Types intersection et interprétation relationnelle des automates d'arbre

Charles Grellois
travail effectué sous la direction de Paul-André Melliès

PPS

19 septembre 2013

Introduction

- On s'intéresse à un problème de vérification
- Vérification = modèle du programme + formule logique (ou automate équivalent)
- Quelle interprétation sémantique ?

Introduction

- On s'intéresse à un problème de vérification
- Vérification = modèle du programme + formule logique (ou automate équivalent)
- Quelle interprétation sémantique ?

Schémas de récursion d'ordre supérieur

- Modèles de programmes récursifs utilisés depuis les années 60 en vérification
- Correspondent à du λ-calcul avec récursion et constantes
- Informellement : on a un alphabet signé Σ, des non-terminaux, des variables, un axiome et des règles de réécriture paramétrées
- Exemple : $\Sigma = \{a : 2, b : 1, c : 0\}$.

Schémas de récursion d'ordre supérieur

- Modèles de programmes récursifs utilisés depuis les années 60 en vérification
- Correspondent à du λ-calcul avec récursion et constantes
- Informellement : on a un alphabet signé Σ, des non-terminaux, des variables, un axiome et des règles de réécriture paramétrées
- Exemple : $\Sigma = \{a : 2, b : 1, c : 0\}$.

Schémas de récursion d'ordre supérieur : un exemple

$$\begin{array}{c} S \to F\ c \\ F\ x \to a\ x\ (F\ (b\ x)) \end{array} \quad \text{génère} \quad S$$

Schémas de récursion d'ordre supérieur : un exemple

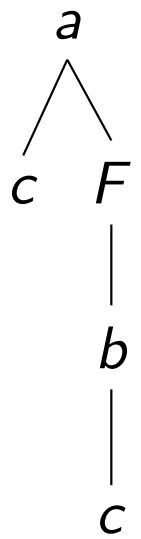
$$S \to F\ c$$
$$F\ x \to a\ x\ (F\ (b\ x))$$

