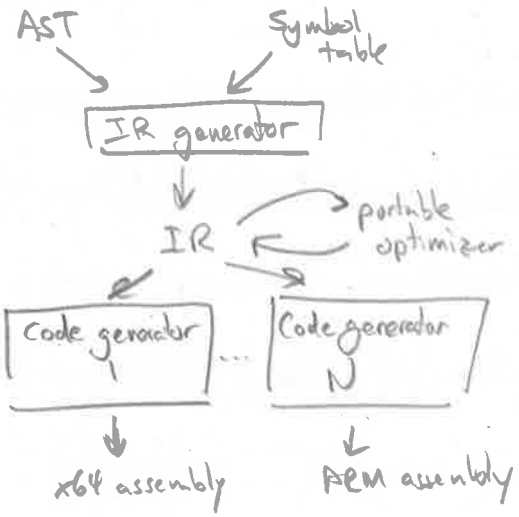
Intermediate-code generation pg. 1

Lecture topics

I. Intermediate representation
II. Expressions
III. Statements
IV. Jumping code
V. Calls and returns
VI. Memory accesses
VII. Literals

I. Intermediate representation



Variety of choices for IR:
- 3-address code
- SSA form
- stack-based vs. register-based

This class:
- 3-address code
- "infinite" number of virtual registers

IR for Tack: see handout.

II Expressions

Example TACK:

$x = (y + 2) * z$

IR:

$t_1 = y + 2$
$t_2 = t_1 * z$
$x = t_2$

Idea: use synthesized attributes code, addr

SDD (see Dragon book Fig. 6.19 / p. 379

$S \to ID = E.$	$S.code = E.code \parallel ID = E.addr$
$E \to E_1$ infixOp E_2	$E.addr = new\ TempAddr()$ $E.code = E_1.code \parallel E_2.code$ $\parallel E.addr = E_1.addr$ infixOp $E_2.addr$
$\mid$ prefixOp E_1	$E.addr = new\ TempAddr()$ $E.code = E_1.code$ $\parallel E.addr =$ prefixOp $E_1.addr$
$\mid (E_1)$	$E.addr = E_1.addr$
$\mid ID$	$E.addr = new\ NameAddr(ID)$

III Statements

Example Tack

```
if x {
    y := 1;
} else {
    y := 2;
}
z = y;
```

IR

```
if false x goto l2.
l1: y = 1;
    goto l3
l2: y = 2
l3: z = y
```

Idea: use inherited attributes for labels

AST Node	Attribute	Example
Stmt	next	l_3
Expr (boolean)	true, false	l_1, l_2

Intermediate Code generation pg. 2

SDD (see Dragon book Fig. 6.36 / p. 402)

$B \to \{ S_1 \dots S_n \}$	$S_1.next = new\ Label()$ ... $S_{n-1}.next = new\ Label()$ $S_n.next = B.next$ $B.code = S_1.code$ $\quad \| \| S_1.next : S_2.code$ $\quad \| \| \dots$ $\quad \| \| S_{n-1}.next : S_n.code$
$S \to B \mid I \mid W \mid A$	
$I \to if\ E\ B_1\ else\ B_2$	$E.true = new\ Label()$ $E.false = new\ Label()$ $B_1.next = B_2.next = I.next$ $I.code = E.code$ $\quad \| \| E.true : B_1.code$ $\quad \| \| goto\ I.next$ $\quad \| \| E.false : B_2.code$
$W \to while\ E\ B$	$E.true = new\ Label()$ $E.false = W.next$ $B.next = new\ Label()$ $W.code = B.next : E.code$ $\quad \| \| E.true : B.code$ $\quad \| \| goto\ B.next$

IV Jumping code

Two ways to translate boolean expressions

	Jumping code	Value code
Context	Control statement	Assignment
	other boolean expression	other non-boolean expression
Attributes	true/false label (inherited)	addr (synthesized)
	code (synthesized)	code (synthesized)

Example TAC

```
if a&&b == 0 {
    y := 1
} else {
    y := 2
}
z = y
```

IR

```
if False a goto l2
if b==0 goto l1
goto l2
l1: y=1
    goto l3
l2: y=2
l3: z=y
```

