow^{\tau_p} \rightarrow^{\tau_a} \rightarrow^{\tau_p} \rightarrow^{\tau_a} !p.(\bar{a} \mid \bar{b}) \mid !a.\bar{b} \mid \bar{b} \mid \bar{b} \mid \bar{b}$, on constate que l'évolution du vecteur poids du multiensemble des émissions libres est $[2, 0, 0] \rightarrow [1, 1, 1] \rightarrow [1, 0, 2] \rightarrow [0, 1, 3] \rightarrow [0, 0, 4]$.

L'inconvénient principal de ce système est qu'il ne permet pas la présence d'*émissions récursives*: si la réplication est de la forme $!a.P$, alors P ne peut pas contenir d'émission sur a . Ce point est plutôt gênant lorsque l'on veut écrire des modèles propageant des requêtes: par exemple pour lancer une recherche sur le successeur d'un maillon dans une liste chaînée.

Le système 1 de [13] est décrit par les règles de typage suivantes, qui sont standard en π -calcul:

$$(\mathbf{T} - \mathbf{In}) \frac{a : \sharp T \quad x : T \quad \vdash P}{\vdash a(x).P}$$

$$(\mathbf{T} - \mathbf{Out}) \frac{a : \#T \quad v : T \quad \vdash P}{\vdash a\bar{v}.P}$$

$$(\mathbf{T} - \mathbf{Nil}) 0$$

$$(\mathbf{T} - \mathbf{Par}) \frac{\vdash P \quad \vdash Q}{\vdash P \mid Q}$$

$$(\mathbf{T} - \mathbf{Sum}) \frac{\vdash P \quad \vdash Q}{\vdash P + Q}$$

$$(\mathbf{T} - \mathbf{Res}) \frac{a : L \quad \vdash P}{\vdash (\nu a)P}$$

Seule la règle pour typer un processus répliqué change au gré des systèmes de types que nous avons considérés, car c'est elle qui capture l'analyse faite par le système de types. Pour le Système 1, cette règle est $(lvl(n))$ désigne le niveau affecté au nom n):

$$(\mathbf{T} - \mathbf{Ter}) \frac{\Gamma \vdash a.Q \quad \forall \bar{n} \in \text{fe}(Q), lvl(a) > lvl(n)}{\Gamma \vdash !a(x).Q}$$

Un processus répliqué est bien typé si chaque émission libre \bar{n} de la continuation Q (les émissions libres de Q sont notées $\text{fe}(Q)$) a un niveau inférieur à celui de la réception a – rappelons qu'une émission est dite *libre* si elle n'apparaît pas sous une réplification: \bar{b} est libre dans $a.(\bar{b}|\bar{c})$, mais ne l'est pas dans $a.!c.\bar{b}$. Les systèmes 2, 3 et 4 sont obtenus à partir du Système 1 en modifiant la règle $\mathbf{T} - \mathbf{Ter}$ (cf. plus bas).

3 Inférences des premiers Systèmes de types

3.1 Inférence du Système 1

La question de l'inférence est, étant donné un processus du π -calcul, de savoir s'il est typable (auquel cas il est terminant) ou pas (auquel cas, on ne peut conclure). Pour typer un terme avec le Système 1, on doit affecter un niveau à chacun des noms syntaxiquement présents dans le terme. Comme c'est la comparaison entre deux niveaux qui compte, non les valeurs numériques des niveaux, on s'aperçoit que l'on peut se restreindre au cas où il y a moins de niveaux que de noms.

Une première démarche exhaustive se dessine donc pour l'inférence: si n noms sont en présence, essayer les n^n affectations possibles de niveaux au noms, si le terme n'est typable par aucune d'entre elles, il n'est pas typable pour ce système, s'il est typable pour une affectation, alors il est terminant. Cette méthode de force brute est peu satisfaisante.

Proposition 1 *Le problème d'affectation de niveau pour la 1-typabilité est dans P .*

Le problème peut être résolu en construisant un graphe orienté $G = (V, E)$ où V est l'ensemble des noms du programme et une arête (a, b) est dans E si le processus contient comme sous-processus $!a.P$ avec b émission libre dans P . Ainsi une arête de a à b caractérise la nécessité que le niveau de a soit plus grand que celui de b .

On prouve que l'existence d'une affectation de niveau permettant le typage de toutes les réplifications du terme équivaut à l'absence de cycle dans G . Un parcours en profondeur permet de décider l'existence de cycle en temps polynomial. La typabilité d'un terme par ce système est donc polynomiale.

Le raisonnement que l'on vient de faire vaut pour CCS. Dans le cas du π -calcul, il faut en plus s'atteler à identifier les niveaux des noms qui peuvent être utilisés de la même façon, mais l'inférence reste polynomiale.

L'article [13] propose un raffinement de ce système de types, pour le π -calcul, où l'on autorise les répliquations $!p(n).P$ quand tous les noms de P ont un niveau strictement plus petit que celui de p sauf éventuellement (la différence est là) une ou plusieurs émissions $\bar{p}\langle k \rangle$ telles que $k < n$ pour un certain ordre bien fondé. Ainsi l'expression en π -calcul de l'opérateur factoriel devient typable. Une preuve que toute fonction primitive récursive admet un encodage typable dans ce système est donnée. Aucun travail sur le système 2 n'a été fait au cours du stage, si ce n'est s'assurer qu'il pouvait être facilement étendu pour coder la fonction d'Ackermann.

