Programmer = démontrer?
La correspondance de Curry-Howard aujourd'hui

Huitième cours

À pas comptés : les techniques de step-indexing

Xavier Leroy

Collège de France

2019-01-09

COLLÈGE
DE FRANCE
1530

I

Les relations logiques en sémantique opérationnelle

Rappels sur les relations logiques

(Cours du 19 décembre 2018)

Une relation logique est une famille de relations $R(t)$
(indexée par un type t) entre deux (dénotations de) programmes,
et telle que

> *Deux fonctions sont reliées au type $t \to s$ si et seulement si*
> *elles envoient des arguments reliés au type t*
> *sur des résultats reliés au type s.*

Exemple :
les fonctions $\lambda n.\ n + n$ et $\lambda n.\ n \times 2$ sont reliées par $R(\texttt{int} \to \texttt{int})$,
en supposant que $R(\texttt{int})$ est la relation égalité.

Un cadre de sémantique opérationnelle

Par la suite, on n'utilisera pas de sémantiques dénotationnelles, uniquement des approches opérationnelles.

Les relations logiques relient deux expressions du langage a_1, a_2.

La sémantique est donnée par la relation de réduction $a \to a'$.

$a \to a_1 \to \cdots \to a_n \to \cdots$		divergence
$a \to a_1 \to \cdots \to v \not\to$	$v \in \mathrm{Val}$	terminaison normale
$a \to a_1 \to \cdots \to a_n \not\to$	$a_n \notin \mathrm{Val}$	terminaison en erreur

Pour simplifier encore plus, on fixe une stratégie de réduction : l'appel par valeur

$$(\lambda x.\, a)\, v \to a[x \leftarrow v] \quad \text{si } v \in \mathrm{Val} \qquad (\text{réduction } \beta_v)$$

Relations logiques opérationnelles

On définit deux relations logiques : $V(t)$ entre valeurs

$$V(\texttt{int}) = \{\ (n, n) \mid n \text{ entier}\ \}$$
$$V(t \to s) = \{\ (\lambda x_1.\, a_1, \lambda x_2.\, a_2) \mid \forall (v_1, v_2) \in V(t),\ (a_1[x_1 \leftarrow v_1], a_2[x_2 \leftarrow v_2]) \in E(s)\ \}$$

et $E(t)$ entre expressions (calculs)

$$E(t) = \{\ (a_1, a_2) \mid \forall b_1,\ a_1 \xrightarrow{*} b_1 \wedge b_1 \text{ irréductible} \Rightarrow \exists b_2,\ a_2 \xrightarrow{*} b_2 \wedge (b_1, b_2) \in V(t)\ \}$$

La définition est bien fondée par récurrence sur t :
si on expanse la définition de $E(s)$ dans la définition de $V(t \to s)$, on voit que cette dernière dépend uniquement de $V(t)$ et de $V(s)$.

Relations logiques et équivalence contextuelle

Théorème (théorème fondamental des relations logiques)

Si $x_1 : t_1, \ldots, x_n : t_n \vdash a : t$, *les interprétations de* a *dans deux environnements reliés sont reliées :*

$$\text{si } (v_i, v_i') \in V(t_i) \text{ pour } i = 1, \ldots, n, \text{ alors } (a[x_i \leftarrow v_i], a[x_i \leftarrow v_i']) \in E(t)$$

Corollaire (équivalence contextuelle)

Si $(a_1, a_2) \in E(t)$ *et* $(a_2, a_1) \in E(t)$,
alors pour tout contexte $C[.]$ *de type* $t \to \mathit{int}$ *et tout entier* n,

$$C[a_1] \xrightarrow{*} n \text{ si et seulement si } C[a_2] \xrightarrow{*} n$$

(Autres corollaires : indépendance vis-à-vis des représentations, si on ajoute des types abstraits ; ≪théorèmes gratuits≫ si on ajoute du polymorphisme ; cf. cours du 19 décembre 2018)

Relations logiques unaires

Dans ce cadre opérationnel, les relations logiques unaires donnent une interprétation des types t comme ensembles de valeurs $V(t)$:

$$V(\texttt{int}) = \{\ n \mid n \text{ entier}\ \}$$
$$V(t \to s) = \{\ \lambda x.\, a \mid \forall v \in V(t),\ a[x \leftarrow v] \in E(s)\ \}$$

et comme ensembles d'expressions $E(t)$:

$$E(t) = \{\ a \mid \forall b,\ a \xrightarrow{*} b \wedge b \text{ irréductible} \Rightarrow b \in V(t)\ \}$$

Note : une expression en erreur (irréductible mais pas une valeur, comme 1 2) n'est dans aucun $V(t)$. Donc une expression qui se réduit en erreur (comme $a \xrightarrow{*} 1\ 2$) n'est dans aucun $E(t)$.

