Devoir de maison
Langages de programmation : traducteurs et sémantique
2006/2007

October 16, 2006

1 La machine abstraite AM

La machine AM est définie par un système de transition dont les configurations sont des triplets :

- Une liste $instr_1, \cdots, instr_n$ d'instructions. C'est le code qui reste à exécuter.
- Une pile qui est utilisée pour évaluer les expressions.
- La mémoire de la machine qui est décrite par un état; donc une fonction des variables vers $\mathbb{Z}$.

Nous allons définir l'ensemble des instructions et la relation de transition entre configurations

$$(c, p, m) \triangleright (c', p', m')$$

La machine AM n'a donc ni registres, ni accumulateur. C'est le sommet de la pile qui joue le role d'accumulateur et le reste de la pile celui de registres.

1.1 Les instructions

Instructions	Effet
push-n,True,False	empiler la constante n,**tt**,**ff**
fetch(x) store(x)	empiler la valeur de x dans l'état actuel dépiler le sommet de la pile et l'affecter à x
add sub,mult.and,le,equal,neg branch(c_1,c_2) loop(c_1,c_2) noop	remplacer les deux plus hauts éléments de la pile par leur somme similaire si le sommet est **tt** exécuter c_1 s'il est **ff** alors c_2, sinon blocage exécuter c_1, puis si le sommet de la pile est **tt**, exécuter c_2 suivie de loop(c_1,c_2)) sinon, si c'est **ff** s'arrêter skip

On définit les progammes cibles comme des mots sur l'alphabet des d'instructions. L'ensemble des programmes cibles est dénoté **Code**.
Une configuration de AM est un triplet (c, p, m) où c est un programme cible, p est le contenu de la pile qui est un mot sur $\mathbb{Z} \cup \mathbb{B}$ et m est un état dans **State** $= [\mathbf{Var} \to \mathbb{Z}]$.

1.2 La relation de transition-sémantique des instructions

La relation $\triangleright$ est inductivement définie par :

$(\mathsf{push\text{-}n} \cdot c, p, m)$	$\triangleright (c, n \cdot p, m)$	
$(\mathsf{True} \cdot c, p, m)$	$\triangleright (c, \mathbf{tt} \cdot p, m)$	
$(\mathsf{False} \cdot c, p, m)$	$\triangleright (c, \mathbf{ff} \cdot p, m)$	
$(\mathsf{fetch}(x) \cdot c, p, m)$	$\triangleright (c, m(x) \cdot p, m)$	
$(\mathsf{store}(x) \cdot c, v \cdot p, m)$	$\triangleright (c, p, m[x \mapsto v])$	si $v \in \mathbb{Z}$
$(\mathsf{add} \cdot c, v_1 \cdot v_2 \cdot p, m)$	$\triangleright (c, (v_1 + v_2) \cdot p, m)$	si $v_1, v_2 \in \mathbb{Z}$
$(\mathsf{sub} \cdot c, v_1 \cdot v_2 \cdot p, m)$	$\triangleright (c, (v_1 - v_2) \cdot p, m)$	si $v_1, v_2 \in \mathbb{Z}$
$(\mathsf{mult} \cdot c, v_1 \cdot v_2 \cdot p, m)$	$\triangleright (c, (v_1 * v_2) \cdot p, m)$	si $v_1, v_2 \in \mathbb{Z}$
$(\mathsf{le} \cdot c, v_1 \cdot v_2 \cdot p, m)$	$\triangleright (c, (v_1 \leq v_2) \cdot p, m)$	si $v_1, v_2 \in \mathbb{Z}$
$(\mathsf{equal} \cdot c, v_1 \cdot v_2 \cdot p, m)$	$\triangleright (c, (v_1 = v_2) \cdot p, m)$	si $v_1, v_2 \in \mathbb{Z}$
$(\mathsf{and} \cdot c, b_1 \cdot b_2 \cdot p, m)$	$\triangleright (c, (b_1 \wedge b_2) \cdot p, m)$	si $b_1, b_2 \in \mathbb{B}$
$(\mathsf{neg} \cdot c, b \cdot p, m)$	$\triangleright (c, (\neg b) \cdot p, m)$	si $b \in \mathbb{B}$
$(\mathsf{branch}(c_1, c_2) \cdot c, \mathbf{tt} \cdot p, m)$	$\triangleright (c_1 \cdot c, p, m)$	
$(\mathsf{branch}(c_1, c_2) \cdot c, \mathbf{ff} \cdot p, m)$	$\triangleright (c_2 \cdot c, p, m)$	
$(\mathsf{loop}(c_1, c_2) \cdot c, p, m)$	$\triangleright (c_1 \cdot \mathsf{branch}(c_2 \cdot \mathsf{loop}(c_1, c_2), \mathsf{noop}) \cdot c, p, m)$	
$(\mathsf{noop} \cdot c, p, m)$	$\triangleright (c, p, m)$	

Les configurations terminales sont de la forme (ϵ, p, m).

2 Quelques propriétés de AM

La sémantique de la machine AM a les propriétés suivantes :

- La relation de transition $\triangleright$ est déterministe :

$$(c, p, m) \triangleright (c_1, p_1, m_1) \wedge (c, p, m) \triangleright (c_2, p_2, m_2) \Rightarrow (c_1, p_1, m_1) = (c_2, p_2, m_2)$$

  Ceci peut être démontré par induction sur la longueur de c.