génère

Schémas de récursion d'ordre supérieur : un exemple

$$S \to F\ c$$
$$F\ x \to a\ x\ (F\ (b\ x))$$

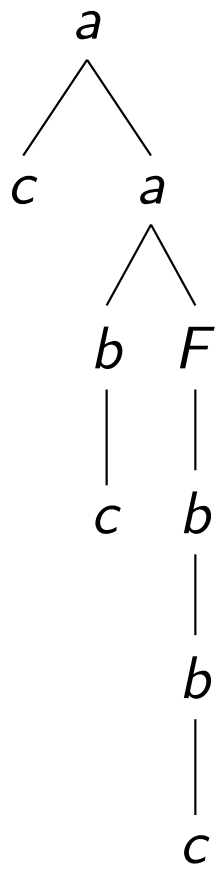
génère



Schémas de récursion d'ordre supérieur : un exemple

$$S \to F\ c$$
$$F\ x \to a\ x\ (F\ (b\ x))$$

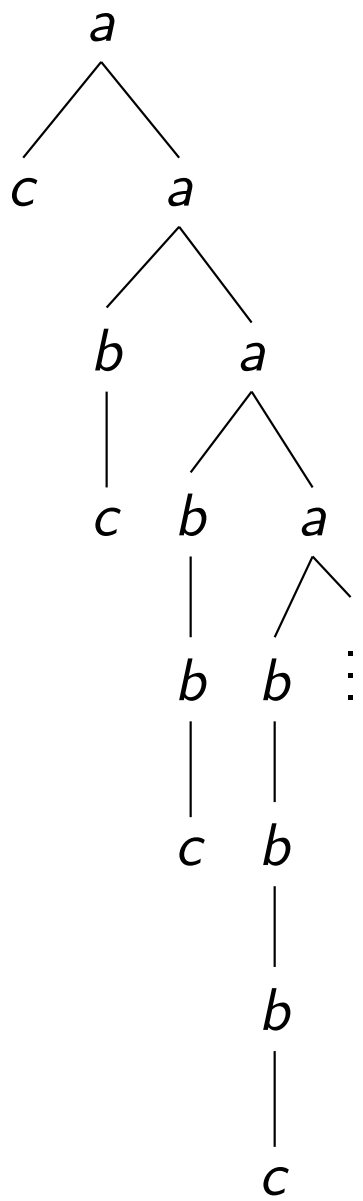
génère



Schémas de récursion d'ordre supérieur : un exemple

$$S \to F\ c$$
$$F\ x \to a\ x\ (F\ (b\ x))$$

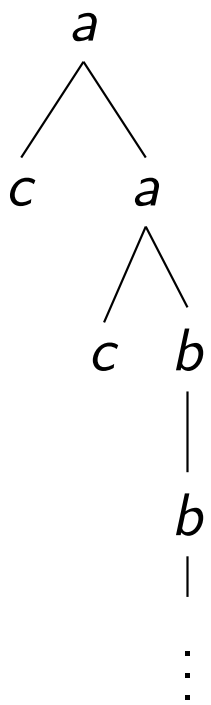
génère



Schémas de récursion d'ordre supérieur : un autre exemple

$$S \to H\ c$$
$$H\ z \to F\ (a\ z)$$
$$F\ \phi \to \phi\ (\phi\ (F\ b))$$

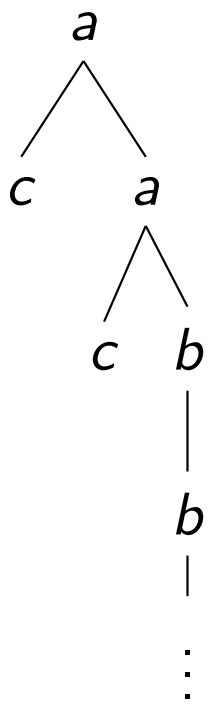
génère



Schémas de récursion d'ordre supérieur et λ-calcul

$$S \to H\ c$$
$$H \to \lambda z.\ F\ (a\ z)$$
$$F \to \lambda \phi.\ \phi\ (\phi\ (F\ b))$$

génère



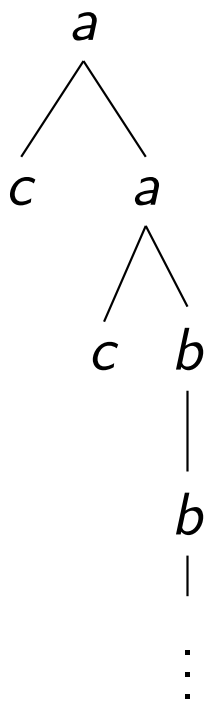
$\to$ λ-termes avec récursion

Typage : dans un premier temps, comme d'habitude avec $a : o \to o \to o$ (o : type des arbres)

Schémas de récursion d'ordre supérieur et λ-calcul

$$S \to H\ c$$
$$H \to \lambda z.\ F\ (a\ z)$$
$$F \to \lambda \phi.\ \phi\ (\phi\ (F\ b))$$

génère



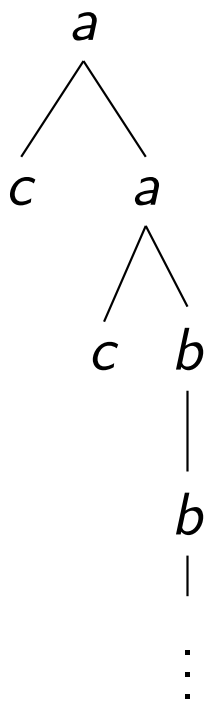
$\to$ λ-termes avec récursion

Typage : dans un premier temps, comme d'habitude avec $a : o \to o \to o$ (o : type des arbres)

Schémas de récursion d'ordre supérieur et λ-calcul

$$S \to H\ c$$
$$H \to \lambda z.\ F\ (a\ z)$$
$$F \to \lambda \phi.\ \phi\ (\phi\ (F\ b))$$

génère



$\to$ λ-termes avec récursion

Typage : dans un premier temps, comme d'habitude avec $a : o \to o \to o$ (o : type des arbres)

Automates d'arbres alternants

- Vérifier les propriétés du μ-calcul sur ces arbres (= comportement des programmes) ?
- On utilise des automates d'arbres alternants.
- Particularité : un *run* sur un arbre n'a pas nécessairement la même structure. . .
- L'automate peut dupliquer ou effacer certains sous-arbres.
- Un comportement qui rappelle le ! de la logique linéaire. . .

Automates d'arbres alternants

- Vérifier les propriétés du μ-calcul sur ces arbres (= comportement des programmes) ?
- On utilise des automates d'arbres alternants.
- Particularité : un *run* sur un arbre n'a pas nécessairement la même structure...
- L'automate peut dupliquer ou effacer certains sous-arbres.
- Un comportement qui rappelle le ! de la logique linéaire...