Relations logiques et sûreté du typage

Théorème (théorème fondamental des relations logiques)

Si $x_1 : t_1, \ldots, x_n : t_n \vdash a : t$, *et si* $v_i \in V(t_i)$ *pour* $i = 1, \ldots, n$,
alors $a[x_1 \leftarrow v_1, \ldots, x_n \leftarrow v_n] \in E(t)$

Corollaire (sûreté du typage)

Si $\vdash a : t$, *alors* a *ne se réduit pas en erreur :*
soit a *se réduit en une valeur, soit* a *diverge.*

Well-typed terms do not go wrong. (R. Milner)

II

Types récursifs

Types de données non récursifs

Il est facile d'étendre la relation V à des types de données non récursifs comme le produit $t \times s$ et la somme $t + s$:

$$V(t \times s) = \{(v, w) \mid v \in V(t) \wedge w \in V(s)\}$$

$$V(t + s) = \{\mathtt{inj}_1(v) \mid v \in V(t)\} \cup \{\mathtt{inj}_2(w) \mid w \in V(s)\}$$

La définition de $V(t)$ reste bien fondée par récurrence sur t.

Types inductifs

Pour des types inductifs comme les listes

```
type 'a list = Nil | Cons of 'a * 'a list
```

on a une circularité apparente :

$$V(t\ \texttt{list}) = \{\texttt{Nil}\} \cup \{\ \texttt{Cons}(v, w) \mid v \in V(t) \wedge w \in V(t\ \texttt{list})\ \}$$

Mais la définition de $v \in V(t)$ reste bien fondée : dans le cas des listes, on fait une récurrence locale sur la structure de la valeur v ; ensuite, une récurrence globale sur la structure du type t. En d'autres termes :

$$V(t\ \texttt{list}) = \mu X.\ \{\texttt{Nil}\} \cup \{\ \texttt{Cons}(v, w) \mid v \in V(t) \wedge w \in X\ \}$$

c'est-à-dire

$$V(t\ \texttt{list}) = \{\ \texttt{Cons}(v_1, \ldots, \texttt{Cons}(v_n, \texttt{Nil})) \mid v_i \in V(t)\ \}$$

Types récursifs généraux

Le problème est les types récursifs qui ne sont pas inductifs
(occurrences non strictement positives dans les types des constructeurs)

```
type lam = Lam of (lam -> lam)
```

La ≪définition≫ de $V(\texttt{lam})$ est clairement circulaire :

$$\begin{aligned} V(\texttt{lam}) &= \{\, \texttt{Lam}(f) \mid f \in V(\texttt{lam} \to \texttt{lam}) \,\} \\ &= \{\, \texttt{Lam}(\lambda x.\, a) \mid \forall v \in V(\texttt{lam}),\ a[x \leftarrow v] \in E(\texttt{lam}) \,\} \end{aligned}$$

Cette ≪définition≫ est juste une équation de point fixe, qu'on ne sait pas résoudre dans les ensembles, mais qu'on peut résoudre dans d'autres catégories comme les domaines de Scott p.ex.
(Cf. le domaine $D_\infty \approx D_\infty \to_{cont} D_\infty$.)

An indexed model of recursive types

(A. Appel et D. McAllester, TOPLAS(23), 2001)

Appel et McAllester ont l'idée de fonder la définition de $V(t)$ non pas par récurrence sur la structure de t, mais sur un autre paramètre (un entier) : le nombre d'étapes de réduction que l'on se permet de faire sur les expressions et les (applications de) valeurs.

La technique passe dans la littérature sous le nom de step-indexing («comptage de pas», comptage d'étapes de calcul).

Intuition du comptage de pas

Que veut dire, sémantiquement, que l'expression a est de type `int` ?

Réponse habituelle :

- si $a \xrightarrow{*} n$ (n entier) ou a diverge : oui, a est de type `int` ;
- si a se réduit en erreur ou en une valeur qui n'est pas un entier : non, a n'est pas de type `int`.

Réponse «à pas comptés» : pour un entier k donné,

- si en k étapes au plus a se réduit en un entier ou n'atteint pas une forme normale : oui, a semble de type `int` pour k étapes ;
- si en k étapes au plus a se réduit en erreur ou en une valeur qui n'est pas un entier : non, a n'est pas de type `int`.

Finalement, a est de type `int` si pour tout $k \in \mathbb{N}$, a semble de type `int` pour k étapes.

An indexed model of recursive types

(A. Appel et D. McAllester, TOPLAS(23), 2001)

Notation : $a \to_k b$ pour «a se réduit vers b en k étapes».

$$V_k(\texttt{int}) = \{\ n \mid n \text{ entier}\ \}$$
$$V_k(t \to s) = \{\ \lambda x.\, a \mid \forall j < k,\ \forall v \in V_j(t),\ a[x \leftarrow v] \in E_j(s)\ \}$$
$$E_k(t) = \{\ a \mid \forall j \leq k,\ \forall b,\ a \to_j b \wedge b \text{ irréductible} \Rightarrow b \in V_{k-j}(t)\ \}$$

Intuitions :

- L'expression a semble être de type t en k étapes si, ayant dépensé $j \leq k$ étapes pour réduire a en b, b semble être une valeur de type t en $k - j$ étapes restantes.
- Une abstraction $\lambda x.a$ semble être une valeur $t \to s$ en k étapes si l'application $(\lambda x.a)\ v$ semble être de type t en au plus k étapes. La β-réduction compte pour 1 étape, d'où $a[x \leftarrow v] \in E_j(s)$ pour $j < k$.
- En j étapes, l'expression $a[x \leftarrow v]$ ne peut pas examiner la valeur v pendant plus de j étapes ! Donc il suffit que v semble de type t pour j étapes.