3.2 Le Système 3

Le troisième système de types présenté dans [13] est une extension du premier, on affecte toujours un niveau à chaque nom. On ne se contente plus de comparer la réception préfixe d'une répliquaion aux émissions libres de la continuation. On compare la séquence maximale préfixe de réceptions aux émissions libres.

C'est à dire que les répliquations qui deviennent typables sont celles du type $!\kappa.Q$ où κ est une séquence de réceptions et telles que le vecteur poids v_i associé à cette séquence soit strictement plus grand pour l'ordre lexicographique que le vecteur poids v_o associé aux émissions libres de Q .

Par exemple on peut typer $!p.a.(\bar{p} \mid \bar{b})$ avec p de niveau 3, a de niveau 2, b de niveau 1. La séquence maximale de réceptions κ est $p.a$ de vecteur poids $[1, 1, 0]$ et le vecteur poids des émissions libres $\{p, b\}$ est $[1, 0, 1]$, plus petit.

Le Système 3 accepte davantage de processus terminants que le Système 1: ainsi, le processus précédent est rejeté par le Système 1, car l'émission sur p conduit à l'inégalité $lvl(p) > lvl(p)$.

3.3 Inférence du Système 3

Proposition 2 *L'inférence du Système 3 est NP-complète.*

L'inférence du Système 3 consiste à trouver une affectation de niveau valide si elle existe. Nous avons établi la NP-complétude de ce problème. Pour cela, nous avons fait appel à Alain Darté (de l'équipe Compsys du LIP), qui nous a aidé à trouver le problème depuis lequel faire la réduction.

On montre ainsi que 1IN3-SAT, le problème de satisfiabilité d'une formule booléenne de clause de trois littéraux dont un littéral exactement doit être vrai dans chaque clause, se réduit polynomialement à notre problème.

Si on a une instance I de 1IN3-SAT on construit I' une instance de notre problème d'affectation de niveau:

- En créant deux noms x_k et x'_k pour chaque variable propositionnelle v_k
- En créant un nom supplémentaire T
- En créant un gadget $!T.T.\overline{x_k.x'_k} \mid !x_k.x'_k.T$ pour chaque variable v_k , les contraintes engendrées par le Système 3 impliquent que, pour typer ce gadget, il faut qu'une variable exactement soit au même niveau que T parmi x_k et x'_k . On interprète ' x_k a le même niveau que T ' comme ' v_k est vraie'.
- En créant un gadget $!n_{i_1}.n_{i_2}.n_{i_3}.\overline{T} \mid !T.T.\overline{n_{i_1}.n_{i_2}.n_{i_3}}$ pour chaque clause $C_i = \{l_{i_1}, l_{i_2}, l_{i_3}\}$ où n_{i_j} est x_k si l_{i_j} est v_k et x'_k si l_{i_j} est $\neg v_k$; les contraintes du Système 3 impliquent que, pour typer ce gadget, il faut qu'une variable exactement soit au niveau de T parmi n_{i_1} , n_{i_2} et n_{i_3} . On interprète ' c 'est n_{i_j} qui est au même niveau que T ' par ' c 'est l_{i_j} qui est vraie'.

On montre ensuite l'équivalence entre l'existence de solution pour I et I' (cf. [3, Section 6.1]).

Implémenter le Système 3. La NP-complétude du troisième système de types de [13] nous pousse à chercher une solution plus acceptable pour l'implémentation. Parallèlement au stage, Naoki Kobayashi a travaillé sur son outil pour l'analyse automatique de processus du π -calcul [7]. Vers la fin de la période du stage, N. Kobayashi a proposé une variante du Système 3, où les niveaux disparaissent, et où chaque nom se voit attribuer un poids entier. Les inégalités produites par le système de types sont ensuite résolues par programmation linéaire en rationnels, puis retransformées en entiers (on peut se le permettre car aucune constante n'est impliquée).

Au vu de la Proposition 2, cette variante présente un avantage certain par rapport au Système 3: le système de N. Kobayashi est en effet au moins aussi expressif que le Système 3, et l'inférence en est polynômiale.

Dans le cas du π -calcul, une difficulté supplémentaire apparaît dans le sens où il faut pouvoir identifier les noms qui ont même usage. En effet si le programme contient une liaison par p de la variable a : $p(a)$ et une émission de u sur p : $\bar{p}(u)$, alors une contrainte de niveau sur a implique la même contrainte sur u car u peut être utilisé à la place de a , ainsi u et a doivent avoir même niveau. Une solution draconienne proposée par Davide Sangiorgi est de supposer que les noms de même type simple ont même niveau. Au cours du stage, un algorithme pour calculer effectivement la relation d'équivalence 'avoir même usage que' a été développé.

4 Affiner l'analyse

4.1 Le principe des "pantalons"

Une contrainte forte qui pèse sur les systèmes 3 (et 4) est l'absence de 'descente' au sein de la réplication. Dans le premier système de types, seule la première réception est comparée à toutes les émissions dans le reste du processus. Dans le troisième système seule la première séquence de réception est comparée à toutes les émissions dans le reste du processus.

Cela restreint la forme des programmes pouvant être acceptés par un système de types: considérons le processus $P = !p.a.(\bar{b} \mid \bar{p})$, il peut être 3-typé avec l'affectation de niveau qui envoie p et b sur 1 et a sur 2 car on compare la suite de réceptions $p.a$ aux émissions \bar{b}, \bar{p} . Maintenant considérons la variante $Q = !p.(a.\bar{b} \mid a.\bar{p})$ où la réception a est distribuée dans chaque composante parallèle, cette réplication n'est pas plus 'dangereuse' pour la terminaison que la précédente, au sens où, pour produire les mêmes émissions, on doit consommer au moins autant de réceptions. Pourtant le système 3 ne peut pas typer ce processus, car la séquence maximale de réception que l'on peut considérer est p , et il faut la comparer à toutes les émissions du reste du processus, c'est à dire p et b .

