BACKPATCHING

SANTHIYA.T
BP150503

- The easiest way to implement the syntax-directed definitions in to use passes.
  - Frist, construct a syntax tree for the input
  - Then walk the tree in depth-first order, computing the translations given in the definition.
  - The main problem with generating code for Boolean expression and flow-of-control statement is a singel pass is that during one single pass.

- Statement will be put on a list of goto statements whose labels will be filled in when the proper label can be determine. We call this subsequent filling in of labels backpatching.
- Use three functions:
  - Makelist(i) creates a new list containing only i, an index the array of quadruples;marklist returns a pointer to the list it has made.

- merge(p1,p2) concatenates the list pointed to by p1 and p2, and returns a pointer to the concatenated list.
- backpatch(p,i) inserts I as the target label for each of the statements on the list pointed to by p.

Boolean Expressions

- Construct a translation scheme suitable for producing quadruples for boolean expressions during bottom-up parsing.
- We insert a marker nonterminal M into the grammar to cause a semantic action to pick up, the index of the next quadruple to be generated.

E ->E1 or M E2

|E1 and M E2

|not E1

|(E1)

| id1 relop id2

| true

| false

M -> E

- Consider the production E->E1 and E2.
- If E1 is false, then W is also false, so the statements on E1.false become part of E.false.
- This target is obtained using the marker nonterminal M.
- Attribute M.quad records the number of the first statement of E2.code.
- With the production M->E we associate the semantic action
- { M.quad := nextquad }

- The variable nextquad holds the index of the next quadruple to follow.
- This value will be backpatched onto the E1.truelist when we have seen the remainder of the production E->E1 and M E2.
- The translation scheme is as follows.

E->E1 or M E2 { backpatch(E1.falselist,M.quad);
E.truelist := merge(E1.truelist,E2.truelist);
E.falselist := E2.falselist }

E->E1 and M E2 { backpath (E1.treulist,M.quad);
E.truelist := E2.truelist;
E.falselist := merge(E1.false, E2falselist)}

E -> not E1 { E.truelist := E1.falselist;
E.falselist := E1.truelist}

E -> (E1) {E.truelist := E1.truelist;
E.falselist := E1.falselist}

E -> true	{E.truelist := makelist(nextquad); emit(‘goto_’)}
E -> false	{E. false list := makelist(nextquad); emit(‘goto_’)}
M -> e	{ M.quad := nextquad }
E -> id1 relop id2	{ E.truelist := makelist(nextquad); E.falselist := makelist(nextquad) +1 ; emit(‘if’ id1.place relop.op id2.place ‘goto’_) emit(‘goto_’)}

Flow-of-Control Statements

- Backpatching can be used to translate flow-of-control statements in one Pass.
- We fix our attention on the generation of quadruples, and the natation regarding translation field names and list-handling procedures from that section carries over to this section as well.

Translation scheme grammar

S -> if E then S

| if E then S else S

| while L do S

| begin E end

| A

L –> L : S

| S

S → Statement,

L → Statement List

A → Assignment statement

E → Boolean Expression

- A given statement in execution, it physically in the quadruple array.
- General approach will be to fill in the jumps out of statements when their targets are found.
- Not only do boolean expressions.

Scheme to Implement the Translation

- A syntax-directed translation scheme to generate translation for the flow-of-control.
- The nonterminal E has two attributes E.truelist and E.falselist.
- L and S each also need a list of unfilled quadruples that must eventually be completed by backpatching.

- Pointed to by the attributes L.nextlist and S.listnext .
- S.nextlist is a pointer to a list of all condition and uncondition jumps and L.nextlist is defined similarly.
- The marker nonterminal M in the following production record the quadruple numbers
- S → while M1 do M2 S1

Labels and Gotos

- Programming landuage construct for changing the flow of control in a program is the label and goto.
- A compiler encounters a statement like goto L, in must check that there is exactly one statement with label L in the scope of this goto statement.

- The symbol table will have an entry giving the compiler-generated label for the first three-address instruction associated with the source statement labeled L.
- A label L is encountered for the first time in the source program either in a declaration or as the target of a forward goto, we enter L into the symbol tabel and generate a symbolic label for L.

THANK YOU