An indexed model of recursive types

(A. Appel et D. McAllester, TOPLAS(23), 2001)

Notation : $a \to_k b$ pour «a se réduit vers b en k étapes».

$$V_k(\texttt{int}) = \{\ n \mid n \text{ entier}\ \}$$
$$V_k(t \to s) = \{\ \lambda x.\, a \mid \forall j < k,\ \forall v \in V_j(t),\ a[x \leftarrow v] \in E_j(s)\ \}$$
$$E_k(t) = \{\ a \mid \forall j \leq k,\ \forall b,\ a \to_j b \wedge b \text{ irréductible} \Rightarrow b \in V_{k-j}(t)\ \}$$

Intuitions :

- L'expression a semble être de type t en k étapes si, ayant dépensé $j \leq k$ étapes pour réduire a en b, b semble être une valeur de type t en $k - j$ étapes restantes.
- Une abstraction $\lambda x.a$ semble être une valeur $t \to s$ en k étapes si l'application $(\lambda x.a)\ v$ semble être de type t en au plus k étapes. La β-réduction compte pour 1 étape, d'où $a[x \leftarrow v] \in E_j(s)$ pour $j < k$.
- En j étapes, l'expression $a[x \leftarrow v]$ ne peut pas examiner la valeur v pendant plus de j étapes ! Donc il suffit que v semble de type t pour j étapes.

An indexed model of recursive types

(A. Appel et D. McAllester, TOPLAS(23), 2001)

Notation : $a \to_k b$ pour «a se réduit vers b en k étapes».

$$V_k(\texttt{int}) = \{\ n \mid n \text{ entier}\ \}$$
$$V_k(t \to s) = \{\ \lambda x.\, a \mid \forall j < k,\ \forall v \in V_j(t),\ a[x \leftarrow v] \in E_j(s)\ \}$$
$$E_k(t) = \{\ a \mid \forall j \leq k,\ \forall b,\ a \to_j b \wedge b \text{ irréductible} \Rightarrow b \in V_{k-j}(t)\ \}$$

Intuitions :

- L'expression a semble être de type t en k étapes si, ayant dépensé $j \leq k$ étapes pour réduire a en b, b semble être une valeur de type t en $k - j$ étapes restantes.
- Une abstraction $\lambda x.a$ semble être une valeur $t \to s$ en k étapes si l'application $(\lambda x.a)\ v$ semble être de type t en au plus k étapes. La β-réduction compte pour 1 étape, d'où $a[x \leftarrow v] \in E_j(s)$ pour $j < k$.
- En j étapes, l'expression $a[x \leftarrow v]$ ne peut pas examiner la valeur v pendant plus de j étapes ! Donc il suffit que v semble de type t pour j étapes.

An indexed model of recursive types

(A. Appel et D. McAllester, TOPLAS(23), 2001)

Notation : $a \to_k b$ pour «a se réduit vers b en k étapes».

$$V_k(\texttt{int}) = \{\ n \mid n \text{ entier}\ \}$$
$$V_k(t \to s) = \{\ \lambda x.\, a \mid \forall j < k,\ \forall v \in V_j(t),\ a[x \leftarrow v] \in E_j(s)\ \}$$
$$E_k(t) = \{\ a \mid \forall j \leq k,\ \forall b,\ a \to_j b \wedge b \text{ irréductible} \Rightarrow b \in V_{k-j}(t)\ \}$$

Intuitions :

- L'expression a semble être de type t en k étapes si, ayant dépensé $j \leq k$ étapes pour réduire a en b, b semble être une valeur de type t en $k - j$ étapes restantes.
- Une abstraction $\lambda x.a$ semble être une valeur $t \to s$ en k étapes si l'application $(\lambda x.a)\ v$ semble être de type t en au plus k étapes. La β-réduction compte pour 1 étape, d'où $a[x \leftarrow v] \in E_j(s)$ pour $j < k$.
- En j étapes, l'expression $a[x \leftarrow v]$ ne peut pas examiner la valeur v pendant plus de j étapes ! Donc il suffit que v semble de type t pour j étapes.

Ajout des types récursifs à la relation logique

Si $F : \texttt{Type} \to \texttt{Type}$, on note μF le type algébrique caractérisé par

$$\texttt{roll} : F(\mu F) \to \mu F \qquad \texttt{unroll} : \mu F \to F(\mu F)$$

et la règle de réduction $\texttt{unroll}(\texttt{roll}(v)) \to v$.