On veut maintenant définir un système de types similaire à ceux présentés précédemment, mais où chaque manière de consommer le processus est traitée séparément, et où les émissions qui entrent en compte dans une comparaison sont directement accessibles, c'est-à-dire non gardées par des réceptions. Chaque réplication donne lieu à plusieurs inégalités fines au lieu d'une seule inégalité grossière.

Enumérons les différentes manières de consommer Q :

- Si l'unique réception faite est p , alors aucune émission n'est directement disponible car \bar{p} et \bar{b} sont toutes les deux derrière une réception a . L'inégalité associée à cette manière de consommer la réplication est $p > 0$ qui est triviale.
- Si on réceptionne p , puis a dans la première composante, on libère une émission sur b . La comparaison p, a contre b peut être validée par l'affectation d'un niveau 2 à a et 1 à b .
- Réciproquement, si on réceptionne p , puis a dans la deuxième composante, on libère une émission p . L'inégalité $p + a > p$ est triviale.
- Enfin, avec les réceptions p, a et a encore on libère les deux émissions et la comparaison devient $a + a + p > b + p$ qui est compatible avec l'affectation faite plus haut

On appellera communément 'pantalon' un processus répliqué et 'sous-pantalon' une manière de consommer ce pantalon. On ne considère comme pertinent que certains sous-pantalons: on prend les séquences de

réception dans leur entier. Par exemple dans le pantalon $!p.a.\bar{b}$, le sous-pantalon résultant d'une unique réception p , qui ne libère aucune émission, n'est pas pertinent (il ne libère pas plus d'émission que le sous-pantalon vide), seul le sous-pantalon résultant de la séquence de réception $p.a$ l'est.

Formellement, l'ensemble des sous-pantalons d'un sous processus est défini par:

- $\mathbf{Part}(\mathbf{0}) \stackrel{\text{def}}{=} \{\mathbf{0}\}$
- $\mathbf{Part}(P|Q) \stackrel{\text{def}}{=} \{A|B. A \in \mathbf{Part}(P), B \in \mathbf{Part}(Q)\}$
- $\mathbf{Part}(\kappa.P) \stackrel{\text{def}}{=} \{\mathbf{0}\} \cup \{\kappa.A. A \in \mathbf{Part}(P)\}$, où κ est la séquence préfixe de réception maximale dans $\kappa.P$
- $\mathbf{Part}(\rho.P) \stackrel{\text{def}}{=} \{\rho.A. A \in \mathbf{Part}(P)\}$, où ρ est la séquence préfixe d'émission maximale dans $\rho.P$
- $\mathbf{Part}(P + Q) \stackrel{\text{def}}{=} \mathbf{Part}(P) \cup \mathbf{Part}(Q)$
- $\mathbf{Part}(\text{if } b \text{ then } P \text{ else } Q) \stackrel{\text{def}}{=} \mathbf{Part}(P) \cup \mathbf{Part}(Q)$

L'ensemble des notation formelles associées ainsi qu'une preuve de correction de la définition se trouve dans [3, Section 2.2]

Il faut noter que:

Lemme 3 *Le système de types 3 avec pantalons est strictement plus expressif que le système 3 sans pantalons.*

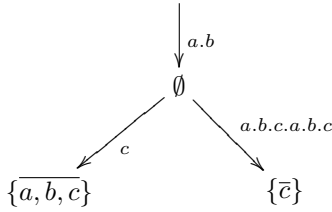
L'exemple Q montre que l'on type des processus avec les pantalons que l'on ne typait pas sans. Inversement, si un processus est typable avec le Système 3, on peut noter $I > O$ l'inégalité associée (qui compare des multi-ensembles de noms) et montrer que pour chaque nouvelle inégalité $I' > O'$ associée à un sous-pantalon, on a $O' \subseteq O$ et $I \subseteq I'$, impliquant que si $I > O$ est satisfaite, chaque $I' > O'$ le sera.

4.2 Un Exemple de Décomposition en Sous-pantalons

Une manière de visualiser les sous-pantalons d'un pantalon est de l'écrire sous forme arborescente, chaque arête représentant une séquence de réceptions, et chaque noeud de l'arbre énumérant les émissions qui sont libérées si on consomme les réceptions étiquetant le chemin de la racine à ce noeud.

Soit $P = !a.b.(c.\bar{a}.\bar{b}.\bar{c} \mid a.b.c.a.b.c.\bar{c})$

La vision arborescente de P est:

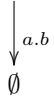


Les sous-pantalons associés à P sont:

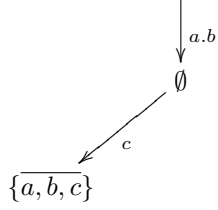
1. $A_1 = a.b$
2. $A_2 = a.b.c.\bar{a}.\bar{b}.\bar{c}$
3. $A_3 = a.a.b.c.a.b.c.\bar{c}$
4. $A_4 = a.b.(c.\bar{a}.\bar{b}.\bar{c} \mid a.b.c.a.b.c.\bar{c})$

correspondant aux branches de l'arbre:

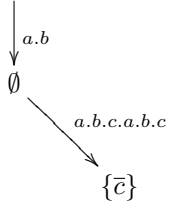
1.