SDD (see Dragon book Fig 6.37 / p. 404)

$E \to E_1 \| E_2$	$E_1.true = E.true$ $E_1.false = new\ Label()$ $E_2.true = E.true$ $E_2.false = E.false$ $E.code = E_1.code$ $\quad \| \| E_1.false : E_2.code$
$\mid E_1 \&\& E_2$	$E_1.true = new\ Label()$ $E_1.false = E.false$ $E_2.true = E.true$ $E_2.false = E.false$ $E.code = E_1.code$ $\quad \| \| E_1.true : E_2.code$
$\mid !\, E_1$	$E_1.true = E.false$ $E_1.false = E.true$ $E.code = E_1.code$
$\mid E_1\ relop\ E_2$	$E.code = E_1.code$ $\quad \| \| E_2.code$ $\quad \| \| if\ E_1.addr\ relop\ E_2.addr\ goto\ E.true$ $\quad \| \| goto\ E.false$
$\mid true$	$E.code = goto\ E.true$
$\mid false$	$E.code = goto\ E.false$

Sometimes, need to convert:

given	needed	example
jumping code	value code	$x = y \| \| z$
value code	jumping code	$if\ x.f\ \{ \dots \}$

Given jumping code, generate value code:

$E \to \dots$	$E.true = new\ Label()$; $E.false = new\ Label()$; $jcode = genJumpingCode(E)$ $E.addr = new\ TempAddr()$; $E.code = E.addr = true$ $\quad \| \| jcode$ $\quad \| \| E.false : E.addr = false$ $\quad \| \| E.true :$

Given value code, generate jumping code

$E \to \dots$	$vcode = genValueCode(E)$ $E.code = vcode$ $\quad \| \| if\ E.addr\ goto\ E.true$ $\quad \| \| goto\ E.false$

V Calls and returns

Example TACK

$x = f(E_0, \ldots E_{n-1})$

IR

E_0.code
...
E_{n-1} code
param [0:n] = E_0.addr
...
param [n-1 :n] = E_{n-1}.addr
x = call f:n

Example TACK

→ E ;

IR

E.code
return E.addr

VI Memory accesses

Two ways to translate array/record accesses

	L-value	R-value
Context	left-hand-side of assignment	anywhere else
Attributes	addr refers to storage location	addr refers to storage contents
Dereference	no	yes

TACK has "array of arrays";
not true "multi-dimensional arrays".
easier to translate than in Dragon book

Example TACK

IR

x = a[i] [j] [k];

$t_1 = a[i]$
$t_2 = t_1[j]$
$t_3 = t_2[k]$

a [i] [j] [k] = x;

$t_1 = a[i]$
$t_2 = t_1[j]$
$t_2[k] = x$

x = r.f.g.h

$t_1 = r.f$
$t_2 = t_1.g$
$x = t_2.h$

r.f.g.h = x;

$t_1 = r.f$
$t_2 = t_1.g$
$t_2.h = x$

SDD

$S \to ID = E$	S.code = E.code || ID = E.addr
| $E_1 [E_2] := E_3$	S.code = E_3.code || E_1.code || E_2.code || E_1.addr [E_2.addr] = E_3.addr
| E_1.ID := E_2	S.code = E_2.code || E_1.code || E_1.addr.ID = E_2.addr
$E \to ID$	E.addr = new NameAddr(ID)
| $E_1 [E_2]$	E.addr = new TempAddr() E.code = E_1.code || E_2.code || E.addr = E_1.addr [E_2.addr]
| E.ID	E.addr = new TempAddr() E.code = E_1.code || E.addr = E_1.addr.ID

VII Literals

Example TACK

$a = [E_0, \ldots, E_{n-1}]$

IR

param [0:2] = sizeof(...)
param [1:2] = n
a = call newArray, 2
E_0.code
a [0] = E_0.addr
...
E_{n-1}.code
a [n-1] = E_{n-1}.addr

Example TACK

$r = (ID_0 = E_0, \ldots, ID_{n-1} = E_{n-1})$

IR

param [0:1] = sizeof(...)
r = call newRecord,
E_0.code
r.ID_0 = E_0.addr
...
E_{n-1}.code
r.ID_{n-1} = E_{n-1}.addr