

VM ZAM

Le corps de la ZAM (interp.ml)

Types : Nombre : Entier / Pointeur / Labels

Pointeur : Entier Pair tous distinct.

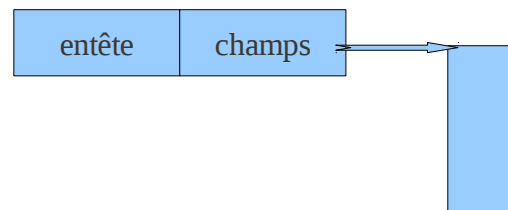
Générateur

gen_ptr : Pointeur

Fonction de verification de type

outTypes : type

Blocs : {entête : Int ; champs : tab<Int>}



Outils : Tas / Globals / extra_args

Tas : Hash<int, blocs> (51)

Allocation d'une structure sur le tas. Param : entete, champs

Entete, champs sont un type bloc

out : Pointeur, clef du bloc inséré dans le tas ##

alloc_bloc(Int, Nombre) : Pointeur

##Allocation d'une constante ##

alloc_const (n : Entier | ptr : Pointeur | (n, constlist)) : (Entier | Pointeur | Pointeur sur constlist dans tas)

Globals : Hash<int, Nombre>(89) *référéncé par env*

Accesseur de variables globales. Param : clef

getglobal (Int) : Nombre

Ajouter une variable global. Param : clef, valeur

setglobal(Int ; Nombre) : Unit

Accesseur d'un bloc ? Param : Pointeur

bloc (ptr) : Bloc

Extra_args : Un registre

Pile & Accu :

Accu seulement ref

pile : ModifStack.creat _any

Ajouter dans la pile Param : v

push (v) : unit

enleve la de tête de pile (pop)

pop() : v

enleve n fois la tete de pile (popn)

Param : n, le nombre de pop ##

popn (n) : v

reinitialiser

reset() : unit

getteur au n element de la pile

Param n, iteration ##

get(n) : v

remplacer le n° element de la pile

Param n, l'iteration

v, element ##

set(n, v) : unit

Fonctions auxiliaires :

Permet les opérations arithmétiques

Ces opérations entrent le résultat dans l'accumulateur

Op unaires

Param f, la fonction

Opération Unaire sur entier dans l'accumulateur ##

do_arith_unop (f) : unit

Op Binaires

Param f, la fonction

Operation binaire entre l'accumulateur et la tête de pile ##

do_arith_unop (f) : Unit

Comparaison Binaires

Param f, la fonction

Idem qu'une operation Binaire hormis quelle renvoie le resultat ##

do_arith_unop (f) : Entier (boolean)

Labels : Hashtbl.create 51

Les labels marquent chaque instructions (klabel) du code en cours

variable max_label

##Initialisés au début d'une execution

Si l'instruction est Klabel max_label = sa valeur et elle est associée à l'instruction dans la table de hachage

Param : code, le code à exécuter ##

prepare_labels (code) : unit

Creation d'un label sur une instruction

param pc, le pointeur de l'instruction ##

set_label (pc) : unit

Interpreteur :

Il y a deux types d'instructions, les simples et les complexes

Les instructions ne changent pas le flot d'exécution du programmes ##

do_simpleinstr (instruction)

Les instructions simples sont :

Klabel	()
Knop	()
Kreadint	lire un entier et le placer dans l'accumulateur
Kreadstring	Non supporte
Kprintint	Affiche l'entier dans l'accumulateur
Kprintstring	Non supporte
Kgetglobal n (clef)	place la valeur de la global associée à n dans l'accumulateur
Ksetglobal n (clef)	place la valeur de l'accumulateur en global et l'associe à n (accu=unit)
Kacc n	place le n° elem de la pile dans l'accumulateur
Kpush	place la valeur de l'accumulateur dans la pile
Kpop n	enlève n valeur dans la pile (popn)
Kassign n	place dans le n° elem de la pile la valeur de l'accumulateur
Kconst cst	allouer une constante, valeur dans placee dans l'accumulateur
Kmakeblock size, tag	allouer un tableau (tab) de taille size du type dans l'accumulateur, il est rempli par les size champs de la pile (dans l'ordre de pop). Le bloc cree a un entete tag et un champs tab. Le pointeur est ensuite placé dans l'accumulateur.
Kgetfield n	Accès au n° champs du bloc pointé par l'accumulateur. Résultat dans l'accumulateur.
Ksetfield n	Place au n° champs du bloc pointé par l'accumulateur, la valeur de la tête de pile (pop). L'accumulateur est mis à unit.
Kvectlength	Entre la taille du bloc pointé par l'accumulateur dans l'accumulateur
Kgetvectitem (arg en pile)	Place le n° element du bloc pointé par l'accumulateur dans l'accumulateur. N est pris en tete de pile (pop).
Ksetvectitem (args en pile)	Place dans le n°element du bloc pointé par l'accumulateur la valeur v. L'accumulateur est mis à unit et n et v sont pris dans la tete de pile (dans cette ordre).
Instruction d'operation	(Voir plus haut) Les arguments sont faites entre l'accumulateur et la tete de pile. Le résultat est ensuite placé dans l'accumulateur.
Knegint	~- Operation unaire de negation
Kaddint	+ Operation binaire d'addition
Ksubint	- Operation binaire de soustraction
Kmulint	* Operation binaire de multiplication
Kdivint	/ Operation binaire d'addition
Kmodint	mod Operation binaire de modulo