Automates d'arbres alternants

- Vérifier les propriétés du μ-calcul sur ces arbres (= comportement des programmes) ?
- On utilise des automates d'arbres alternants.
- Particularité : un *run* sur un arbre n'a pas nécessairement la même structure...
- L'automate peut dupliquer ou effacer certains sous-arbres.
- Un comportement qui rappelle le ! de la logique linéaire...

Automates d'arbres alternants

Plus précisément, δ associe à un état et un symbole de Σ une conjonction d'états pour étiqueter chaque fils.

Exemple : $\delta(q_0, a) = (1, q_1) \wedge (2, q_0) \wedge (2, q_2)$
L'automate duplique le fils droit, poursuit dans l'état q_0 sur une copie, dans q_2 sur l'autre, et étiquette q_1 le fils gauche.

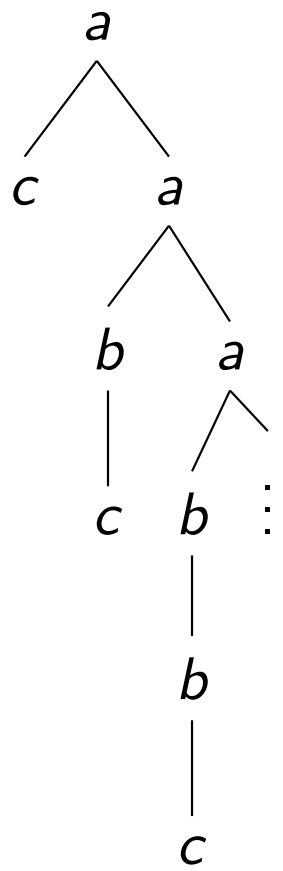
Automates d'arbres alternants

Plus précisément, δ associe à un état et un symbole de Σ une conjonction d'états pour étiqueter chaque fils.

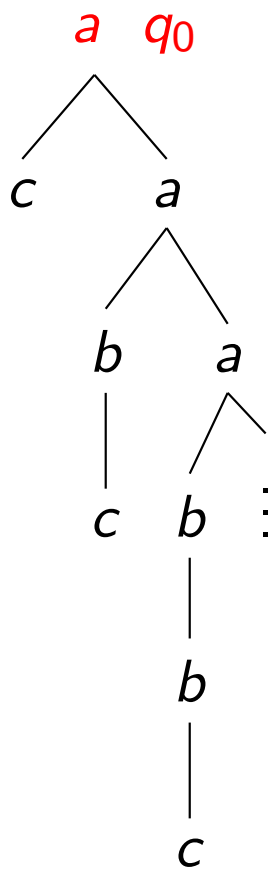
Exemple : $\delta(q_0, a) = (1, q_1) \wedge (2, q_0) \wedge (2, q_2)$
L'automate duplique le fils droit, poursuit dans l'état q_0 sur une copie, dans q_2 sur l'autre, et étiquette q_1 le fils gauche.

Automates d'arbres alternants : exemple d'exécution

On part dans l'état q_0 sur l'arbre suivant :

