11 - Generation of LR(0) items

Thursday, April 18, 2024 11:51 AM

Ashima Fatima Seik Mujibur Rahman, 21BAI1830

AIM: Write a program to generate 3- address codes for the given expression.

Procedure:

1. Initialize 'itemid', which is a variable used to keep track of the number of LR(0) items processed, to -1.
2. Initialize 'lr0 items', a vector of 'LR0Item' objects representing the LR(0) items, with the first LR(0) item.
3. Initialize 'globalGoto', a map representing the global GOTO table, mapping productions to LR(0) items IDs, as an empty map.
4. Input the grammer in the format: LHS → RHS
5. Load the grammer into the 'AugmentedGrammer' map.
   a. Initialize the start production of the grammer and add it to 'lr0 items[0]'
   b. For each production, add it to the grammer map and push the augmented grammer (with the '@' symbol) to the first LR0Item.
6. Loop over each LR(0) item in 'lr0items'
   a. Call 'get_LR0_items' function to generate LR(0) items.
   b. In the function, ensure closure of the item.
   c. For each production in the LR(0) item:
      - check the lookahead symbol
      - if goto is not defined for the current i/p symbol, create a new item and add the production to it. update the gotos for the current item.

current item.
- if goto is defined, add the production to the corresponding next item.

7. Print the LR(0) items and their transitions (gotos).

Sample Input:

```
  Enter Grammar
-----------------
S
S->AA
S->aA
A->b
```

Sample Output:

```
Sets of LR(0) Items
-------------------
I0:
        '->@S                   goto(S)=I1
        S->@AA                  goto(A)=I2
        S->@aA                  goto(a)=I3
        A->@b                   goto(b)=I4
I1:
        '->S@
I2:
        S->A@A                  goto(A)=I5
        A->@b                   goto(b)=I4
I3:
        S->a@A                  goto(A)=I6
        A->@b                   goto(b)=I4
I4:
        A->b@
I5:
        S->AA@
I6:
        S->aA@
```

