

Génération de code pour le langage While

On s'intéresse au langage `While`. Nous donnons ci-dessous une grammaire qui décrit des arbres abstraits. Le problème auquel on s'intéresse est le suivant :

A partir d'une arbre abstrait, générer de l'assembleur

1 Syntaxe abstraite

1.1 Catégories syntaxiques

<i>Métavariabiles</i>	<i>Catégories</i>	<i>Commentaires</i>
a	: AExp	expressions arithmétiques
b	: BExp	expressions booléennes
S	: Stm	instructions (ou commandes)

1.2 Syntaxe

$S \rightarrow x := a \mid \text{skip} \mid S; S \mid \text{If } b \text{ then } S \text{ else } S \mid \text{while } b \text{ do } S \text{ od}$
 $a \rightarrow n \mid x \mid a + a$
 $b \rightarrow \text{True} \mid \text{False} \mid \neg b \mid b \wedge b \mid a = a \mid a < a$

2 Machine M

Nous donnons, dans ce qui suit, une syntaxe d'une machine à registres, notés Ri. Les opérations arithmétiques et de comparaison mettent à jour les codes condition. L'opération `OPER` désigne les opérations arithmétiques et logiques. Le registre `CP` désigne le compteur programme, le registre `FP` désigne la base de l'environnement local et le registre `SP` le sommet de pile. Les instructions et adresses sont codées sur 4 octets, le pointeur de pile désigne le dernier emplacement occupé, on empile en décrémentant de 4 le registre `SP` et on dépile en incrémentant de 4 le registre `SP`. Les entiers sont codés sur 4 octets.

A chaque variable d'un programme, on associe une adresse de la forme `FP-depl`, où `depl` est calculé et mis dans la table des symboles.

`OPER Ri,Rj,Rk` Ri est le registre destination

`OPER Ri, Rk, val` Ri est le registre destination

`CMP Ri, Rj`

LD Ri, [adr]

ST Ri, [adr]

BRANCH label

SET Ri, label

CALL Ri

CALL label

RET

Les opérations arithmétiques OPER sont ADD, SUB, AND, OR.

Les opérations de branchement BRANCH sont BA, BEQ, BNE, BGT, BLT.

Les adresses sont décrites de la manière suivante :

adr ::= Ri + Rj | Ri + val | Ri | val

CALL label (resp. RET) est équivalent à :

CALL label	RET
ADD R1, CP, +4	ADD SP, SP, +4
ADD SP, SP, -4	LD CP, [SP+4]
ST R1, [SP]	
BA label	

2.1 Prologue /épilogue d'une procédure

prologue(Taille)	épilogue	empiler registre
ADD SP, SP, -4 ST FP, [SP] ADD FP, SP, 0 ADD SP, SP, -Taille	ADD SP, FP, 0 LD FP, [SP] ADD SP, SP, +4	ADD SP, SP, -4 ST R, [SP]

2.2 Passage de paramètres

Soit f une fonction à un paramètre y, situé à l'adresse FP-8.

Passage par Valeur	Passage par Adresse
LD R2, [FP-8] ADD SP, SP, -4 ST R2, [SP] CALL f ADD SP, SP, +4	ADD R2, FP, -8 ADD SP, SP, -4 ST R2, [SP] CALL f ADD SP, SP, +4
Passage par Résultat	Passage par Valeur-Résultat
ADD SP, SP, -4 CALL f LD R2, [SP] ST R2, [FP-8] ADD SP, SP, +4	LD R2, [FP-8] ADD SP, SP, -4 ST R2, [SP] CALL f LD R2, [SP] ST R2, [FP-8] ADD SP, SP, +4

3 Principe de génération de code

On va générer du code assembleur. Nous définissons trois fonctions de génération de code

$\text{GenCodeAExp} : \text{AExp} \rightarrow \text{Code}^* \times \mathbb{N}$
 $\text{GenCodeBExp} : \text{BExp} \times \text{Label} \times \text{Label} \rightarrow \text{Code}^*$
 $\text{GenCodeStm} : \text{Stm} \rightarrow \text{Code}^*$

où Code^* est l'ensemble des séquences de code 3 adresses, le séparateur de séquence étant $\|$.