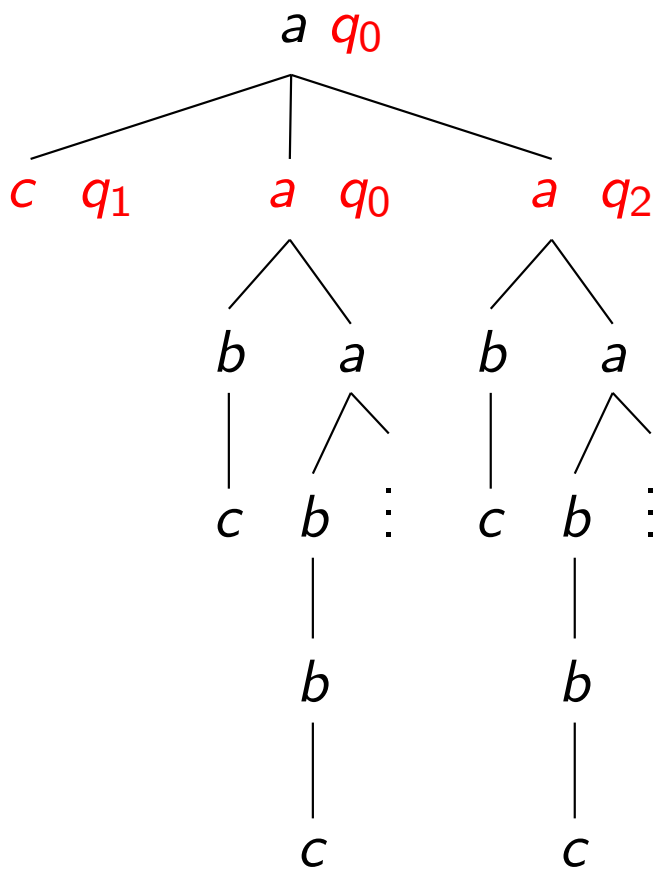


Automates d'arbres alternants : exemple d'exécution



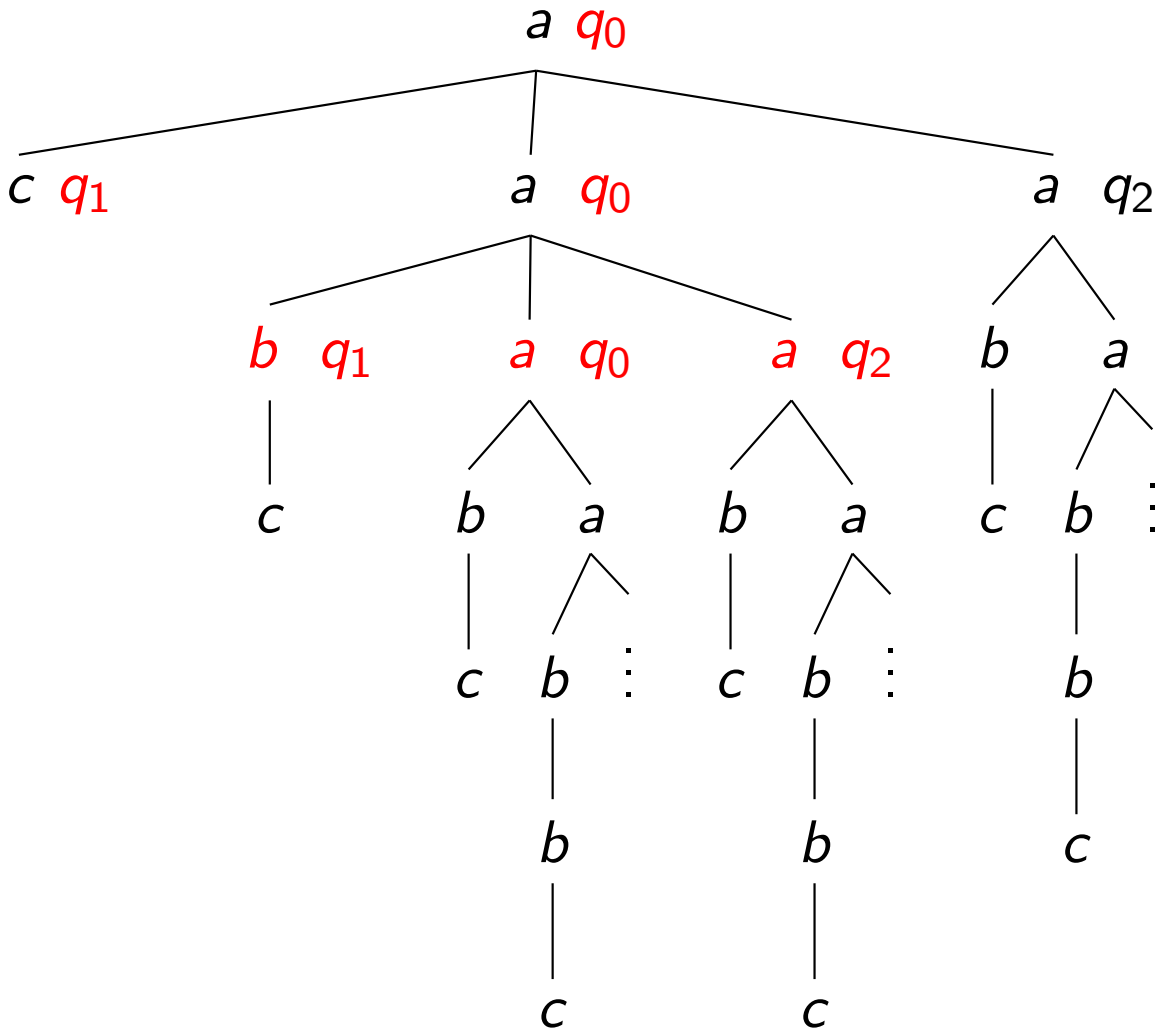
$$\delta(q_0, a) \;=\; (1, q_1) \wedge (2, q_0) \wedge (2, q_2)$$