Code:

```
#include <cstdlib>
#include <iostream>
#include <string>
#include <string.h>
#include <vector>
#include <algorithm>
#include <map>

using namespace std;

typedef map<char, vector<string> > AugmentedGrammar;
typedef map<string, int> GotoMap; // <aps productions to LR(0) item ids

// structy for representing an augmented grammar production
```

```
struct AugmentedProduction
{
   char lhs;   // left hand side of production
   string rhs; // right hand side of production

   AugmentedProduction() {}
   AugmentedProduction(char _lhs, string _rhs) : lhs(_lhs), rhs(_rhs) {}
};

// Class for representing LR(0) items.
class LR0Item
{
private:
   // list of productions
   vector<AugmentedProduction*> productions;

public:
   // list of out-edges
   map<char, int> gotos;

   // constructor
   LR0Item() {}
   // destructor
   ~LR0Item() {}

   // add production
   void Push(AugmentedProduction *p) { productions.push_back(p); }
   // return the number of productions
   int Size() { return int(productions.size()); }

   // return whether or not this item contains the production prodStr
   bool Contains (string production) {
      for (auto it = productions.begin(); it != productions.end(); it++) {
         string existing = string(&(*it)->lhs, 1) + "->" + (*it)->rhs;
         //cout << " Comparing: " << thisStr << " , " << prodStr << endl;
         if (strcmp(production.c_str(), existing.c_str()) == 0) {
            return true;
         }
      }
      return false;
   }

   // overloaded index operator; access pointer to production.
   AugmentedProduction* operator[](const int index) {
      return productions[index];
   }
};

/* void add_closure
 * If 'next' is the current input symbol and next is nonterminal, then the set
 * of LR(0) items reachable from here on next includes all LR(0) items reachable
 * from here on FIRST(next). Add all grammar productions with a lhs of next */
void
add_closure(char lookahead, LR0Item& item, AugmentedGrammar& grammar)
```

```
{
   // only continue if lookahead is a non-terminal
   if (!isupper(lookahead)) return;

   string lhs = string(&lookahead, 1);
   // iterate over each grammar production beginning with p->rhs[next]
   // to see if that production has already been included in this item.
   for (int i = 0; i<grammar[lookahead].size(); i++) {
      string rhs = "@" + grammar[lookahead][i];
      // if the grammar production for the next input symbol does not yet
      // exist for this item, add it to the item's set of productions
      if (!item.Contains( lhs + "->" + rhs )) {
         item.Push(new AugmentedProduction(lookahead, rhs));
      }
   }
}

// produce the graph of LR(0) items from the given augmented grammar
void
get_LR0_items(vector<LR0Item>& lr0items, AugmentedGrammar& grammar, int& itemid, GotoMap& globalGoto)
{
   printf("I%d:\n", itemid);

   // ensure that the current item contains te full closure of it's productions
   for (int i = 0; i<lr0items[itemid].Size(); i++) {
      string rhs = lr0items[itemid][i]->rhs;
      char lookahead = rhs[rhs.find('@')+1];
      add_closure(lookahead, lr0items[itemid], grammar);
   }

   int nextpos;
   char lookahead, lhs;
   string rhs;
   AugmentedProduction *prod;

   // iterate over each production in this LR(0) item
   for (int i = 0; i<lr0items[itemid].Size(); i++) {
      // get the current production
      lhs = lr0items[itemid][i]->lhs;
      rhs = lr0items[itemid][i]->rhs;
      string production = string(&lhs,1) + "->" + rhs;

      // get lookahead if one exists
      lookahead = rhs[rhs.find('@')+1];
      if (lookahead == '\0') {
         printf("\t%-20s\n", &production[0]);
         continue;
      }

      // if there is no goto defined for the current input symbol from this
      // item, assign one.
      if (lr0items[itemid].gotos.find(lookahead) == lr0items[itemid].gotos.end()) {
         // that one instead of creating a new one
         // if there is a global goto defined for the entire production, use
```

```
            if (globalGoto.find(production) == globalGoto.end()) {
               lr0items.push_back(LR0Item()); // create new state (item)
               // new right-hand-side is identical with '@' moved one space to the right
               string newRhs = rhs;
               int atpos = newRhs.find('@');
               swap(newRhs[atpos], newRhs[atpos+1]);
               // add item and update gotos
               lr0items.back().Push(new AugmentedProduction(lhs, newRhs));
               lr0items[itemid].gotos[lookahead] = lr0items.size()-1;
               globalGoto[production] = lr0items.size()-1;
            } else {
               // use existing global item
               lr0items[itemid].gotos[lookahead] = globalGoto[production];
            }
            printf("\t%-20s goto(%c)=I%d\n", &production[0], lookahead, globalGoto[production]);
         } else {
            // there is a goto defined, add the current production to it
            // move @ one space to right for new rhs
            int at = rhs.find('@');
            swap(rhs[at], rhs[at+1]);
            // add production to next item if it doesn't already contain it
            int nextItem = lr0items[itemid].gotos[lookahead];
            if (!lr0items[nextItem].Contains(string(&lhs, 1) + "->" + rhs)) {
               lr0items[nextItem].Push(new AugmentedProduction(lhs, rhs));
            }
            swap(rhs[at], rhs[at+1]);
            printf("\t%-20s\n", &production[0]);
         }
      }
   }

   /**
    * void load_grammar
    * scan and load the grammar from stdin while setting first LR(0) item */
   void load_grammar(AugmentedGrammar& grammar, vector<LR0Item>& lr0items)
   {
      string production;
      string lhs, rhs;
      string delim = "->";

      getline(cin, lhs); // scan start production
      grammar['\''].push_back(lhs);
      lr0items[0].Push(new AugmentedProduction('\'', "@" + lhs));
      //printf("aug: \n");
      //printf("'->%s\n", lhs.c_str());
      while(1) {
         getline(cin, production);
         if (production.length() < 1) return;

         auto pos = production.find(delim);
         if(pos!=string::npos){
            lhs = production.substr(0,pos);
            rhs = production.substr(pos+delim.length(),std::string::npos);
         }
```

```
        grammar[lhs[0]].push_back(rhs);
        //printf("aug: \n");
        //printf("%s->%s\n", lhs.c_str(), rhs.c_str());
        lr0items[0].Push(new AugmentedProduction(lhs[0], "@" + rhs));
    }
}

// main
int main() {
    int itemid = -1; // counter for the number of LR(0) items
    AugmentedGrammar grammar;
    vector<LR0Item> lr0items = { LR0Item() }; // push start state
    GotoMap globalGoto;

    printf("  Enter Grammar\n");
    printf("-----------------\n");
    load_grammar(grammar, lr0items);
    printf("\n");

    printf("Sets of LR(0) Items\n");
    printf("-------------------\n");
    while (++itemid < int(lr0items.size())) {
        get_LR0_items(lr0items, grammar, itemid, globalGoto);
    }
    printf("\n");
    return 0;
}
```