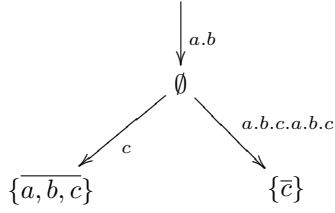
2.



3.



4.



Et les inégalités de noms associées sont

1. $a + b > 0$

2. $a + b + c > a + b + c$

3. $a + a + a + b + b + b + c + c > c$

4. $a + a + a + b + b + b + c + c + c > a + b + c + c$

Comme on vérifie qu'il n'ya pas de branches qui peut produire plus que ce qu'elle consomme, le second sous-pantalon montre que P n'est pas typable. $Q = P|\bar{a}|\bar{b}|\bar{c}$ n'est d'ailleurs pas terminant, car $P \Rightarrow^a \Rightarrow^b \Rightarrow^c P$

4.3 Prouver la terminaison du nouveau système de types

Un nouveau système de types, appelé Système 3 avec pantalons, est obtenu en remplaçant l'ancienne règle de typage de la réplication par une nouvelle règle:

$$(\mathbf{T} - \mathbf{Ter}) \frac{\Gamma \vdash a(\tilde{x}).Q \quad \forall A \not\parallel a(\tilde{x}).Q, wt(I(A)) > wt(O(A))}{\Gamma \vdash !a(\tilde{x}).Q}$$

Où $\forall A \not\parallel a(\tilde{x}).Q, wt(I(A)) > wt(O(A))$ est l'écriture formelle de 'Pour tout sous-pantalon A du pantalons associé à la réplication $!a(\tilde{x}).Q$, les réceptions consommées $I(A)$ pèsent plus lourd que les émissions directement accessibles $O(A)$ ' (cette notation a été utilisée informellement plus haut dans la preuve du Lemme 3).

Proposition 4 *Le Système de types 3 avec pantalons assure la terminaison.*

La preuve de terminaison du système 3 reposait, sous certaines contraintes sur les séquences de réception, sur la commutativité des transitions dans une réduction infinie. Ces contraintes, plutôt lourdes, étaient conjecturées inutiles par Davide Sangiorgi.

La nouvelle preuve de la terminaison du système 3 ([3, Section 3]), sans ces contraintes, a ouvert la voie à la preuve du système 3 avec pantalons. L'idée est d'associer un opérateur σ_A à chaque sous-pantalons A . Cet opérateur retranche le poids des réceptions consommées de A et ajoute le poids des émissions directement accessibles. Ainsi, si le processus est typé, chaque opérateur σ_A est strictement décroissant pour le poids.

On considère ensuite une dérivation infinie d'un processus typé. On associe à chaque étape i de la dérivation une table **SEQ** (i) qui garde en mémoire l'état de chaque pantalon que l'on a commencé à consommer. Un opérateur global $\Sigma(i)$ est défini comme la composition des opérateurs σ_A pour tous les A présents dans **SEQ** (i).

L'étape suivante est de particulariser dans la dérivation infinie, les pantalons qui seront consommés une infinité de fois (il en existe forcément). On construit les ensembles $\mathcal{B} = \{A \mid \forall n, \exists j \in \mathbb{N}. A \text{ apparaît } n \text{ fois dans } \mathbf{SEQ}(j)\}$ et $\mathcal{B}' = \{\sigma_A \mid A \in \mathcal{B}\}$. On considère ensuite le niveau maximum m qu'un sous-pantalons de cet ensemble fait décroître. On montre que la composante de $\Sigma(i)$ sur ce niveau m croît arbitrairement quand i croît.

Il ne reste plus qu'à lier cette valeur abstraite à une valeur concrète: le nombre d'émissions de niveau m contenues dans le processus. Cette valeur ne peut être négative. On aboutit à une contradiction, il ne peut donc exister de dérivation infinie d'un processus bien typé.

5 Des Systèmes plus expressifs

5.1 Le Système 4

S'il est plus large que le premier système de types, le système 3 implique la 'perte' d'un nom à chaque utilisation d'une réplication. Le vecteur d'entrée doit être strictement plus grand que le vecteur de sortie. Cela motive la création d'un 4ème système de types qui englobe le précédent, où l'on autorise, sous certaines conditions, le poids à ne pas décroître lors d'une réplication.

Ce qui va assurer la terminaison est un ordre partiel sur les noms de même type. Cette amélioration n'a donc de sens que pour le π -calcul. On peut désormais comparer deux noms qui pèsent le même poids a et b , en écrivant que $a \mathcal{R} b$ (où \mathcal{R} est une relation d'ordre partiel). Un élément d'une liste chaînée pourra donc être en relation avec son successeur, rendant possible la propagation d'une recherche dans le sens descendant.

L'ordre partiel sur les noms étant composante des types, on ne peut comparer des noms que s'ils sont arguments d'une même émission ou d'une même réception, l'ordre étant attaché au sujet de cette action. Par exemple, si on veut qu'un élément d'une liste a soit désigné comme prédécesseur de b , on peut avoir un constructeur p binaire dont le type indique que le premier argument domine pour l'ordre le deuxième, et avoir quelque part dans le processus $p(a, b)$.

5.2 Appliquer l'analyse des pantalons aux Système 4

Le Système de types 4, pour le π -calcul, est très similaire au système 3, mais ajoute la possibilité de comparer des noms d'un unique niveau l . Il devient naturel de lui appliquer le principe de l'exploration des sous-pantalons décrite à la Section 4. Néanmoins, comme on type maintenant des sous-pantalons qui consomment, au niveau du poids, autant que ce qu'ils produisent, il faut ajouter des contraintes sur la règle de typage.