Automates d'arbres alternants : exemple d'exécution



$$\delta(q_0, a) \;=\; (1, q_1) \wedge (2, q_0) \wedge (2, q_2)$$

Automates d'arbres alternants : exemple d'exécution



$$\delta(q_0, a) \;=\; (1, q_1) \wedge (2, q_0) \wedge (2, q_2)$$

Automates d'arbres alternants et typage

Un autre regard sur ces automates : δ permet de typer les symboles de Σ.

On avait $a : o \to o \to o$.

On aura $a : q_1 \to (q_0 \wedge q_2) \to q_0$.

Automates d'arbres alternants et typage

Un autre regard sur ces automates : δ permet de typer les symboles de Σ.

On avait $a : o \to o \to o$.

On aura $a : q_1 \to (q_0 \wedge q_2) \to q_0$.

Automates d'arbres alternants et typage

Un autre regard sur ces automates : δ permet de typer les symboles de Σ.

On avait $a : o \to o \to o$.

On aura $a : q_1 \to (q_0 \wedge q_2) \to q_0$.

Automates d'arbres alternants et typage

- Kobayashi (2009) : système de typage des règles de réécriture d'un schéma décidant l'acceptation de celui-ci par un automate alternant donné.
- On type directement le schéma qui produit l'arbre, pas celui-ci (qui est infini et dont la structure peut être très complexe).
- La forme des sous-types est contrainte : le type de retour d'une fonction ne doit jamais être une intersection (lié à $A \Rightarrow B \ = !A \multimap B$).

Automates d'arbres alternants et typage

- Kobayashi (2009) : système de typage des règles de réécriture d'un schéma décidant l'acceptation de celui-ci par un automate alternant donné.
- On type directement le schéma qui produit l'arbre, pas celui-ci (qui est infini et dont la structure peut être très complexe).
- La forme des sous-types est contrainte : le type de retour d'une fonction ne doit jamais être une intersection (lié à $A \Rightarrow B \ = !A \multimap B$).

Automates d'arbres alternants et typage

- Kobayashi (2009) : système de typage des règles de réécriture d'un schéma décidant l'acceptation de celui-ci par un automate alternant donné.
- On type directement le schéma qui produit l'arbre, pas celui-ci (qui est infini et dont la structure peut être très complexe).
- La forme des sous-types est contrainte : le type de retour d'une fonction ne doit jamais être une intersection (lié à $A \Rightarrow B \ = !A \multimap B$).

Automates d'arbres alternants et typage

$$\text{Axiome} \quad \frac{\vdash \ \tau_j :: \kappa \quad (\forall j \in J)}{\Gamma\,, x : \bigwedge_{j\in J} \tau_j :: \kappa \ \vdash \ x : \tau_j :: \kappa}$$

$$\text{App} \quad \frac{\Gamma \vdash M : (\bigwedge_{j\in J} \tau_j) \to \sigma :: \kappa \to \kappa' \qquad \Gamma \vdash N : \tau_j :: \kappa \quad (\forall j \in J)}{\Gamma \vdash MN : \sigma :: \kappa'}$$

$$\text{Lambda} \quad \frac{\Gamma, x : \bigwedge_{j\in J} \tau_j :: \kappa \ \vdash \ M : \sigma :: \kappa'}{\Gamma \vdash \lambda x.M : (\bigwedge_{j\in J} \tau_j) \to \sigma :: \kappa \to \kappa'}$$

Dans App, N est dérivé autant de fois qu'il y a de types dans l'intersection.