Il suffit de prendre

$$V_0(\mu F) = \{\, \texttt{roll}(v) \mid v \text{ valeur} \,\}$$
$$V_{k+1}(\mu F) = \{\, \texttt{roll}(v) \mid v \in V_k(F(\mu F)) \,\}$$

La définition de $V_k(t)$ n'est plus bien fondée par récurrence sur t, mais reste bien fondée par récurrence sur k. C'est évident pour le type μF, et c'est vrai aussi pour le type $t \to s$, puisque la définition de $V_k(t \to s)$ fait intervenir $V_j(t)$ et $V_j(s)$ seulement pour $j < k$.

Application : le lambda-calcul pur

On peut coder le lambda-calcul pur avec le type $D \stackrel{def}{=} \mu(\lambda t.\, t \to t)$.

Sans surprises, on a

$$V_0(D) = \{\, \texttt{roll}(v) \mid v \text{ valeur} \,\}$$
$$V_{k+1}(D) = \{\, \texttt{roll}(\lambda x.a) \mid \forall j < k, \forall v \in V_j(D),\ a[x \leftarrow v] \in E_j(D) \,\}$$

Principales propriétés

Décroissance : $V_k(t) \subseteq V_j(t)$ et $E_k(t) \subseteq E_j(t)$ si $k \geq j$.

Compatibilité avec les réductions :
si $a \to b$ alors $a \in E_{k+1}(t)$ ssi $b \in E_k(t)$.

Théorème fondamental :
si $x_1 : t_1, \ldots, x_n : t_n \vdash a : t$, et si $v_i \in V_k(t_i)$ pour $i = 1, \ldots, n$,
alors $a[x_1 \leftarrow v_1, \ldots, x_n \leftarrow v_n] \in E_k(t)$

La comptabilité des pas

Lemme (le cas de l'application)

Si $a \in E_k(t \to s)$ et $b \in E_k(t)$, alors $a\ b \in E_k(s)$.

Démonstration.

Une réduction de $a\ b$ en un terme irréductible d est de la forme

$$a\ b \to_n (\lambda x.\, c)\ b \to_m (\lambda x.\, c)\ v \to_1 c[x \leftarrow v] \to_p d$$

avec $j = n + m + 1 + p$ étapes de réduction et $j \leq k$.

Pour conclure, il faut montrer $d \in V_q(s)$ avec $q = k - j$.

Par hyp sur a, $\lambda x.\, c \in V_{k-n}(t \to s)$ et donc $\lambda x.\, c \in V_{p+q+1}(t \to s)$ (1).

Par hyp sur b, $v \in V_{k-m}(t)$ et donc $v \in V_{p+q}(t)$ (2).

Par (1) et (2), $c[x \leftarrow v] \in E_{p+q}(s)$ (3).

Par (3), $d \in V_q(s)$, CQFD. □

Extension aux relations logiques binaires

$$
\begin{aligned}
V_k(\texttt{int}) &= \{\ (n, n) \mid n \text{ entier}\ \} \\
V_k(t \to s) &= \{\ (\lambda x_1.\, a_1, \lambda x_2.\, a_2) \mid \\
&\qquad \forall j < k,\ \forall (v_1, v_2) \in V_j(t),\ (a_1[x_1 \leftarrow v_1], a_2[x_2 \leftarrow v_2]) \in E_j(s)\ \} \\
V_0(\mu F) &= \{\ (\texttt{roll}(v_1), \texttt{roll}(v_2)) \mid v_1, v_2 \text{ valeurs}\ \} \\
V_{k+1}(\mu F) &= \{\ (\texttt{roll}(v_1), \texttt{roll}(v_2)) \mid (v_1, v_2) \in V_k(F(\mu F))\ \} \\
E_k(t) &= \{\ (a_1, a_2) \mid \forall j \leq k,\ \forall b_1,\ a_1 \to_j b_1 \wedge b_1 \text{ irréductible} \Rightarrow \\
&\qquad \exists b_2,\ a_2 \xrightarrow{*} b_2 \wedge (b_1, b_2) \in V_{k-j}(t)\ \}
\end{aligned}
$$

Note : on a $a_1 \to_j b_1$ (j étapes) et $a_2 \xrightarrow{*} b_2$ (nombre quelconque d'étapes), ce qui permet de relier des calculs a_1, a_2 de durées différentes.

III

Formulation modale du step-indexing

Reformuler la comptabilité des pas

Revenons sur la définition de $E_k(t)$, l'ensemble des expressions a qui semblent de type t en k étapes :

$$E_k(t) = \{\ a \mid \forall j \leq k,\ \forall b,\ a \to_j b \wedge b \text{ irréductible} \Rightarrow b \in V_{k-j}(t)\ \}$$

Au lieu de considérer $j \leq k$ étapes de réduction ($a \to_j b$),
on peut considérer deux cas : 0 réductions (irréductible) et 1 réduction.

- Si a est irréductible, $a \in E_k(t)$ ssi $a \in V_k(t)$.
- Si $a \to b$, $a \in E_k(t)$ ssi $b \in E_{k-1}(t)$ ou $k = 0$.