Kandint	land	Operation binaire and
Korint	lor	Operation binaire or
Kxorint	lxor	Operation binaire xor
Klslint	lsl	Operation binaire decalage à gauche
Klsrint	lsr	Operation binaire decalage à droite
Kasrint	asr	Operation binaire decalage à droite arithmétique
Koffsetint n	Execute n fois + unaire ?	
Kboolnot	!	Non booleen
Koffsetref n	Ajoute n au premier champs du bloc pointé par l'accumulateur et le place à unit	
Operation de comparaison		
Kintcomp comp	Même fonctionnement que pour les opération binaire comp peut etre :	
Ceq	=	
Cneq	<>	
Clf	<	
Cgt	>	
Cle	<=	
Cge	>=	
Kisint	Teste si la valeur de l'accumulateur est un entier. Place _true (1) si oui _false (0) sinon	
Kenvacc n	Place dans l'accumulateur la valeur du n° champs du bloc pointé par l'env (n° variables globales).	
Kclosure lbl, n	Création de fermeture. Un bloc tagué 247 comprenant un tableau de n valeurs. L'accumulateur est placé dans champs[1] (si n <> 0) Les champs de 2 à n sont rempli par la tête de pile (n-1 pop) L'accumulateur prend le pointeur du bloc de fermeture créé.	
Kclosurerec lblist, n	(Non géré, idem que closure, l'accu est seulement ajouté dans la pile)	
Kpush_retaddr l	Met extra_args, env, un label dans la pile	
Krestart	Restauration du contexte	
	Place les variables globales (env) dans la pile (sauf le 1 qui est mis dans env). On ajoute ensuite le nombre d'argument à extra_args.	
Koffsetclosure n	Place le pointeur d'environnement (env) dans l'accumulateur. (les fonctions rec n'étants pas prises en compte)	

Les instructions complexes :

Kbranch l
 Kbranchif l
 Kstrictbranchif l
 Kbranchifnot l
 Kstrictbranchifnot l
 Kswitch tbl_const tbl_bloc
 Kapply nbargs
 Kappterm (nbargs, slotsize)
 Kgrab n
 Kstop
 Kreturn slotsize
 i

execution du code sous forme de suite d'instructions

la fonction comporte plusieurs variables ##

`execute(code) : Unit`

##La derniere instruction executee##

last

Execution des instructions complexes :

Param pc, le pointeur de l'instruction courante ##

do_instr (pc)

##Lancement de la ZAM##

-v ou -verbose fichier

ou fichier

##Lancement du traitement##

Interp.execute (Parser.programme Lexer.token (Lexing.from_channel f))

Les Instructions des programmes : (lexer.mll / parser.mly)

L'analyseur syntaxique et lexical de ZAM. Ils définissent les mots clés et la syntaxe que doit avoir un programme pour être compris par la ZAM.

Lexer :

Le lexer est celui qui indique les mots clefs du langage. Ce que la ZAM reconnaît et transforme en token pour le parser.

La table keyword_table est une table d'association entre instructions dans le langage du programme au token du lexer.

Le lexer de ZAM reonnait : ##

Les chiffres : 0-9

les lettres : a-zA-Z

les alphanumeriques : digit | alpha | _ | '

les identifiants : alpha+

les entier : (-?) digit+

debut de commentaire ((*) et les fin de commentaire (*))

Traitements associe

Les séparateurs sont ignorés (' \t \n)

comment|commentlevel : Gère le profondeur de commentaire

Plusieurs niveau de commentaires sont gérés, ils sont aussi ignorés.

les autres entré sont traitées selon ce qu'elles sont les entiers sont transformé de string en entier pour le parser

Entrée

identifiant

L entier

L entier :

[entier]

[entier :

]

entier a

entier

«

'

,

/

instructions

instruction reconnue de la table (voir plus bas)

Tlabel(entier)

Tdef_label(entier)

Tatom(entier)

Topen_block(entier)

Tclose_block

Tnum(entier)

Tnum(entier)

Tstring

Tchar

Tcomma

Tslash

eof TEOF
D'autres entrées entraîneront une levée d'exception

Parser :

Le programme doit commencer par instructions et finir par TEOF (end of file).