Automates d'arbres alternants et typage

$$\text{Axiome} \quad \frac{\vdash \ \tau_j :: \kappa \quad (\forall j \in J)}{\Gamma\ , x : \bigwedge_{j\in J} \tau_j :: \kappa \ \vdash \ x : \tau_j :: \kappa}$$

$$\text{App} \quad \frac{\Gamma \vdash M : (\bigwedge_{j\in J} \tau_j) \to \sigma :: \kappa \to \kappa' \qquad \Gamma \vdash N : \tau_j :: \kappa \quad (\forall j \in J)}{\Gamma \vdash MN : \sigma :: \kappa'}$$

$$\text{Lambda} \quad \frac{\Gamma, x : \bigwedge_{j\in J} \tau_j :: \kappa \ \vdash \ M : \sigma :: \kappa'}{\Gamma \vdash \lambda x.M : (\bigwedge_{j\in J} \tau_j) \to \sigma :: \kappa \to \kappa'}$$

Dans App, N est dérivé autant de fois qu'il y a de types dans l'intersection.

Automates d'arbres alternants et logique linéaire

Idée clef :

$$\text{les automates alternants} = \text{le typage avec intersections} = \text{le } ! \text{ de la logique linéaire}$$

NB : dans des modèles où $\bot = Q$ et non pas $\{*\}$.

Automates d'arbres alternants et logique linéaire

Dans le système de types de Kobayashi, les types intersection sont idempotents :

$$q \wedge q = q$$

Autrement dit, on ne compte pas le nombre d'utilisations de la ressource q.

$\rightarrow$ modèles qualitatifs de la logique linéaire.

Automates d'arbres alternants et logique linéaire

Dans le système de types de Kobayashi, les types intersection sont idempotents :

$$q \wedge q = q$$

Autrement dit, on ne compte pas le nombre d'utilisations de la ressource q.

$\rightarrow$ modèles qualitatifs de la logique linéaire.

Un modèle qualitatif : Scott$L_!$

- Un modèle qualitatif : on retiendra les ressources utilisées mais pas combien de fois elles l'ont été
- Lié à l'idempotence des types intersection considérés
- Modèle à base d'ensembles préordonnés
- Ici : on ne s'intéressera qu'au modèle du λ-calcul donné par ce modèle
- Le type o est interprété par $(Q, =)$
- $A \Rightarrow B$ par $\mathcal{P}(A) \times B$ muni d'un préordre adapté

Un modèle qualitatif : Scott$L_!$

- Un modèle qualitatif : on retiendra les ressources utilisées mais pas combien de fois elles l'ont été
- Lié à l'idempotence des types intersection considérés
- Modèle à base d'ensembles préordonnés
- Ici : on ne s'intéressera qu'au modèle du λ-calcul donné par ce modèle
- Le type o est interprété par $(Q, =)$
- $A \Rightarrow B$ par $\mathcal{P}(A) \times B$ muni d'un préordre adapté

Typage pour ScottL$_!$

Terui (2009) donne un système de types qui construit la sémantique d'un λ-terme dans ScottL$_!$:

$$\text{Axiome} \quad \frac{\exists \alpha' \in X \ \alpha' \geq \alpha}{\Gamma, x : X \vdash x : \alpha}$$

$$\text{App} \quad \frac{\Gamma \vdash M : X \to \alpha \qquad \Gamma \vdash N : \beta \quad (\forall \beta \in X)}{\Gamma \vdash MN : \alpha}$$

$$\text{Lambda} \quad \frac{\Gamma, x : X \vdash M : \alpha}{\Gamma \vdash \lambda x.M : X \to \alpha}$$

Penser X comme une intersection

Typage pour ScottL$_!$

Terui (2009) donne un système de types qui construit la sémantique d'un λ-terme dans ScottL$_!$:

$$\text{Axiome} \quad \frac{\exists \alpha' \in X \ \alpha' \geq \alpha}{\Gamma, x : X \ \vdash \ x : \alpha}$$

$$\text{App} \quad \frac{\Gamma \vdash M : X \to \alpha \qquad \Gamma \vdash N : \beta \quad (\forall \beta \in X)}{\Gamma \vdash MN : \alpha}$$

$$\text{Lambda} \quad \frac{\Gamma, x : X \ \vdash \ M : \alpha}{\Gamma \vdash \lambda x.M : X \to \alpha}$$

Penser X comme une intersection

Un lien entre sémantique et vérification

- Types $\rightarrow$ préordre qui est en fait (équivalent à) l'ensemble des types intersections possibles bâtis sur Q
- Termes $\rightarrow$ sous-ensembles du préordre interprétant leur type
- L'interprétation d'un terme = l'ensemble de ses actions sur les états
- Permet de retrouver les δ des automates alternants d'états Q acceptant un arbre généré par un schéma
- Plus précisément : on peut réduire l'acceptation à l'appartenance à la sémantique

