Intermediate Code Generation

Sudakshina Dutta

IIT Goa

19^{th} April, 2022

Backpatching

- One-pass code generation
- List of jumps are provided as synthesized attributes
- When a jump is generated, the target is temporarily left unspecified
- Each such jump is put on a list of jumps whose labels are to be filled in when proper label can be determined

One Pass code generation using Backpatching

- Synthesized attributes *truelist* and *falselist* of nonterminal B are used to manage labels for boolean expression
- $B.truelist$ will be the list of jump or conditional jump instructions that eventually gets label to which control goes if B is *true*
- $B.falselist$ will be the list of jump or conditional jump instructions that eventually gets label to which control goes if B is *false*

One Pass code generation using Backpatching

- To manipulate list of jumps, we use three functions
  1. *makelist*(i) creates a new list containing only i, an index into the array of instructions; *makelist* returns a pointer to the newly created list
  2. *merge*(p_1, p_2) concatenates the list pointed by p_1 and p_2 and returns a pointer to the concatenated list
  3. *backpatch*(p, i) inserts i as the target label for each of the instructions on the list pointed to by p
- By a marker nonterminal M, the index of the next instruction is generated

This non terminal M just goes to EPSILON.

Backpatching Evaluation of Boolean Expressions

If B1 goes false, then only we need to look for B2. Hence we use backpatch with M.instr which yields the next instruction i.e. B2

$1) B \rightarrow B_1 \| M\ B_2$	$\{backpatch(B_1.falselist, M.instr);$ $B.truelist = merge(B_1.truelist, B_2.truelist);$ $B.falselist = B_2.falselist;\}$
$2) B \rightarrow B_1 \&\& M\ B_2$	$\{backpatch(B_1.truelist, M.instr);$ $B.truelist = B_2.truelist;$ $B.falselist = merge(B_1.falselist, B_2.falselist);\}$
$3) B \rightarrow !B_1$	$\{B.truelist = B_1.falselist;$ $B.falselist = B_1.truelist;\}$
$4) B \rightarrow (B_1)$	$\{B.truelist = B_1.truelist;$ $B.falselist = B_1.falselist;\}$

Backpatching Evaluation of Boolean Expressions...continued

5) $B \rightarrow E_1\ rel\ E_2$	$\{B.truelist = makelist(nextinstr);$ $B.falselist = makelist(nextinstr + 1);$ $gen('if'\ E_1.addr\ rel.op\ E_2.addr\ 'goto\ \_');$ $gen('goto\ \_');\}$
6) $B \rightarrow true$	$\{B.truelist = makelist(nextinstr);$ $gen('goto\ \_');\}$
7) $B \rightarrow false$	$\{B.falselist = makelist(nextinstr);$ $gen('goto\ \_');\}$
8) $M \rightarrow \epsilon$	$M.instr = nextinstr;$

Target label of goto is unfilled.

Translation of Statements

- List of unfilled jumps are filled when targets are found
- The nonterminal B has two lists of jumps, $B.truelist$ and $B.falselist$
- Statements generated by nonterminals S and L have lists of unfilled jumps to the instructions following them. They are denoted by synthesized attribute *nextlist*

Translation of Statements

Here we are filling the unfilled targets of truelist of B with the M.instr which points to start address of S1.

1) $S \rightarrow if(B)\ M\ S_1$	$\{backpatch(B.truelist, M.instr);$ $S.nextlist = merge(B.falselist, S_1.nextlist);\}$
2) $S \rightarrow if(B)\ M_1\ S_1\ N\ else\ M_2\ S_2$ N, M1, M2 go to EPSILON	$\{backpatch(B.truelist, M_1.instr);$ $backpatch(B.falselist, M_2.instr);$ $temp = merge(S_1.nextlist, N.nextlist);\}$ $S.nextlist = merge(temp, S_2.nextlist);\}$
3) $S \rightarrow while\ M_1(B)\ M_2\ S_1$	$\{backpatch(S_1.nextlist, M_1.instr);$ $backpatch(B.truelist, M_2.instr);$ $S.nextlist = B.falselist;$ $gen('goto'\ M_1.instr)\}$

Translation of Statements..continued

$4) S \rightarrow \{L\}$	$\{S.nextlist = L.nextlist;\}$
$5) S \rightarrow A$ If A is an assignment statement. There wont be any jump terms.	$\{S.nextlist = null;\}$
$6) M \rightarrow \epsilon$	$\{M.instr = nextinstr;\}$
$7) N \rightarrow \epsilon$	$\{N.nextlist = makelist(nextinstr); gen('goto \_')\}$
$8) L \rightarrow L_1\ M\ S$	$\{backpatch(L_1.nextlist, M.instr);$ $L.nextlist = S.nextlist;\}$
$9) L \rightarrow S$	$\{L.nextlist = S.nextlist;\}$