Sémantique parallèle d'un langage fonctionnel indulgent

Bruno Malenfant et Guy Tremblay

Dépt. d'informatique, UQAM

Mai 1998

Aperçu

- Qu'est-ce qu'un langage fonctionnel indulgent?

  Pourquoi un tel langage est-il naturellement parallèle?

- Qu'est-ce que le π-calcul?

- De quelle façon peut-on spécifier la sémantique d'un langage indulgent à l'aide du π-calcul?

1. Introduction

Cadre de la recherche = Développement d'un compilateur pour `pH`

`pH` = `parallel Haskell`
= `Haskell`
\+ I/M-structures
\+ indulgence et parallélisme implicite

Machine cible = Architecture multi-contextes

- Machine MIMD avec mémoire distribuée
- Processeurs à contextes multiples
  ⇒ Tolérance de latence mémoire
  ⇒ Support pour parallélisme fin

2. Qu'est-ce qu'un lang. fonct. indulgent?

- Langage strict
  = Appel par valeur
  ⇒ Évaluation des arguments *avant* l'appel

- Langage non-strict
  ⇒ Peut produire un résultat même si les arguments ne sont pas encore évalués

  Exemple:

  ```
  f x y z = if x == 0 then 0 else y + z
  ```

- Deux classes de langages non-stricts
  1. Paresseux: *call-by-need*
  2. Indulgent: arguments et fonction évalués *en parallèle*

Deux exemples illustrant l'indulgence:

- Récursivité de structures de données

```
fib_array n =
  let A = Array (0, n) (
            (0, 1) :
            (1, 1) :
            [(i, A(i-1) + A(i-2)) | i <- [2..n]] )
  in A
```

- Parallélisme (de pipeline)

```
map f [] = []
map f (x : xs) = (f x) : (map f xs)

double x = 2*x;
inc x = x+1;

resultat = map double (map inc [1..100])
```


3. Qu'est-ce que le π-calcul?

Motivation/rôle du π-calcul

- *"Manière de décrire et d'analyser les systèmes composés d'agents qui interagissent entre eux et dont la configuration et l'environnement sont continuellement en changement"*
  [Milner 91]

- Principe très simple, de bas niveau

  $\approx$ λ-calcul des processus concurrents.

Syntaxe

$$
\begin{array}{lcl}
P, Q, T & = & \text{soit } \vec{x}\text{: } R \mid R \\
R & = & * \vec{\imath}\ S \mid \vec{\imath}\ S \\
S & = & \{P_1 \mid \ldots \mid P_n\} \mid \\
 & & \{[x_1 = y_1]P_1 + \ldots + [x_n = y_n]P_n\} \mid \\
 & & \text{erreur "mssg"} \mid \text{inert} \\
 & & x!\vec{y} \mid x! \oplus (\vec{y}) \\
\imath & = & x?\vec{y}.
\end{array}
$$

Règles de congruence structurelle

$$
\begin{array}{lcl}
P \mid Q & \equiv & Q \mid P \\
\{\{P \mid Q\} \mid T\} & \equiv & \{P \mid Q \mid T\} \\
P \mid \text{inert} & \equiv & P \\
\text{soit } \vec{x}\text{: inert} & \equiv & \text{inert} \\
* P & \equiv & P \mid * P \\
\text{soit } x\text{: } \{ P \mid Q \} & \equiv & \{ P \mid \text{soit } x\text{: } Q \} \\
 & & x \notin VL(P) \\
\text{soit } x\text{: } P & \equiv & \text{soit } y\text{: } P[y/x] \\
 & & y \notin VL(P)
\end{array}
$$

Sémantique opérationnelle

$$\text{COMM}: \quad x?\vec{y}.P \mid x!\vec{z} \longrightarrow P[\vec{z}/\vec{y}]$$

$$\text{CHOIX}: \quad \{\ldots + [x = y]P\} \longrightarrow P \qquad \text{si } x = y$$

$$\text{PAR}: \quad \frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q}$$

$$\text{RES}: \quad \frac{P \longrightarrow P'}{\text{soit } x\text{: } P \longrightarrow \text{soit } x\text{: } P'}$$

$$\text{STRUCT}: \quad \frac{Q \equiv P \ \wedge \ P \longrightarrow P' \ \wedge \ P' \equiv Q'}{Q \longrightarrow Q'}$$