Un lien entre sémantique et vérification

- Types $\rightarrow$ préordre qui est en fait (équivalent à) l'ensemble des types intersections possibles bâtis sur Q
- Termes $\rightarrow$ sous-ensembles du préordre interprétant leur type
- L'interprétation d'un terme $=$ l'ensemble de ses actions sur les états
- Permet de retrouver les δ des automates alternants d'états Q acceptant un arbre généré par un schéma
- Plus précisément : on peut réduire l'acceptation à l'appartenance à la sémantique

Un lien entre sémantique et vérification

De plus :

Pour un terme en forme $\beta\eta$-normale (mais aussi pour d'autres), le typage dans ce système est équivalent au typage dans le système de Kobayashi.

Exécuter un automate se ramène ainsi à calculer la sémantique d'un terme.

Un lien entre sémantique et vérification

De plus :

Pour un terme en forme $\beta\eta$-normale (mais aussi pour d'autres), le typage dans ce système est équivalent au typage dans le système de Kobayashi.

Exécuter un automate se ramène ainsi à calculer la sémantique d'un terme.

Aspects quantitatifs ?

Comment interpréter ces automates dans des modèles quantitatifs ?

Aspects quantitatifs : logique indexée et modèles relationnels

- On considère une variante intuitionniste de la logique linéaire indexée (Bucciarelli - Ehrhard, 2000-2001) pour typer nos termes
- Cette logique indexée reflète la sémantique relationnelle des formules
- On aura : une flèche $\multimap$, et une exponentielle indexée $!_u$
- L'indexation va permettre de superposer des preuves identiques
- Elle joue ici le rôle de l'intersection des types d'un même terme
- NB : ici l'intersection des types n'est pas idempotente.

Aspects quantitatifs : logique indexée et modèles relationnels

- On considère une variante intuitionniste de la logique linéaire indexée (Bucciarelli - Ehrhard, 2000-2001) pour typer nos termes
- Cette logique indexée reflète la sémantique relationnelle des formules
- On aura : une flèche $\multimap$, et une exponentielle indexée $!_u$
- L'indexation va permettre de superposer des preuves identiques
- Elle joue ici le rôle de l'intersection des types d'un même terme
- NB : ici l'intersection des types n'est pas idempotente.

Aspects quantitatifs : logique indexée et modèles relationnels

- On considère une variante intuitionniste de la logique linéaire indexée (Bucciarelli - Ehrhard, 2000-2001) pour typer nos termes
- Cette logique indexée reflète la sémantique relationnelle des formules
- On aura : une flèche $\multimap$, et une exponentielle indexée $!_u$
- L'indexation va permettre de superposer des preuves identiques
- Elle joue ici le rôle de l'intersection des types d'un même terme
- NB : ici l'intersection des types n'est pas idempotente.

Indexation et familles de types intersection

On indexe l'exponentielle $!_u$ pour structurer les preuves.

Considérons par exemple $u : 5 \to 3$ avec :

- $u^{-1}(1) = \emptyset$
- $u^{-1}(2) = \{1, 2, 4\}$
- $u^{-1}(3) = \{3, 5\}$

$!_u \perp_5$ se comprend alors comme une famille de trois formules, correspondant à trois multi-ensembles structurés par u à partir de cinq ressources de $\perp = Q$.

Indexation et familles de types intersection

On indexe l'exponentielle $!_u$ pour structurer les preuves.

Considérons par exemple $u : 5 \to 3$ avec :

- $u^{-1}(1) = \emptyset$
- $u^{-1}(2) = \{1, 2, 4\}$
- $u^{-1}(3) = \{3, 5\}$

$!_u \perp_5$ se comprend alors comme une famille de trois formules, correspondant à trois multi-ensembles structurés par u à partir de cinq ressources de $\perp = Q$.