D'où une autre définition, équivalente et toujours bien fondée par récurrence sur k :

$$E_k(t) = \{\ a \mid (a \text{ irréductible} \Rightarrow a \in V_k(t)) \wedge (\forall b,\ a \to b \Rightarrow b \in E_{k-1}(t))\ \}$$

avec, par convention, $E_{-1}(t) =$ tous les termes.

Le retour de la modalité «plus tard» ($\triangleright$)

On définit $\triangleright E$ par $(\triangleright E)_{k+1} = E_k$ et $(\triangleright E)_0 =$ tous les termes. Alors :

$$E_k(t) = \{\ a \mid (a \text{ irréductible} \Rightarrow a \in V_k(t)) \wedge (\forall b,\ a \to b \Rightarrow b \in \triangleright E_k(t))\ \}$$

De même, définissons $(\triangleright V)_{k+1} = V_k$ et $(\triangleright V)_0 =$ toutes les valeurs.
On peut réécrire les cas de la définition

$$V_0(\mu F) = \{\ \texttt{roll}(v) \mid v \text{ valeur}\ \}$$
$$V_{k+1}(\mu F) = \{\ \texttt{roll}(v) \mid v \in V_k(F(\mu F))\ \}$$

en un seul cas «modal»

$$V_k(\mu F) = \{\ \texttt{roll}(v) \mid v \in \triangleright V_k(F(\mu F))\ \}$$

Le retour de la modalité «plus tard» ($\triangleright$)

Dans le même esprit, on peut réécrire le cas $V_k(t \to s)$ en utilisant $\triangleright V$.
On avait

$$V_k(t \to s) = \{\ \lambda x.\, a \mid \forall j < k, \forall v \in V_j(t),\ a[x \leftarrow v] \in E_j(s)\ \}$$

et on peut écrire

$$V_k(t \to s) = \{\ \lambda x.\, a \mid \forall v, \forall j \leq k, v \in \triangleright V_j(t) \Rightarrow a[x \leftarrow v] \in \triangleright E_j(s)\ \}$$

Cela nous donne une quantification $\forall j \leq k$ qui a la forme de l'implication dans les modèles de Kripke intuitionniste :
$k \Vdash A \Rightarrow B$ ssi $\forall j \leq k,\ j \Vdash A \Rightarrow j \Vdash B$.

Une relation logique modale

Finalement, on peut rendre le paramètre k (le nombre d'étapes) implicite en se plaçant dans la logique du topos des arbres vue au cours précédent, avec sa modalité $\triangleright$.

$E(t)$ et $V(t)$ sont alors définis par les équations

$$
\begin{aligned}
V(\texttt{int}) &= \{\ n \mid n \text{ entier}\ \} \\
V(t \to s) &= \{\ \lambda x.\, a \mid \forall v \in \triangleright V(t),\ a[x \leftarrow v] \in \triangleright E(s)\ \} \\
V(\mu F) &= \{\ \texttt{roll}(v) \mid v \in \triangleright V(F(\mu F))\ \} \\
E(t) &= \{\ a \mid (a \text{ irréductible} \Rightarrow a \in V(t)) \wedge (\forall b,\ a \to b \Rightarrow b \in \triangleright E(t))\ \}
\end{aligned}
$$

Notons que E et V sont définies comme une fonction de $\triangleright E$ et $\triangleright V$.
La règle de Löb garantit qu'il existe un unique point fixe pour E et V.

Propriétés de la modalité $\triangleright$

$$A \Rightarrow \triangleright A$$

$$\triangleright(A \wedge B) \text{ ssi } \triangleright A \wedge \triangleright B$$

$$\triangleright(A \vee B) \text{ ssi } \triangleright A \vee \triangleright B$$

$$\triangleright(A \Rightarrow B) \text{ ssi } \triangleright A \Rightarrow \triangleright B$$

si $\triangleright A \Rightarrow A$ alors A (règle de Löb)

si $A \wedge (\triangleright A \Rightarrow \triangleright B) \Rightarrow B$ alors $A \Rightarrow B$ (≪récurrence de Löb≫)

Plus de comptabilité des pas

Lemme (le cas de l'application)

Si $a \in E(t \to s)$ et $b \in E(t)$, alors $a\ b \in E(s)$.

Démonstration.

Par récurrence de Löb. L'hypothèse de récurrence est
$a' \in \triangleright E(t \to s) \wedge b' \in \triangleright E(t) \Rightarrow a'\ b' \in \triangleright E(s)$ pour tous a', b'.
On raisonne par cas suivant que a ou b se réduit.

- a et b sont irréductibles. Alors $a \in V(t \to s)$ et donc a est de la forme $\lambda x.c$. Aussi, $b \in V(t)$ est une valeur.

  Par définition de $V(t \to s)$ et parce que $b \in \triangleright V(t)$, on a $c[x \leftarrow v] \in \triangleright E(s)$. Par ailleurs, $a\ b \to c[x \leftarrow v]$. Donc $a\ b \in E(s)$.

- $a \to a'$. Alors $a' \in \triangleright E(t \to s)$ et donc par hypothèse de récurrence $a'\ b \in \triangleright E(s)$. Comme $a\ b \to a'\ b$, il vient $a\ b \in E(s)$.