Rappelons qu'en π -calcul, les noms transportent d'autres noms et de nouveaux noms peuvent être créés dynamiquement dans des processus puis ensuite envoyés plus loin que la portée de ce processus. Cela peut engendrer des effets pervers sur le typage. Par exemple considérons le processus $P = !p(a, b).a.(b \mid (\nu c)\bar{p}(b, c))$.

Le typer avec $a \mathcal{R} b$ semble possible. Mais ce processus est dangereux car il peut engendrer une dérivation infinie en créant à chaque étape un nouveau nom c_{i+1} plus petit que le c_i précédent, ainsi $(P \mid \bar{p}\langle u, v \rangle \mid \bar{u}) \rightarrow^{\tau_p} \rightarrow^{\tau_u}$
 $P \mid (\nu c_1)\bar{p}\langle v, c_1 \rangle \rightarrow^{\tau_p} \rightarrow^{\tau_u} P \mid (\nu c_2)\bar{p}\langle c_1, c_2 \rangle \rightarrow^{\tau_p} \rightarrow^{\tau_u} \dots$

Par ailleurs, le système 4 tel que le proposent Sangiorgi et Deng autorise, dans le typage d'un programme, un unique niveau à contenir l'ordre partiel. S'il n'est pas possible de comparer des noms de types différents, on s'attend toutefois à pouvoir comparer entre eux n'importe quel couple de noms de même niveau. L'ordre \mathcal{R} devient ainsi un ordre feuilleté. On définit ensuite $\bar{\mathcal{R}}$ l'extension lexicographique par niveau de l'extension multi-ensemble de cet ordre. On a besoin de prouver (voir [3, Section 4.1]):

Lemme 5 *Sur un support de noms fini, $\bar{\mathcal{R}}$ est bien-fondé.*

pour accéder à la terminaison du nouveau système:

Proposition 6 *Le Système de types 4 avec pantalons et feuilletage de l'ordre assure la terminaison.*

La preuve diffère de la précédente, dans le sens où les opérateurs σ_A ne sont pas tous strictement décroissants (il est permis de consommer, en poids, autant que ce que l'on produit). L'on doit d'abord prouver le lemme:

Lemme 7 *\mathcal{B}' contient au moins un opérateur strictement décroissant.*

(\mathcal{B}' est défini comme à la fin de 4.3).

En effet, si les seuls sous-pantalons présents un nombre arbitrairement grand de fois dans la dérivation infinie sont typés par l'utilisation de $\bar{\mathcal{R}}$, le caractère bien fondé de cet ordre aboutit à une contradiction.

Donc il existe un opérateur strictement décroissant dans \mathcal{B}' et on peut appliquer le même raisonnement que précédemment.

5.3 Un Exemple et sa décomposition en Sous-pantalons

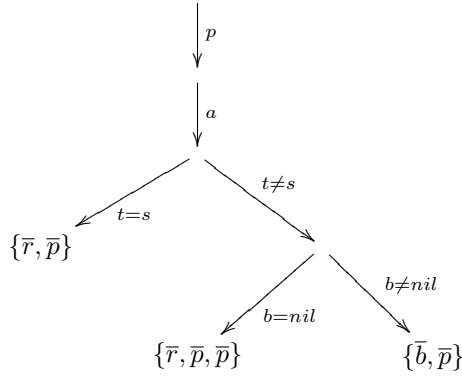
L'exemple choisi par [13] pour confronter l'analyse de type à un exemple de structure de données concurrentes est la table de symboles (*symbol table*). Son code en π -calcul est:

$$\begin{aligned}
 G = & !p(a, b, n, s).a(t, r). \\
 & \text{if } t = s \text{ then} \\
 & \quad \bar{r}\langle n \rangle.\bar{p}\langle a, b, n, s \rangle \\
 & \text{else if } b = \text{nil} \text{ then} \\
 & \quad \bar{r}\langle n + 1 \rangle.(\nu c)(\bar{p}\langle c, \text{nil}, n + 1, t \rangle \mid \bar{p}\langle a, c, n, s \rangle) \\
 & \quad \text{else } \bar{b}\langle t, r \rangle.\bar{p}\langle a, b, n, s \rangle
 \end{aligned}$$

Ici un serveur p crée et modifie les cases de la liste. Lorsqu'une requête $a(t, r)$ est reçue, on compare la valeur du noeud s avec celle de la requête t . Si elles sont identiques, on a trouvé, et on envoie le numéro n de la case sur le canal de réponse r , et on recrée la case sur p . Sinon si on est en fin de liste on ajoute une case: un appel à p supplémentaire et on envoie le numéro de cette nouvelle case $n + 1$. Sinon on doit passer la requête à son successeur b et se relancer sur p . Le caractère concurrent apparaît dans la possibilité d'avoir plusieurs requêtes parcourant simultanément la liste.

On remarque dans les deux premiers cas, en comparant ce que l'on récupère comme émission après avoir reçu a et p , que le canal a doit être le canal de poids fort. Le successeur de a dans la liste b a un type similaire, il a donc le même poids. Seul ce système de types permet de typer cet exemple car dans le dernier cas de la structure conditionnelle, on consomme un constructeur p et une requête a pour un constructeur p et une requête b . L'ordre partiel permet de valider cette réplification, en imposant que l'on doit, dans ce cas, émettre une requête sur un élément plus loin dans la liste. L'acyclicité de la liste ainsi que son caractère fini implique donc la terminaison de ce programme.

En appliquant la visualisation arborescente des réplifications on peut écrire cet exemple sous la forme:



Cela nous donne les inégalités suivantes.