Il est de type instruction array

Les instructions sont transformées en tableau et renversées.##

programme : instructions TEOF { Array.of_list (List.rev \$1)}

Une instruction peut être vide ou une instruction suivie d'une instruction

Une file est alors créée avec instruction :: instructions (d'où le reverse) ##

instructions:

{[]}
| instructions instruction {\$2::\$1}
;

Les instructions : Le parser passe des tokens du lexer au langage de la machine ZAM. On voit ainsi la transformation des instructions.

Un tableau correspondant aux transformations du langage au lexer à la ZAM sera fait plus bas. Certaines instructions ont besoin d'une forme spéciale ##

num_list : est une liste de nombre. Le token nombre étant Tnum il est décrit ainsi

num_list : { [] }
| num_list Tnum {\$2::\$1}
;

num_ne_list : est aussi une liste de nombre mais ne pouvant être vide

num_ne_list : Tnum { [\$1] }
| num_list Tnum {\$2::\$1}
;

constant : Syntaxe pour création de constante

constant : Tnum { Const_char \$1 }
| Tatom { Const_block (\$1, []) }
| Topen_block constant_list Tclose_block { Const_block (\$1, List.rev \$2) }
;

une liste de constante

constant_list : {[]}
| constant_list constant { \$2::\$1 }
;

Tableau de coïncidence langage | token | ZAM

L entier	TLabel	
L entier :	Tdef_label	Klabel \$1
nop	Tnop	Knop
stop	Tstop	Kstop
readint	Treadint	Kreadint
readstring	Treadstring	Kreadstring
printint	Tprintint	Kprintint
printstring	Tprintstring	Kprintstring
getglobal entier	Tgetglobal Tnum	Ksetglobal \$2

setglobal entier	Tsetglobal Tnum	Ksetglobal \$2
acc entier	Tacc Tnum	Kacc \$2
envacc entier	Tenvacc Tnum	Kenvacc \$2
push	Tpush	Kpush
pop entier	Tpop Tnum	kpop \$2
assign entier	Tassign Tnum	Kassign \$2
push_retaddr label	Tpush_retaddr Tlabel	Kpush_retaddr \$2
apply entier	Tapply Tnum	Kapply \$2
appterm entier , entier	Tappterm Tnum Tcomma Tnum	Kappterm(\$2, \$4)
return entier	Treturn Tnum	Kreturn \$2
restart	Trestart	Krestart
grab entier	Tgrab Tnum	Kgrab \$2
closure label , entier	Tclosure Tlabel Tcomma Tnum	Kclosure(\$2,\$4)
closurerec entier entier ...	Tclosurerec num_ne_list Tcomma Tnum	Kclosurerec (list.rev \$2, \$4)
offsetclosure entier	Toffsetclosure Tnum	Koffsetclosure \$2
offsetref entier	Toffsetref Tnum	Koffsetref \$2
const constant	Tconst constant	Kconst \$2
makeblock entier , entier	Tmakeblock Tnum Tcoma Tnum	Kmakeblock(\$2,\$4)
getfield entier	Tgetfield Tnum	Kgetfield \$2
setfield entier	Tsetfield Tnum	Ksetfield \$2
vectlength	Tvectlength	Kvectlength
getvectitem	Tgetvectitem	Kgetvectitem
setvectitem	Tsetvectitem	Ksetvectitem
branch label	Tbranch Tlabel	Kbranch \$2
branchif label	Tbranchif Tlabel	Kbranchif \$2
branchifnot label	Tbranchifnot Tlabel	Kbranchifnot \$2
strictbranchif label	Tstrictbranchif Tlabel	Kstrictbranchifnot \$2
strictbranchifnot label	Tstrictbranchifnot Tlabel	Kstrictbranchifnot \$2
switch entier,entier \ entier,entier	Tswitch num_list Tslash num_list	Kswitch (list.rev\$2, list.rev\$4)
boolnot	Tboolnot	Kboolnot
negint	Tnegint	Knegint
addint	Taddint	Kaddint
subint	Tsubint	Ksubint
mulint	Tmulint	Kmulint
divint	Tdivint	Kdivint
modint	Tmodint	Kmodint
andint	Tandint	Kandint
orint	Torint	Korint
xorint	Txorint	Kxorint
lslint	Tlslint	Klslint
lsrint	Tlsrint	Klsrint
asrint	Tasrint	Kasrint
offsetint entier	Toffsetint Tnum	Koffsetint \$2
isint	Tisint	Kisint
eqint	Tcomparison(Ceq)	Kintcomp \$1
neqint	Tcomparison(Cneq)	Kintcomp \$1
ltint	Tcomparison(Clt)	Kintcomp \$1
gtint	Tcomparison(Cgt)	Kintcomp \$1
leint	Tcomparison(Cle)	Kintcomp \$1
geint	Tcomparison(Cge)	Kintcomp \$1