- a est irréductible et $b \to b'$. Comme le cas précédent.

□

Compter certaines réductions uniquement

On peut choisir de compter les réductions $\texttt{unroll}(\texttt{roll}(v)) \to v$ mais pas les β-réductions, ce qui revient à utiliser $\triangleright$ pour les types μF mais pas pour les types $t \to s$.

$$
\begin{aligned}
V(\texttt{int}) &= \{\ n \mid n \text{ entier}\ \} \\
V(t \to s) &= \{\ \lambda x.\, a \mid \forall v \in V(t),\ a[x \leftarrow v] \in E(s)\ \} \\
V(\mu F) &= \{\ \texttt{roll}(v) \mid v \in \triangleright V(F(\mu F))\ \} \\
E(t) &= \{\ a \mid (\forall b,\ a \xrightarrow{*}_{\beta} b \wedge b \text{ irréductible} \Rightarrow b \in V(t)) \\
&\qquad \wedge (\forall b,\ a \xrightarrow{*}_{\beta} \to_{unroll} b \Rightarrow b \in \triangleright E(t))\ \}
\end{aligned}
$$

La définition de $V(t)$ et $E(t)$ est bien fondée par récurrence sur la structure du type t puis par une récurrence de Löb.

Extension aux relations logiques binaires

Sans surprises.

$$
\begin{aligned}
V(\texttt{int}) &= \{\ (n, n) \mid n \text{ entier}\ \} \\
V(t \to s) &= \{\ (\lambda x_1.\, a_1, \lambda x_2.\, a_2) \mid \\
&\qquad \forall (v_1, v_2) \in \triangleright V(t),\ (a_1[x_1 \leftarrow v_1], a_2[x_2 \leftarrow v_2]) \in \triangleright E(s)\ \} \\
V(\mu F) &= \{\ (\texttt{roll}(v_1), \texttt{roll}(v_2)) \mid (v_1, v_2) \in \triangleright V(F(\mu F))\ \} \\
E(t) &= \{\ (a_1, a_2) \mid (a_1 \text{ irréductible} \Rightarrow \exists b_2, a_2 \xrightarrow{*} b_2 \wedge (a_1, b_2) \in V(t)) \\
&\qquad \wedge\ (\forall b_1,\ a_1 \to b_1 \Rightarrow \exists b_2, a_2 \xrightarrow{*} b_2 \wedge (b_1, b_2) \in \triangleright E(t))\ \}
\end{aligned}
$$

IV

État mutable

L'état mutable

C'est la caractéristique principale des langages impératifs :
la capacité de modifier ≪en place≫ une structure de données déjà construite ou une variable déjà définie.

Exemple (Concaténation ≪en place≫ de deux listes)

```
struct list { int head; struct list * tail; }

void concat (struct list * l, struct list * m)
{
  while (l->tail != NULL) l = l->tail;
  l->tail = m;
}
```

Les références

Une présentation de l'état mutable utilisée par les langages de la famille ML (langages fonctionnels et impératifs, typés).

Une référence $\approx$ cellule d'indirection mutable $\approx$ tableau à 1 élément.

Exemple : un équivalent OCaml des listes mutables de C

```
type 'a mlist = Nil | Cons of 'a ref * 'a mlist ref
```

Les opérations sur les références :

$\texttt{ref} : t \to t\ \texttt{ref}$	création et initialisation
$! : t\ \texttt{ref} \to t$	déréférencement (valeur courante)
$:= : t\ \texttt{ref} \to t \to \texttt{unit}$	affectation (changement de valeur)

Sémantique des références

Une sémantique par β-réductions simples est fausse car elle ne rend pas compte du partage d'une référence entre une lecture et une écriture :

$$\texttt{let } r = \texttt{ref 1 in } r := 2; !r \neq (\texttt{ref 1} := 2); !(\texttt{ref 1})$$

Il faut un niveau d'indirection :

- les références ont pour valeurs des adresses ℓ ($\approx$ entiers);
- un état mémoire m : adresse $\rightarrow_{fin}$ valeur donne la valeur courante de chaque référence;
- la sémantique opérationnelle réduit des configurations $\langle a, m \rangle$ (un terme a dans un état mémoire m).

Règles de réduction pour les références

$$\langle (\lambda x.\, a)\; v,\, m\rangle \to \langle a[x \leftarrow v],\, m\rangle \qquad \text{(réduction } \beta_v \text{ usuelle)}$$

$$\langle \texttt{ref}\;\; v,\, m\rangle \to \langle \ell,\, m + \{\ell \mapsto v\}\rangle \qquad \text{si } \ell \notin \mathit{Dom}(m)$$

$$\langle !\ell,\, m\rangle \to \langle m(\ell),\, m\rangle \qquad \text{si } \ell \in \mathit{Dom}(m)$$

$$\langle \ell := v,\, m\rangle \to \langle \texttt{()},\, m + \{\ell \mapsto v\}\rangle \qquad \text{si } \ell \in \mathit{Dom}(m)$$

Typage des états mémoire

Un état mémoire est un objet ≪hétérogène≫ : deux adresses différentes peuvent contenir deux valeurs de types différents.