1. $\{p, a\} \succ \{p, r\}$
2. $\{p, a\} \succ \{p, p, r\}$
3. $\{p, a\} \succ \{p, b\}$

Pour typer ce programme, nous devons garder à l'esprit que a et b jouent le même rôle (premier argument de p) et donc doivent avoir le même niveau.

Une affectation de niveau qui convient est:

- $a, b \rightarrow 2$
- $p \rightarrow 1$
- $r \rightarrow 1$

et en ajoutant la relation d'ordre $a \mathcal{R} b$ on résout les inégalités:

1. $[1, 1] > [0, 2]$
2. $[1, 1] > [0, 3]$
3. $[1, 1] = [1, 1]$ et $\{a, p\} \bar{\mathcal{R}} \{b, p\}$

Et ce programme est typable, donc terminant.

5.4 Structure de Données Arborescente et Système 5

On peut regretter que si le système 4 peut typer une liste simplement chaînée, en considérant chaque maillon comme plus bas dans l'ordre que son prédécesseur et plus haut que son successeur, il ne peut pas typer un arbre, où chaque noeud a plusieurs 'successeurs'. Cela est dû au fait que si l'on propage une requête de recherche sur un tel arbre aux deux fils d'un noeud, le poids des émissions correspondantes est strictement supérieur à celui des réceptions (alors qu'il était égal dans le cas du système 4). Notons qu'un *arbre binaire de recherche* est typable avec le système 4 uniquement car la requête sur un noeud n'est propagée que sur un seul des deux fils.

Une première étape dans cette direction est l'écriture en π -calcul d'une telle structure de données. Le code ne peut pas être adapté directement depuis celui de la *symbol table*. Comme noté précédemment, l'avantage du codage dans un langage concurrent d'une telle structure réside dans le fait que plusieurs requêtes peuvent parcourir en même temps la structure. Cela pose problème dans le cas de l'arbre: que faire si la requête

arrive au bout d'une branche sans avoir trouvé la valeur qu'elle cherchait, si elle n'attend pas que les autres composantes concurrentes de la même requête qui parcourent d'autres branches arrivent elles aussi à une feuille? Si plusieurs requêtes sur la même valeur arrivent au serveur en même temps et que cette valeur n'est pas présente dans la structure, on veut s'assurer que l'on ajoute exactement une fois cette valeur dans l'arbre.

Ainsi le code de la structure comprend:

- La structure arborescente elle-même dont les noeuds peuvent être activés selon deux modes: 'ajout' ou 'recherche'.
- La structure de contrôle qui reçoit les requêtes de l'utilisateur et active la recherche, l'insertion ou la réponse.
- Une liste de listes gardant en mémoire les requêtes en train d'être traitées et les canaux de réponses associés.

Lorsque la structure de contrôle reçoit une requête sur la valeur v avec le canal de réponse r , elle parcourt sa liste de requêtes en cours. Si la valeur v est déjà en train d'être traitée, on ajoute juste le canal r à la liste de canaux de réponse pour cette valeur. Sinon on crée une nouvelle entrée dans la liste et on lance une recherche dans l'arbre. Chaque noeud compare sa valeur s avec v , si elles sont différentes, la requête est propagée sur les deux fils du noeud et on attend une réponse d'eux. Si la valeur est trouvée, la réponse renvoyée au père est l'adresse du noeud, et une feuille renvoie le mot vide. Ainsi chaque père concatène les réponses reçues de chacun de ses deux fils. La racine de l'arbre renvoie à la structure de contrôle l'adresse du noeud qui contient v s'il existe, le mot vide sinon. Dans le premier cas cette adresse est envoyée sur chaque canal de réponse associé. Dans le deuxième cas une requête d'ajout est envoyée sur l'arbre, cette requête se propage de manière non-déterministe sur le fils gauche ou droit de chaque noeud et est ajoutée dès qu'une place se présente, la nouvelle adresse du noeud créé est ensuite envoyée à tous les canaux de réponse associés. Dans les deux cas, la valeur v est retirée de la liste de valeurs en train d'être traitée.

On présente ici uniquement le code de la structure arborescente et sa représentation (voir [3, Section 6.2] pour le code de l'ensemble des termes):