La règle clef

La règle clef est la promotion :

$$\frac{x_1 : !_{u_1} A_1 \ , \ \ldots \ , \ x_n : !_{u_n} A_n \vdash_J M : B}{x_1 : !_{v \circ u_1} A_1 \ , \ \ldots \ , \ x_n : !_{v \circ u_n} A_n \vdash_K M : !_v B} \qquad v : J \to K$$

La règle clef

On l'utilise lors des applications pour remplacer :

$$\text{App} \quad \frac{\Gamma \vdash M : (\bigwedge_{j\in J} \tau_j) \to \sigma :: \kappa \to \kappa' \qquad \Gamma \vdash N : \tau_j :: \kappa \quad (\forall j \in J)}{\Gamma \vdash MN : \sigma :: \kappa'}$$

par un bloc de preuve dans lequel on superpose les dérivations de N, en interprétant $\bigwedge$ avec $!_u$

On peut alors oublier le terme : la dérivation logique permet de le reconstruire.

De plus, cette dérivation reflète un point de la sémantique relationnelle du terme.
L'ensemble des dérivations dans cette logique = la sémantique relationnelle du terme.

La règle clef

On l'utilise lors des applications pour remplacer :

$$\text{App} \quad \frac{\Gamma \vdash M : (\bigwedge_{j\in J} \tau_j) \to \sigma :: \kappa \to \kappa' \qquad \Gamma \vdash N : \tau_j :: \kappa \quad (\forall j \in J)}{\Gamma \vdash MN : \sigma :: \kappa'}$$

par un bloc de preuve dans lequel on superpose les dérivations de N, en interprétant $\bigwedge$ avec $!_u$

On peut alors oublier le terme : la dérivation logique permet de le reconstruire.

De plus, cette dérivation reflète un point de la sémantique relationnelle du terme.
L'ensemble des dérivations dans cette logique = la sémantique relationnelle du terme.

La règle clef

On l'utilise lors des applications pour remplacer :

$$\text{App}\quad \frac{\Gamma \vdash M : (\bigwedge_{j\in J} \tau_j) \to \sigma :: \kappa \to \kappa' \qquad \Gamma \vdash N : \tau_j :: \kappa \quad (\forall j \in J)}{\Gamma \vdash MN : \sigma :: \kappa'}$$

par un bloc de preuve dans lequel on superpose les dérivations de N, en interprétant $\bigwedge$ avec $!_u$

On peut alors oublier le terme : la dérivation logique permet de le reconstruire.

De plus, cette dérivation reflète un point de la sémantique relationnelle du terme.
L'ensemble des dérivations dans cette logique = la sémantique relationnelle du terme.

Un résultat

On peut traduire une dérivation de typage du système de Kobayashi en une dérivation de notre variante de la logique indexée, et réciproquement, modulo $\beta\eta$-conversion et raffinement du type intersection.

Aspects relationnels

une preuve de ce système $\Leftrightarrow$ un point de la sémantique relationnelle

$\Leftrightarrow$ une preuve chez Kobayashi
(modulo β, η et sous-typage)

$\Leftrightarrow$ l'exécution d'automates alternants

Lien entre les deux systèmes

- Les deux systèmes sémantiques reflètent le calcul des points des sémantiques des termes dans Rel et ScottL.
- Rel est quantitatif, ScottL qualitatif
- Lien entre les deux : (Ehrhard 2012) : ScottL est le collapse extentionnel de Rel
- ScottL, c'est Rel sans compter la multiplicité (et en mettant un peu d'ordre !)

Lien entre les deux systèmes

- Les deux systèmes sémantiques reflètent le calcul des points des sémantiques des termes dans Rel et ScottL.
- Rel est quantitatif, ScottL qualitatif
- Lien entre les deux : (Ehrhard 2012) : ScottL est le collapse extentionnel de Rel
- ScottL, c'est Rel sans compter la multiplicité (et en mettant un peu d'ordre !)

La suite

- Les automates alternants utilisés en vérification sont en général à parité
- Ceci modélise le μ-calcul modal, logique avec deux points fixes (induction / coinduction)
- Il reste donc à ajouter aux systèmes sémantiques considérés ici deux récursions sur les types pour refléter l'exécution d'un tel automate sur un terme (ou sur un schéma, en traitant proprement Y).

La suite

- Les automates alternants utilisés en vérification sont en général à parité
- Ceci modélise le μ-calcul modal, logique avec deux points fixes (induction / coinduction)
- Il reste donc à ajouter aux systèmes sémantiques considérés ici deux récursions sur les types pour refléter l'exécution d'un tel automate sur un terme (ou sur un schéma, en traitant proprement Y).