Un type d'état mémoire M : Adr $\rightarrow_{fin}$ Type associe un type à chaque adresse.

Initialement on va dire que $M(\ell)$ est un type syntaxique (c.à.d. une expression de type) et non un type sémantique (un ensemble de valeurs).

Évolution des types d'états mémoire

D'un côté : le type $M(\ell)$ d'une adresse valide ℓ doit rester le même tout au long de l'exécution. Sinon on pourrait casser la sûreté du typage :

$$\ell := 1 \quad \to \cdots \to \quad !\ell\, 2$$

(possible si $M(\ell) = \texttt{int}$) (possible si $M(\ell) = \texttt{int} \to \texttt{int}$)

De l'autre côté : lorsqu'on alloue une nouvelle référence à l'adresse ℓ, il faut mettre dans $M(\ell)$ le type t de son contenu.

D'où un ordre entre types d'états mémoire : $M' \sqsupseteq M$
qui signifie «M peut évoluer en M' durant l'exécution».

$$M' \sqsupseteq M \stackrel{def}{=} Dom(M') \supseteq Dom(M) \wedge \forall \ell \in Dom(M),\ M'(\ell) = M(\ell)$$

Un modèle syntaxique des types références

On interprète des couples (type t, type mémoire M) par un ensemble de valeurs $V(t)(M)$ ou d'expressions $E(t)(M)$. Un type mémoire M est interprété par un ensemble $[M]$ d'états mémoires.

$$
\begin{aligned}
V(\texttt{int})(M) &= \{\, n \mid n \text{ entier} \,\} \\
V(t\ \texttt{ref})(M) &= \{\, \ell \mid M(\ell) = t \,\} \\
V(t \to s)(M) &= \{\, \lambda x.\, a \mid \forall M' \sqsupseteq M,\ \forall v \in \triangleright V(t)(M'),\ a[x \leftarrow v] \in \triangleright E(s)(M') \,\} \\
[M] &= \{\, m \mid Dom(m) = Dom(M) \\
&\qquad \wedge \forall \ell \in Dom(m),\ m(\ell) \in V(M(\ell))(M) \,\} \\
E(t)(M) &= \{\, a \mid \forall m \in [M], \\
&\qquad (\langle a, m\rangle \text{ irréductible} \Rightarrow a \in V(t)(M)) \\
&\qquad \wedge (\forall b, \forall m',\ \langle a, m\rangle \to \langle b, m'\rangle \Rightarrow \\
&\qquad\qquad \exists M' \sqsupseteq M,\ m' \in [M'] \wedge b \in \triangleright E(t)(M')) \,\}
\end{aligned}
$$

Un modèle syntaxique des types références

Ce typage de la mémoire par des types syntaxiques suffit pour démontrer la sûreté du typage en présence de références.

On peut vouloir un typage davantage ≪sémantique≫, où une adresse est associée à un ensemble de valeurs possibles stockées à cette adresse.

Par exemple, c'est utile pour représenter des invariants sur la valeur d'une référence qui découlent de son ≪encapsulation≫ dans une fonction :

```
let gensym = let c = ref 0 in fun () -> c := !c + 1; !c
```

En supposant une arithmétique entière exacte (sans débordements), on a un invariant `!c >= 0` qu'on voudrait refléter dans le modèle en prenant $M(\ell) = \{n \mid n \geq 0\}$ si ℓ est la valeur de c.

Un modèle sémantique des types références

Essayons donc de prendre $\mathit{TypeMem} \stackrel{def}{=} \mathit{Adr} \to_{fin} \mathit{TypeSem}$.

Problème : un type sémantique *TypeSem* n'est pas juste un ensemble de valeurs, c'est un ensemble de valeurs paramétrées par un type mémoire, cf. $V(t)(M) = \{v \mid \cdots\}$.

On obtient donc une circularité :

$$\mathit{TypeSem} = \mathit{TypeMem} \to \mathcal{P}(\mathit{Val})$$
$$\mathit{TypeMem} = \mathit{Adr} \to_{fin} \mathit{TypeSem}$$

ou encore

$$\mathit{TypeSem} = (\mathit{Adr} \to_{fin} \mathit{TypeSem}) \to \mathcal{P}(\mathit{Val})$$

Un modèle sémantique des types références

$$\mathit{TypeSem} = (\mathit{Adr} \to_{\mathit{fin}} \mathit{TypeSem}) \to \mathcal{P}(\mathit{Val})$$

Pas de solution dans les ensembles ; sans doute une solution dans les domaines. Mais une fois de plus, le step-indexing / la modalité $\triangleright$ nous donnent une solution facile !

Accéder au contenu d'une référence ($!\ell$) consomme une étape de calcul.

Donc, le *TypeSem* associé à une adresse ℓ peut être ≪plus tard≫ et donc ≪moins précis≫ que le *TypeSem* que l'on cherche à construire.

