ANNEXE TECHNIQUE
Schéma de traduction

Nous attestons que ce travail est original, qu'il indique de façon appropriée tous les emprunts, et qu'il fait référence de façon appropriée à chaque source utilisée.

GICQUEL Antoine, CHAPDELAINE Julie, DAGUIN Cléo, TESTIER Thomas, CHIHABY Yasser, LAAFOUDI Amine

ESIR 2 - SI
26/01/2020

ECOLE SUPERIEURE D'INGENIEURS DE RENNES

UNIVERSITÉ DE RENNES 1

Source WHILE	Code intermédiaire

if **E1** then
  C1
else
  C2
fi
	{ **E.place** = ε
	  E.true = newLabel();
	  E.false = newLabel();
	  E.end = newLabel();
	  E.code = E1.code(E.true, E.false) ·
	  <label E.true, _, _, _> ·
	  C1.code ·
	  <goto E.end, _, _, _> ·
	  <label E.false, _, _, _> ·
	  C2.code ·
	  <label E.end, _, _, _> }

while **E1** do
  C
od
	{ **E.place** = ε
	  E.body = newLabel();
	  E.ifFalse = newLabel();
	  E.cond = newLabel();
	  E.code = <label E.cond, _, _, _> ·
	  E1.code(E.body, E.ifFalse) ·
	  <label E.body, _, _, _> ·
	  C.code ·
	  <goto E.cond, _, _, _> ·
	  <label E.ifFalse, _, _, _> }

nop	{ **E.place** = ε **E.code** = <nop, _, _, _> }

V1,**Vn** := **E1**,**En**	{ **E.place** = ε
	  E.code = E1.code ·
	  <write, _, E1.place, _> ·
	  En.code ·
	  <write, _, En.place, _> ·
	  <read, Vn, _, _> ·
	  <read, V1, _, _> }

for **E1** do
  C
od
	{ **E.place** = newVariable();
	  E.body = newLabel();
	  E.end = newLabel();
	  E.code = <write, _, E1.place, _> ·
	  <read, E.place, _, _> ·
	  E1.code ·
	  <label E.body, _, _, _> ·
	  <iff E.end, _, E.place, _> ·

	C.code · <tl, E.place, E.place, _> · <goto E.body, _, _, _> } <label E.end, _, _, _> }

foreach **V1** in **E1**
do
  C
od
	{ **E.place** = newVariable();
	  E.exp = TS(V1);
	  E.body = newLabel();
	  E.end = newLabel();
	  E.code = <write, _, E1.place, _> ·
	<read, E.exp, _, _> ·
	E1.code ·
	<label E.body, _, _, _> ·
	<iff E.end, _, E.exp, _> ·
	C.code ·
	<hd, E.place, E.exp, _> ·
	<tl, E.exp, E.exp, _> ·
	<goto E.body, _, _, _> }
	<label E.end, _, _, _> }

E1 and … and **En**	{ **E.place** = newVariable();
	  E.ifTrue = newLabel();
	  E.ifFalse = newLabel();
	  E.code = E1.code(E.ifTrue, E.ifFalse) ·
	<iff E.ifFalse, _, E1.place, _> ·
	En.code(E.ifTrue, E.ifFalse) ·
	<iff E.ifFalse, _, En.place, _> ·
	<label E.ifTrue, _, _, _> ·
	<true, E.place, _, _> ·
	<goto E.true, _, _, _> ·
	<label E.ifFalse, _, _, _> ·
	<false, E.place, _, _> ·
	<goto E.false, _, _, _> }

E1 or … or **En**	{ **E.place** = newVariable();
	  E.ifTrue = newLabel();
	  E.ifFalse = newLabel();
	  E.end = newLabel();
	  E.code = E1.code(E.ifTrue, E.ifFalse) ·
	<ift E.ifTrue, _, E1.place, _> ·
	En.code(E.ifTrue, E.ifFalse) ·
	<ift E.ifTrue, _, En.place, _> ·
	<goto E.ifFalse, _, _, _> ·
	<label E.ifTrue, _, _, _> ·
	<true, E.place, _, _> ·

	<goto E.end, _, _, _> · <label E.ifFalse, _, _, _> · <false, E.place, _, _> · <label E.end, _, _, _> }
not **E1**	{ **E.place** = newVariable(); E.ifTrue = newLabel(); E.end = newLabel(); **E.code** = E1.code(E.true, E.false) · <ift E.ifTrue, _, E1.place, _> · <true, E.place, _, _> · <goto E.end, _, _, _> · <label E.ifTrue, _, _, _> · <false, E.place, _, _> · <label E.end, _, _, _> }
E1 =? **E2**	{ **E.place** = newVariable(); E.ifEqual = newLabel(); E.end = newLabel(); **E.code** = E1.code(E.true, E.false) · E2.code(E.true, E.false) · <ifeq E.ifEqual,_, E1.place, E2.place> · <false, E.place, _, _> · <goto E.end, _, _, _> · <label E.ifEqual, _, _, _> · <true, E.place, _, _> · <label E.end, _, _, _> }
E → nil	{ **E.place** = newVariable(); **E.code** = <nil, E.place, _, _> }
E → symb	{ **E.place** = TS(symb); **E.code** = ε }
E → var	{ **E.place** = TS(var); Si expression booléenne : **E.code** = <ift E.true, _, E.place, _> · <goto E.false, _, _, _> Sinon : **E.code** = ε }
cons **E1** … **En**	Si n = 1 : { **E.code** = E1.code } Sinon si n >= 2 : { **E.place** = newVariable();

	E.code = E1.code() · E2.code() · <cons, E.place, E2.place, E1.place> · En.code() · <cons, E.place, En.place, E.place> }
list **E1** … **En**	{ **E.place** = newVariable(); E.nil = newVariable(); **E.code** = <nil, E.nil, _, _> E1.code() · <cons, E.place, E1.place, E.nil> · En.code() · <cons, E.place, En.place, E.place> }
hd **E1**	{ **E.place** = newVariable(); **E.code** = E1.code() · <hd, E.place, E1.place, _> }
tl **E1**	{ **E.place** = newVariable(); **E.code** = E1.code() · <tl, E.place, E1.place, _> }
call **func A1 An**	{ **E.place** = newVariable(); **E.code** = A1.code() · An.code() · <write, _, An.place, _> · <write, _, A1.place, _> · <call func, _, _, _> · <read, E.place, _, _> }

Code intermédiaire	Code C++
<nop, _, _, _>	bin_tree::nop();
<symb S1, D, _, _>	D = make_shared<bin_tree>(S1);
<read, D, _, _>	D = f_stack.top(); f_stack.pop();
<write, _, A1, _>	f_stack.push(A1);
<goto LAB, _, _, _>	goto LAB;
<label LAB, _, _, _>	LAB :

<ifeq LAB, _, A1, A2>	if(bin_tree::equals(A1, A2))
	{
	    goto LAB;
	}

<ift LAB, _, A1, _>	if(A1->isTrue())
	{
	    goto LAB;
	}

<iff LAB, _, A1, _>	if(A1->isFalse())
	{
	    goto LAB;
	}

<true, D, _, _>	D = bin_tree::getTrue();
<false, D, _, _>	D = bin_tree::getFalse();
<nil, D, _, _>	D = bin_tree::nil();
<cons, D, A1, A2>	D = bin_tree::cons(A1, A2);
<hd, D, A1, _>	D = bin_tree::hd(A1);
<tl, D, A1, _>	D = bin_tree::tl(A1);
<call func, _, _, _>	func(f_stack);