Compilerconstructie

najaar 2013

http://www.liacs.nl/home/rvvliet/coco/

Rudy van Vliet
kamer 124 Snellius, tel. 071-527 5777
rvvliet(at)liacs(dot)nl

werkcollege 6, dinsdag 22 oktober 2013

Intermediate Code Generation

Translation of Array References

$S \rightarrow \textbf{id} = E;$ { *gen*(*top.get*(**id**.*lexeme*) ′ =′ E.*addr*); }

$S \rightarrow L = E;$ { *gen*(L.*array*.*base* ′[′L.*addr* ′]′ ′ =′ E.*addr*); }

$E \rightarrow E_1 + E_2$ { E.*addr* = **new** *Temp*();
gen(E.*addr* ′ =′ E_1.*addr* ′ +′ E_2.*addr*); }

$E \rightarrow \textbf{id}$ { E.*addr* = *top.get*(**id**.*lexeme*); }

$E \rightarrow L$ { E.*addr* = **new** *Temp*();
gen(E.*addr* ′ =′ L.*array*.*base* ′[′L.*addr* ′]′); }

$L \rightarrow \textbf{id}\ [E]$ { L.*array* = *top.get*(**id**.*lexeme*);
L.*type* = L.*array*.*type*.*elem*;
L.*addr* = **new** *Temp*();
gen(L.*addr* ′ =′ E.*addr* ′ ∗′ L.*type*.*width*); }

$L \rightarrow L_1[E]$ { L.*array* = L_1.*array*;
L.*type* = L_1.*type*.*elem*;
t = **new** *Temp*();
L.*addr* = **new** *Temp*();
gen(t ′ =′ E.*addr* ′ ∗′ L.*type*.*width*);
gen(L.*addr* ′ =′ L_1.*addr* ′ +′ t); }

Exercise 6.4.3

Use the translation of Fig. 6.22 (also in previous slide) to translate the following assignments:

a) `x = a[i] + b[j]`

Assume:
int[5] a;
int[7] b;

- Syntax tree for types of a and b
- Parse tree for assignment
- Annotate

Translation Scheme for Backpatching

(Boolean Expressions)

$B \to B_1 || M B_2$ { *backpatch*(B_1.*falselist*, M.*instr*);
B.*truelist* = *merge*(B_1.*truelist*, B_2.*truelist*);
B.*falselist* = B_2.*falselist*; }

$B \to B_1 \&\& M B_2$ { *backpatch*(B_1.*truelist*, M.*instr*);
B.*truelist* = B_2.*truelist*;
B.*falselist* = *merge*(B_1.*falselist*, B_2.*falselist*); }

$B \to (\ B_1\ )$ { B.*truelist* = B_1.*truelist*;
B.*falselist* = B_1.*falselist*; }

$B \to E_1$ **rel** E_2 { B.*truelist* = *makelist*(*nextinstr*);
B.*falselist* = *makelist*(*nextinstr* + 1);
gen('`if` E_1.*addr* **rel**.*op* E_2.*addr* '`goto _`');
gen('`goto _`'); }

$M \to \epsilon$ { M.*instr* = *nextinstr*; }

Exercise

(Derived from problem 6.7.1(a) from second edition book)

- Construct the parse tree for the following boolean expression:

  ```
  a==b && (c==d || e==f)
  ```

- Using the translation scheme of Fig. 6.43, translate the above expression.
  Show the true and false lists for each subexpression.
  You may assume the address of the first instruction generated is 100.

The semantic actions needed from Fig. 6.43 are listed in the previous slide. They are also listed in the next slide, with a different numbering of the variables. This numbering may be more useful for the exercise.

Translation Scheme for Backpatching

(Boolean Expressions)

$B_3 \rightarrow B_4 || M_2 B_5$ { *backpatch*(B_4.*falselist*, M_2.*instr*);
B_3.*truelist* = *merge*(B_4.*truelist*, B_5.*truelist*);
B_3.*falselist* = B_5.*falselist*; }

$B \rightarrow B_1 \&\& M_1 B_2$ { *backpatch*(B_1.*truelist*, M_1.*instr*);
B.*truelist* = B_2.*truelist*;
B.*falselist* = *merge*(B_1.*falselist*, B_2.*falselist*); }

$B_2 \rightarrow (\ B_3\ )$ { B_2.*truelist* = B_3.*truelist*;
B_2.*falselist* = B_3.*falselist*; }

$B \rightarrow E_1$ **rel** E_2 { B.*truelist* = *makelist*(*nextinstr*);
B.*falselist* = *makelist*(*nextinstr* + 1);
gen('`if` E_1.*addr* **rel**.*op* E_2.*addr* '`goto _`');
gen('`goto _`'); }

$M \rightarrow \epsilon$ { M.*instr* = *nextinstr*; }

Translation Scheme for Backpatching

(Flow-of-Control Statements)

$S \rightarrow$ **if** (B) MS_1	{	*backpatch*$(B.truelist, M.instr)$;
		$S.nextlist =$ *merge*$(B.falselist, S_1.nextlist)$; }
$S \rightarrow \{L\}$	{	$S.nextlist = L.nextlist$; }
$S \rightarrow A$;	{	$S.nextlist =$ **null**; }
$M \rightarrow \epsilon$	{	$M.instr =$ *nextinstr*; }
$L \rightarrow L_1 MS$	{	*backpatch*$(L_1.nextlist, M.instr)$;
		$L.nextlist = S.nextlist$; }
$L \rightarrow S$	{	$L.nextlist = S.nextlist$; }

Exercise

(Extension of problem 6.7.1(a) from second edition book)

- Construct the parse tree for the following 'program':

```
{ if (a==b && (c==d || e==f)) x=1; y=x+1 }
```

- Using the translation scheme of Fig. 6.43 and Fig. 6.46, translate the above program.
  Show the next list for each statement or statement list.
  You may assume the address of the first instruction generated is 100.

The semantic actions needed from Fig. 6.46 are listed in the previous slide. They are also listed in the next slide, with a different numbering of the variables. This numbering may be more useful for the exercise.

Translation Scheme for Backpatching

(Flow-of-Control Statements)

$S_2 \rightarrow$ **if** (B) $M_4 S_3$	{	*backpatch*$(B.truelist, M_4.instr)$;
		$S_2.nextlist = merge(B.falselist, S_3.nextlist)$; }
$S \rightarrow \{L\}$	{	$S.nextlist = L.nextlist$; }
$S_3 \rightarrow A_1$;	{	$S.nextlist =$ **null**; }
$M \rightarrow \epsilon$	{	$M.instr = nextinstr$; }
$L \rightarrow L_1 M_3 S_1$	{	*backpatch*$(L_1.nextlist, M_3.instr)$;
		$L.nextlist = S_1.nextlist$; }
$L_1 \rightarrow S_2$	{	$L_1.nextlist = S_2.nextlist$; }