- "Extensibilité" du code et de la pile :

$$(c_1, p_1, m_1) \triangleright (c_2, p_2, m_2) \Rightarrow (c_1 \cdot c, p_1 \cdot p, m_1) \triangleright (c_2 \cdot c, p_2 \cdot p, m_2)$$

- Composabilité du code : Pour tout $k \geq 1$, si $(c_1 \cdot c_2, p, m) \triangleright^k (\epsilon, p_2, m_2)$ alors il existe $k' \in \mathbb{N}$ et une configuration (ϵ, p', m') tels que $(c_1, p, m) \triangleright^{k'} (\epsilon, p', m')$ et $(c_2, p', m') \triangleright^{k-k'} (\epsilon, p_2, m_2)$.

Démontrer la propriété de composabilité.

3 Génération de code

Compléter la définition des fonctions $\mathcal{CA}$: Aexp $\rightarrow$ Code, $\mathcal{CB}$: Bexp $\rightarrow$ Code et $\mathcal{CS}$: Stm $\rightarrow$ Code donnée en cours et rappelée ci-dessous :

- $\mathcal{CA}[n] = \mathsf{push\text{-}n}$ où n=$\mathcal{N}[n]$.
- $\mathcal{CA}[x] = \mathsf{fetch}(x)$
- $\mathcal{CA}[a_1 + a_2] = \mathcal{CA}[a_2] \cdot \mathcal{CA}[a_1] \cdot \mathsf{add}$
- $\mathcal{CB}[\mathsf{true}] = \mathsf{True}$

- $\mathcal{CS}[x := a] = \mathcal{CA}[a] \cdot \mathsf{store}(x)$
- $\mathcal{CS}[S_1; S_2] = \mathcal{CS}[S_1] \cdot \mathcal{CS}[S_2]$

de manière à avoir :

1. $(\mathcal{CA}[a], \epsilon, \sigma) \triangleright^* (\epsilon, \mathcal{A}[a]\sigma, \sigma)$
2. $(\mathcal{CB}[b], \epsilon, \sigma) \triangleright^* (\epsilon, \mathcal{B}[b]\sigma, \sigma)$
3. $(\mathcal{CS}[S], \epsilon, \sigma) \triangleright^* (\epsilon, p, \sigma')$ ssi $(S, \sigma) \to \sigma'$

4 Correction du générateur de code

Démontrer les propriétés suivantes :

1. $(\mathcal{CB}[b], \epsilon, \sigma) \triangleright^* (\epsilon, \mathcal{B}[b]\sigma, \sigma)$
2. **Si** $(\mathcal{CS}[S], \epsilon, \sigma) \triangleright^k (\epsilon, p, \sigma')$ **alors** $(S, \sigma) \to \sigma'$ **et** $p = \epsilon$**.**

Vérifier sans preuve formelle que votre définition satisfait les proprétés suivantes :

1. $(\mathcal{CA}[a], \epsilon, \sigma) \triangleright^* (\epsilon, \mathcal{A}[a]\sigma, \sigma)$
2. Si $(S, \sigma) \to \sigma'$ alors $(\mathcal{CS}[S], \epsilon, \sigma) \triangleright^* (\epsilon, p, \sigma')$.

En déduire

$$\mathcal{S}_{ns}[\,] = \mathcal{M} \circ \mathcal{CS}$$

5 Une machine abstraite avec des goto's

Les instructions loop et branch de la machine AM sont peut être de trop haut niveau. Nous introduisons la machine AM_1 dans laquelle elles sont remplacées par les nouvelles instructions label$-\ell$, jump$-\ell$ et jumpfalse$-\ell$.

Les configurations de cette machine sont des quadruplets

$$(pc, c, p, m)$$

où c, p et m sont comme pour la machine AM alors que $pc \in \mathbb{N}$ est le point de contrôle actuel. C'est-à-dire une position dans c qui indique l'instruction à exécuter.

Intuitivement, l'instruction label$-\ell$ définit un point de saut dans le code cible et sinon se comporte comme noop, jump$-\ell$ est un saut vers la marque $\ell \in \mathbb{N}$ et jumpfalse$-\ell$ est un saut conditionnel; la condition étant que **ff** est au sommet de la pile. La sémantique de jump$-\ell$ est donc

$$(pc, c, p, m) \triangleright (i, c, p, m) \text{si } c(pc) = \mathsf{jump}{-}\ell, i \in [0, \#c - 1] \text{ et } c(i) = \mathsf{label}{-}\ell$$

où $\#c$ est la longueur du mot c. Car rappelons-nous que c est un mot sur l'alphabet des instructions et rappelons-nous qu'un mot u est une fonction d'intervalle $[0, \#u - 1]$ de $\mathbb{N}$ vers son alphabet.

Définir la sématique de la machine AM_1.

6 Génération de code

Pour générer du code pour AM_1 il faut gérer les marques. Pour cela, il faut que la fonction $\mathcal{CS}$ non seulement prenne en argument le code source mais aussi l'ensemble des marques dejà utilisées. En résultat on obtient le code cible et l'ensemble des marques utilisées, donc sur-ensemble (ou égal) à l'ensemble en argument.

Définir la fonction $\mathcal{CS}$ pour la machine AM_1.