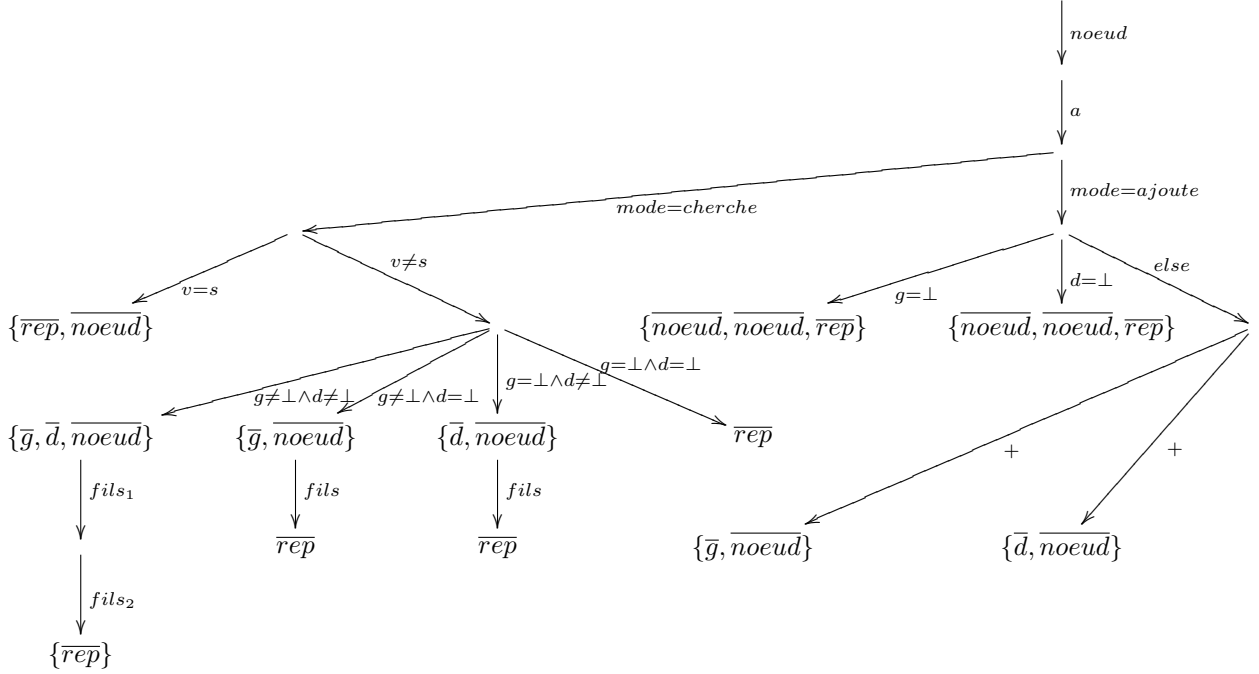
```

ARBRE =
!noeud(a, g, d, e, s)
  a(mode, v, rep).
  if mode = 'cherche'
    if v = s then
       $\overline{rep}(e) \mid \overline{noeud}(a, g, d, e, s)$ 
    else
      if  $g \neq nil \wedge d \neq nil$  then
         $(\nu fils_1)(\nu fils_2)(\overline{g}('cherche', v, fils_1) \mid \overline{d}('cherche', v, fils_2) \mid$ 
         $\overline{noeud}(a, g, d, e, s) \mid fils_1(x).fils_2(y).\overline{rep}(x@y))$ 
      elseif  $g \neq nil \wedge d = nil$ 
         $(\nu fils)(\overline{g}('cherche', v, fils) \mid \overline{noeud}(a, g, d, e, s) \mid fils(x).\overline{rep}(x))$ 
      elseif  $g = nil \wedge d \neq nil$ 
         $(\nu fils)(\overline{d}('cherche', v, fils) \mid \overline{noeud}(a, g, d, e, s) \mid fils(x).\overline{rep}(y))$ 
      else
         $\overline{rep}(\epsilon)$ 
    elseif mode = 'ajoute'
      if  $g = nil$ 
         $(\nu b)\overline{noeud}(a, b, d, e, s).\overline{noeud}(b, nil, nil, e@1, v).\overline{rep}(e@1)$ 
      elseif  $d = nil$ 

```

else
 $(vb)\overline{noeud}\langle a, g, b, e, s \rangle.\overline{noeud}\langle b, nil, nil, e@2, v \rangle.\overline{rep}\langle e@2 \rangle$
 $\bar{g}\langle 'ajoute', v, rep \rangle + \bar{d}\langle 'ajoute', v, rep \rangle$

Dans la dernière ligne, la construction $P + Q$ (qui est un opérateur standard en CCS et en π -calcul) sert à programmer le choix non-déterministe pour l'ajout de v .



La branche qui nous intéresse ici est celle du mode recherche, quand la valeur cherchée est différente de celle du noeud et que le noeud a deux fils ($mode = cherche$, $v \neq s$ et $g \neq \perp \wedge d \neq \perp$), en rouge dans le texte. Typier cette branche conduit à la comparaison $\{noeud, a\} \succ \{noeud, g, d\}$. Sachant que a, g et d ont même type, et donc même niveau, on est conduit à devoir indiquer que $a \mathcal{R} (g, d)$ et que cette descente dans l'ordre compense le fait que le poids augmente.

Le Système 5 Le système 5 autorise les sous-pantalons à consommer moins que ce qu'ils produisent sous un certain nombre de conditions. Pour typer une réplication on examine chaque sous-pantalon, et on compare les multi-ensembles associés d'émissions et de réceptions niveau par niveau en commençant par le plus haut. Au premier niveau où les multiensembles de réceptions et d'émissions sont différents, soit le poids (donc le cardinal), soit l'ordre doit décroître. Il y a là aussi, de nombreuses conditions sur la création de nouveaux noms (se référer à [3, Section 5] pour les détails techniques). On prouve ensuite:

Proposition 8 *Le Système 5 assure la terminaison.*

La preuve diffère largement des précédentes, car il est beaucoup plus difficile de montrer qu'un niveau admet une sous-suite strictement décroissante en poids, étant donné qu'on autorise parfois le poids à croître.

Encore une fois, les ensembles \mathcal{B} et \mathcal{B}' sont définis comme dans la preuve de la proposition 4.3. La preuve de terminaison repose sur le lemme suivant:

Lemme 9 *S'il existe un sous-pantalon A dans \mathcal{B} qui est typé par l'utilisation de l'ordre au niveau l , alors il existe un sous-pantalon dans A' dans \mathcal{B}' qui est typé par une décroissance de poids sur un niveau $l' > l$.*

On prouve ce lemme en supposant qu'il n'existe pas d'autre sous-pantalon dans \mathcal{B} que A tel que les sous-multi-ensembles de niveau l d'émissions et de réceptions sont différents. On aboutit à une contradiction. Il existe donc un autre sous-pantalon A'' qui 'change' le niveau l , on discute alors sur le niveau qui permet de typer ce sous-pantalon. On déduit finalement l'existence d'un sous-pantalon A' répondant aux conditions du lemme. On peut ensuite raisonner comme pour la preuve du système 3 avec pantalons.