4. Sémantique π-calcul d'un langage fonctionnel indulgent

4.1 Syntaxe abstraite du lang. fonct.

Programme ::= e

$$
\begin{array}{rcl}
e, e_1, e_h & ::= & v \\
& | & y \\
& | & \oplus(e_1, \ldots, e_h) \\
& | & \text{let } n_1 = e_1; \ldots; n_h = e_h \text{ in } e \\
& | & \lambda x_1, \ldots, x_h.e \\
& | & f(e_1, \ldots, e_h) \\
& | & K_i(e_1, \ldots, e_h) \\
& | & \text{case } y \text{ as } K_1(n_{11}, \ldots, n_{1a_1}) \text{ in } e_1; \ldots
\end{array}
$$

où

$$
\begin{array}{rcl}
v & \in & \text{Valeur de base. (Int, Float, ...)} \\
h & \in & \mathbf{N} \\
x_i, y, f, n_i & \in & \text{Identificateur} \\
K_i & \in & \text{Constructeur} \\
\oplus & \in & \text{Opérateur de base } (+, -, \ldots)
\end{array}
$$

4.2 Description de la sémantique

Fonction sémantique

$[\![_]\!]__ : \textbf{Terme} \to \textbf{Canal} \to \textbf{Processus}$

$$[\![v]\!]_o \triangleq o!v$$

$$[\![y]\!]_o \triangleq y!o$$

$$
\begin{array}{l}
[\![\oplus(e_1, \ldots, e_h)]\!]_o \triangleq \textbf{soit } o_1, \ldots, o_h: \\
\quad \{[\![e_1]\!]_{o_1} \mid \ldots \mid [\![e_h]\!]_{o_h} \\
\quad\quad | \\
\quad\quad o_1?v_1. \ldots . o_h?v_h.o! \oplus (v_1, \ldots, v_h) \\
\quad \}
\end{array}
$$

$$
\begin{array}{l}
\textbf{associer}(n, e) \stackrel{\text{def}}{=} \textbf{soit } o: \\
\quad \{[\![e]\!]_o \mid o?v.\{*n?p.p!v\}\}
\end{array}
$$

$$
\begin{array}{l}
\textbf{définir}(n_1 = e_1; \ldots; n_h = e_h) \stackrel{\text{def}}{=} \\
\quad \{\textbf{associer}(n_1, e_1) \mid \ldots \mid \textbf{associer}(n_h, e_h)\}
\end{array}
$$

$$
\begin{array}{l}
[\![\textbf{let } n_1 = e_1; \ldots; n_h = e_h \textbf{ in } e]\!]_o \triangleq \\
\quad \textbf{soit } n_1, \ldots, n_h: \\
\quad\quad \{\textbf{définir}(n_1 = e_1; \ldots; n_h = e_h) \\
\quad\quad\quad | \\
\quad\quad\quad [\![e]\!]_o \\
\quad\quad \}
\end{array}
$$

$$
\begin{array}{l}
[\![\lambda x_1, \ldots, x_h.e]\!]_o \triangleq \textbf{soit } l: \\
\quad \{o!l \\
\quad\quad | \\
\quad\quad *l? < r, x_1, \ldots, x_h > . \\
\quad\quad\quad \{[\![e]\!]_r\} \\
\quad \}
\end{array}
$$

$$
\begin{array}{l}
\textbf{activer}(f, [p_1, \ldots, p_h]) \stackrel{\text{def}}{=} \textbf{soit } b: \\
\quad \{f!b \\
\quad\quad | \\
\quad\quad b?c.c! < p_1, \ldots, p_h > \\
\quad \}
\end{array}
$$

$$
\begin{array}{l}
[\![f(e_1, \ldots, e_h)]\!]_o \triangleq \textbf{soit } y_1, \ldots, y_h: \\
\quad \{\textbf{activer}(f, [o, y_1, \ldots, y_h]) \\
\quad\quad | \\
\quad\quad \textbf{définir}(y_1 = e_1; \ldots; y_h = e_h) \\
\quad \}
\end{array}
$$

$[\![K_i(e_1,\ldots,e_h)]\!]_o \triangleq$ **soit** $c_1,\ldots,c_h,p$:
$\{$**définir**$(c_1 = e_1;\ldots;c_h = e_h)$
$\mid$
$o!p$
$\mid$
$*\,p?<c,v>.$
$\{c!K_i \mid v!(c_1,\ldots,c_h)\}$
$\}$

$[\![$**case** y **as** $K_1(n_{11},\ldots,n_{1a_1})$ **in** $e_1;\ldots]\!]_o \triangleq$
soit v,c:
$\{$**activer**$(y,[c,v])$
$\mid$
$c?K.$
$\{[K = K_1]v?(n_{11},\ldots,n_{1a_1}).\{[\![e_1]\!]_o\}$
$+\ldots+$
$[K = K_j]v?(n_{j1},\ldots,n_{ja_j}).\{[\![e_j]\!]_o\}$
$\}$
$\}$

5. Conclusion

Ce qui a été fait:

- Sémantique exprimant explicitement le parallélisme pour un langage indulgent

Travaux en cours et/ou futurs:

- Traitement des structures de données incrémentales avec synchronisation implicite de style producteur/consommateur (I-structures)
- Mise en oeuvre du π-calcul à l'aide des *threads* de `Java`
- Équivalence formelle entre sémantique opérationnelle de `kpH` et modèle π-calcul