Un modèle sémantique des types références

Avec un comptage de pas explicite, cela donne la famille de types

$$\begin{aligned} \mathit{TypeSem}_k &= \mathit{TypeMem}_k \to \mathcal{P}(\mathit{Val}) \\ \mathit{TypeMem}_0 &= \mathit{Adr} \to_{\mathit{fin}} \texttt{unit} \qquad \text{(arbitraire)} \\ \mathit{TypeMem}_{k+1} &= \mathit{Adr} \to_{\mathit{fin}} \mathit{TypeSem}_k \end{aligned}$$

C'est la solution de l'équation suivante exprimée dans la logique du topos des arbres :

$$\mathit{TypeSem} = (\mathit{Adr} \to_{\mathit{fin}} \triangleright \mathit{TypeSem}) \to \mathcal{P}(\mathit{Val})$$

Dans cette logique, on peut faire des récurrences de Löb sur des types quelconques, pas seulement sur des propositions logiques.

La relation logique unaire correspondante

$$
\begin{aligned}
V(\texttt{int})(M) &= \{\, n \mid n \text{ entier} \,\} \\
V(t\ \texttt{ref})(M) &= \{\, \ell \mid M(\ell)(\overline{M}) \subseteq \triangleright V(t)(\overline{M}) \,\} \\
V(t \to s)(M) &= \{\, \lambda x.\, a \mid \forall M' \sqsupseteq M,\ \forall v \in \triangleright V(t)(M'),\ a[x \leftarrow v] \in \triangleright E(s)(M') \,\} \\
[M] &= \{\, m \mid Dom(m) = Dom(M) \\
&\qquad \wedge \forall \ell \in Dom(m), m(\ell) \in M(\ell)(\overline{M}) \,\} \\
E(t)(M) &= \{\, a \mid \forall m \in [M], \\
&\qquad (\langle a,\, m\rangle \text{ irréductible} \Rightarrow \langle a,\, m\rangle \in V(t)(M)) \\
&\qquad \wedge (\forall b, \forall m',\ \langle a,\, m\rangle \to \langle b,\, m'\rangle \Rightarrow \\
&\qquad\qquad \exists M' \sqsupseteq M,\ m' \in [M'] \wedge b \in \triangleright E(t)(M')) \,\}
\end{aligned}
$$

On a noté $\overline{M}$ la troncature $\texttt{next}(M)$ avec $\texttt{next} : \forall A.\, A \to \triangleright A$.

Dans $M' \sqsupseteq M$, M' est ≪plus tard≫ que M, et donc $M' \sqsupseteq M$ est défini comme $Dom(M') \sqsupseteq Dom(M)$ et $M'(\ell) = \overline{M}(\ell)$ pour tout $\ell \in Dom(M)$.

Extension aux relations logiques binaires

Cette approche à base de types sémantiques pour la mémoire s'étend — non sans peine ! — aux relations logiques binaires et à des propriétés d'équivalence contextuelle. Voir en particulier :

- Ahmed, Dreyer, Rossberg. *State-Dependent Representation Independence*, POPL 2009.
- Dreyer, Neis, Rossberg, Birkedal. *A Relational Modal Logic for Higher-Order Stateful ADTs*. POPL 2010.

Exemple d'utilisation (Pitts & Stark, 1998) : montrer que les fonctions `up` et `down` sont indistinguables par le contexte

```
let up   = let c = ref 0 in fun () -> c := !c + 1; !c
let down = let c = ref 0 in fun () -> c := !c - 1; - !c
```

en interprétant les adresses ℓ_1, ℓ_2 des deux `c` par la relation $\{(n, -n) \mid n \geq 0\}$.

Pour aller plus loin

Un rapprochement progressif entre

- Les logiques de programmes pour les langages impératifs du premier ordre : logique de Hoare, logique de séparation, logique de séparation concurrente.
- Les relations logiques pour les langages d'ordre supérieur avec état mutable.

Un exemple récent de convergence : le système Iris, un cadre général pour les logiques de séparation concurrentes incluant également des modalités $\triangleright$ et $\Box$ pour traiter les aspects «ordre supérieur».
(`https://iris-project.org/`)

V

Bibliographie

Bibliographie

L'article fondateur pour les relations unaires :

- A. Appel et D. McAllester. *An indexed model of recursive types for foundational proof-carrying code*. TOPLAS 23(5), 2001.

Extension aux relations binaires :

- Amal Ahmed. *Step-Indexed Syntactic Logical Relations for Recursive and Quantified Types*. ESOP 2006.

Formulations à base de logiques modales :

- A. Appel, P.-A. Melliès, C. Richards, J. Vouillon. *A very modal model of a modern, major, general type system*. POPL 2007.
- D. Dreyer, A. Ahmed, L. Birkedal. *Logical Step-Indexed Logical Relations*. LMCS 7, 2011.

L'état de l'art en logiques de programmes pour langages impératifs et concurrents et d'ordre supérieur :

- Iris Project, *Tutorial Material*, `https://iris-project.org/tutorial-material.html`