6 Développements récents

A l'occasion d'une visite effectuée début juin chez Davide Sangiorgi à l'Université de Bologne pour exposer ces résultats, nous avons pu dégager de nouvelles extensions des système de types existants sur lesquelles nous travaillons actuellement. Ces pistes sont brièvement décrites ici.

6.1 Le Système 4 sans le 3

L'inférence du système 4 étant difficile, mais la possibilité d'émettre sur un canal de réception dans une réplication étant nécessaire pour un minimum d'expressivité, on peut considérer le système de types 4 – 3, où l'on compare la première réception seulement à toutes les émissions de la continuation tout en autorisant cette fois-ci la présence d'émissions du même niveau que la réception, mais plus bas pour un certain ordre \mathcal{R} . On a:

Proposition 10 *L'inférence du système de types 4 – 3 est polynômiale.*

Elle s'apparente à l'inférence du système 1 sauf que l'acyclicité doit être vérifiée dans le graphe quotient par la relation 'avoir le même usage'. Ensuite chaque classe d'équivalence doit pouvoir contenir un ordre partiel \mathcal{R} correct. Cette idée d'algorithme calculant les classes d'équivalence avait déjà été partiellement développée au cours du stage, puis abandonnée en même temps que le système de types qui lui était associé, le système 4 – 3 a permis de raviver cette tentative et de l'adapter à ce système.

6.2 Le Système 6

Dans les systèmes 4 et 4 – 3, pour assurer la correction du typage, l'ordre entre les noms doit être stocké dans les types. Plus précisément, on ne peut comparer deux noms que s'ils sont arguments d'un même sujet.

Il est naturel de s'orienter vers un système 6, à ce jour en train d'être mis au point, qui oublie cette contrainte. On peut librement comparer des noms qui ne sont pas liés au même endroit, et même écrire des contraintes entre un nom de variable (lié) et un nom créé (par une restriction).

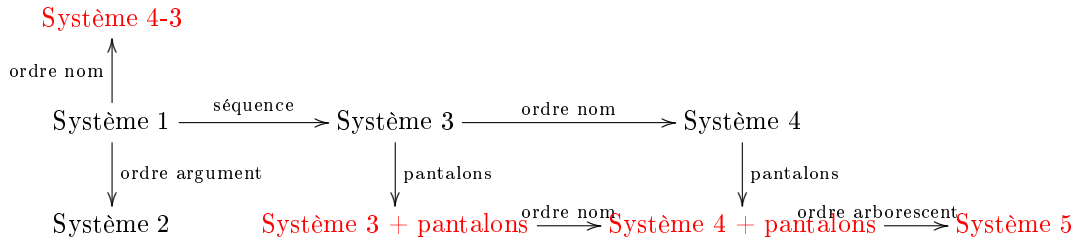
7 Conclusion

Ce stage, imbriqué dans les travaux de Davide Sangiorgi et Naoki Kobayashi, a permis:

- La clarification des preuves des systèmes de types de [13] par l'introduction d'un schéma de preuve réutilisable pour des systèmes de type plus larges.
- La preuve de la difficulté conjecturée des systèmes de type existants.
- Le raffinement de ces systèmes de type par le biais des pantalons, permettant de typer un plus grand nombre de programmes.

- L'écriture d'un nouveau système de types permettant le typage d'une structure de donnée arborescente inédite.

On peut résumer la généalogie des systèmes de types cités dans ce rapport par ce schéma. Les systèmes écrits au cours du stage sont en rouge.



s

References

- [1] Andrey Rybalchenko Byron Cook, Andreas Podelski. Proving thread termination. In *Proc. of PLDI'07*. ACM, 2007. to appear.
- [2] Michael Codish. TerminWeb. available from <http://lvs.cs.bgu.ac.il/~mcodish/suexec/terminweb/bin/terminweb.cgi>, 1999.
- [3] R. Demangeon. Typing termination in the π -calculus. disponible depuis http://perso.ens-lyon.fr/romain.demangeon/draft_rdemangeon.pdf, 2007.
- [4] B. Cook et al. Terminator: proof tools for termination and liveness. available from <http://research.microsoft.com/TERMINATOR/>, 2007.
- [5] Tobias Nipkow Franz Baader. *Term Rewriting and All That*. Cambridge University Press, 1998.
- [6] Yves Lafont Jean-Yves Girard, Paul Taylor. *Proof and Types*. Cambridge University Press, 1989.
- [7] N. Kobayashi. TyPiCal: Type-based static analyzer for the Pi-Calculus. available from <http://www.kb.ecei.tohoku.ac.jp/~koba/typical/>.
- [8] Naoki Kobayashi. Combining type systems for lock freedom and termination. in preparation, 2007.
- [9] Claude Marché. The CiME Rewrite Tool. available from <http://cime.lri.fr/>, 2004.
- [10] Robert Milner. *A Calculus of Communicating Systems*. Springer Verlag, 1980.
- [11] Davide Sangiorgi. Termination of processes. *Mathematical Structures in Computer Science*, 2001.
- [12] Terese. *Term Rewriting Systems*. Cambridge University Press, 2003.
- [13] Davide Sangiorgi Yuxin Deng. Ensuring termination by typability. *Information and Computation*, 204:1045–1082, 2006.