Pour cela, on utilise les fonctions auxiliaires suivantes :

$\text{AllouerRegistre} : \rightarrow \mathbb{N}$
 $\text{nouvelleEtiquette} : \rightarrow \mathbb{N}$
 $\text{GetSymbDepl} : \mathbb{N} \rightarrow \mathbb{N}$

Expressions booléennes A chaque expression booléenne b , on associe deux étiquettes lvrai et lfaux qui sont attachées respectivement à l'instruction à exécuter si l'expression b est évaluée à vrai (resp. faux).

Expressions arithmétiques A chaque expression arithmétique a , on associe le registre contenant le résultat de l'opération.

4 Génération de code pour les instructions de contrôle

A chaque nœud de l'arbre abstrait, on associe du code de la manière suivante :

$\text{GenCodeStm} (x := a)$	=	Soit	$(C, i) = \text{GenCodeAExp}(a),$ $k = \text{GetSymbDepl}(x)$
		dans	$C \ \text{ST } Ri, [FP-k]$
$\text{GenCodeStm} (S_1, S_2)$	=	Soit	$C_1 = \text{GenCodeStm}(S_1),$ $C_2 = \text{GenCodeStm}(S_2)$
		dans	$C_1 \ C_2$
$\text{GenCodeStm} (\text{while } b \text{ do } S \text{ od})$	=	Soit	$\text{ldebut} = \text{nouvelleEtiquette}(),$ $\text{lvrai} = \text{nouvelleEtiquette}(),$ $\text{lfaux} = \text{nouvelleEtiquette}()$
		dans	$\text{ldebut} \ $ $\text{GenCodeBExp}(b, \text{lvrai}, \text{lfaux}) \ $ $\text{lvrai} \ $ $\text{GenCodeStm}(S) \ $ $\text{BA } \text{ldebut} \ $ $\text{lfaux}:$
$\text{GenCodeStm} (\text{If } b \text{ then } S_1 \text{ else } S_2)$	=	Soit	$\text{lsuivant} = \text{nouvelleEtiquette}(),$ $\text{lvrai} = \text{nouvelleEtiquette}(),$ $\text{lfaux} = \text{nouvelleEtiquette}()$
		dans	$\text{GenCodeBExp}(b, \text{lvrai}, \text{lfaux}) \ $ $\text{lvrai}:$ $\text{GenCodeStm}(S_1) \ $ $\text{BA } \text{lsuivant} \ $ $\text{lfaux} \ $ $\text{GenCodeStm}(S_2) \ $ $\text{lsuivant}:$

5 Génération de code pour les expressions arithmétiques

GenCodeAExp(x)	=	Soit	i=AllouerRegistre() k=GetSymbDepl(x)
		dans	(LD Ri,[FP-k],i)
GenCodeAExp(n)	=	Soit	i=AllouerRegistre()
		dans	(ADD Ri,R0,v,i) (où $v=\mathcal{N}(n)$)
GenCodeAExp(a ₁ + a ₂)	=	Soit	(C ₁ ,i ₁)=GenCodeAExp(a ₁), (C ₂ ,i ₂)=GenCodeAExp(a ₂), k=AllouerRegistre()
		dans	(C ₁ C ₂ ADD Rk, Ri ₁ ,Ri ₂ ,k)

6 Génération de code pour les expressions booléennes

GenCodeBExp (a ₁ = a ₂ ,lvrai,lfaux)	=	Soit	(C ₁ ,i ₁)=GenCodeAExp(a ₁), (C ₂ ,i ₂)=GenCodeAExp(a ₂),
		dans	C ₁ C ₂ CMP Ri ₁ , Ri ₂ BEQ lvrai BA lfaux
GenCodeBExp (b ₁ ∧ b ₂ ,lvrai,lfaux)	=	Soit	l=nouvelleEtiq()
		dans	GenCodeBExp(b ₁ ,l,lfaux) l: GenCodeBExp(b ₂ ,lvrai,lfaux)
GenCodeBExp(¬ b,lvrai,lfaux)	=		GenCodeBExp(b,lfaux,lvrai)

7 La pile

On suppose que les variables et les paramètres sont représentés sur 4 octets.

...	0
var _n	←SP, FP- 4*n
...	
var ₁	←FP-4
Chaînage dynamique	←FP
Adresse de retour	←FP+4
Chaînage statique	←FP+8
Param _n	←FP+12
...	
Param ₁	←FP+8+4*n