DOKUMENTACIJA

Programski prevodioci - predmetni zadatak

Osnovni podaci

Broj indeksa	Ime i prezime	Tema
SW11/2019	Milica Popović	Generisanje i ispis AST

Korišćeni alati

Naziv	Verzija
GNU Make	4.1
gcc	7.5.0
flex	2.6.4
bison	3.0.4

Evidencija implementiranog dela

U okviru projekta su odrađena tri glavna zadatka:

- Generisanje AST
- Ispis AST
- Semantička analiza na osnovu AST

Generisanje AST je odrađeno u toku sintaksne analize. Po završetku sintaksne analize se radi semantička analiza na osnovu generisanog AST. U toku semantičke analize se obavljaju sledeće provere:

- Provera deklarisanosti promenljivih i funkcija
- Postojanje *main* funkcije
- Provera tipova (za operande operacije dodele, te aritmetičkih i relacionih operacija)
- Poklapanje broja i tipova parametara i argumenata funkcije
- Provera povratne vrednosti funkcije

Detalji implementacije

Generisanje AST

Čvor AST je implementiran kao struktura *AST_NODE,* kao što je prikazano na isečku 1. Atribut *name* predstavlja naziv čvora. Atribut *type* predstavlja tip čvora i njegove moguće

vrednosti su *NO_TYPE*, *INT* i *UINT*. Čvorovi kojima se zadaje konkretan tip (*INT* ili *UINT*) su čvorovi koji predstavljaju deklaraciju promenljive ili funkcije, literal ili parametar funkcije. Atribut *kind* predstavlja vrstu čvora i podržane vrste su:

- LIT – literal
- FUN – funkcija (deklaracija)
- FUN_CALL – poziv funkcije
- DECL – deklaracija promenljive
- VAR – promenljiva
- PAR – parametar funkcije
- AROP – aritmetički operator
- RELOP – relacioni operator
- ASSIGN – operator dodele
- IF – uslovni iskaz
- RETURN – povratna vrednost funkcije
- FUNCTIONS – sekvenca funkcija
- VARIABLES – sekvenca deklaracija promenljivih
- STATEMENTS – sekvenca iskaza

Takođe, svaki čvor sadrži kolekciju svojih potomaka, što je predstavljeno atribtom *children*. Maksimalan broj potomaka jednog čvora je 256. Atribut *func_ordinal* predstavlja redni broj funkcije kojoj pripada element na kog se čvor odnosi. On služi da bi se mogli razlikovati parametri i promenljive koji imaju isti naziv, a pripadaju različitim funkcijama.

```
typedef struct ast_node {
    char* name;
    unsigned type;
    unsigned kind;
    struct ast_node* children[256];
    int children_cnt;
    unsigned func_ordinal;
} AST_NODE;
```

Isečak 1 – AST čvor

Generisanje AST se obavlja u toku sintaksne analize, u okviru pravila produkcije. Kreirani čvor se postavlja za semantičku vrednost pojma u okviru kojeg je kreiran. Time je omogućeno uvezivanje čvorova u stablo, jer se tako čvorovi koji odgovaraju pojmovima sa desne strane produkcije mogu dodati u niz potomaka čvora koji odgovara levoj strani produkcije.

Međutim, pošto je u pitanju AST, a ne stablo parsiranja, pravila produkcije se ne mapiraju direktno na čvorove. Korenski čvor stabla je sekvenca funkcija (čvor tipa *FUNCTIONS*). On se generiše u okviru *program* pravila i čuva se u globalnoj promenljivoj, da bi mu se moglo pristupiti za svrhe semantičke analize i ispisa stabla. Unutrašnji čvorovi su sekvenca deklaracija promenljivih (VARIABLES), sekvenca iskaza (STATEMENTS), aritmetički operatori (AROP), relacioni operatori (RELOP), operator dodele (ASSIGN),

deklaracija funkcije (FUN), poziv funkcije (FUN_CALL), te uslovni iskaz i iskaz za povratnu vrednost funkcije (IF i RETURN). Listovi stabla su promenljive (DECL i VAR), literali (LIT) i parametri (PAR). Na isečku 2 je prikazano kreiranje AST čvora na primeru izraza (pojmovi *num_exp* i *exp*). Funkcija *build_node* sa isečka kreira novi čvor i postavlja mu zadati naziv, tip, vrstu i broj potomaka.

```
num_exp
  : exp
  | num_exp _AROP exp
    {
      AST_NODE* node = build_node(arops_str[$2], NO_TYPE, AROP, 2);
      node -> children[0] = $1;
      node -> children[1] = $3;
      $$ = node;
    }
  ;

exp
  : literal
  | _ID
    {
      $$ = build_node($1, NO_TYPE, VAR|PAR, 0);
    }
  | function_call
  | _LPAREN num_exp _RPAREN
    {
      $$ = $2;
    }
  ;
```

Isečak 2 – Generisanje AST čvora za izraze

Ispis AST

Stablo se ispisuje u konzolu tako što se radi *preorder* obilazak stabla po dubini (*Depth First Search*) i za svaki obiđeni čvor se ispiše njegov naziv. Pri tome je svaki čvor uvučen na odgovarajući način, tako da ispis bude pregledan i da se jasno mogu videti odnosi predak – potomak između čvorova. Na slici 1 je prikazan ispis AST za MiniC program koji je dat na isečku 3.

```
1   int fun1(int a) {
2       int b;
3       int c;
4       b = a + 2;
5       if (b > 3)
6           c = a - 5;
7       return c;
8   }
9
10  int main() {
11      int d;
12      d = fun1(5);
13      return 0;
14  }
```

Isečak 3 – Primer MiniC programa

```
functions
+--- fun1
|    +--- a
|    +--- variables
|    |    +--- b
|    |    +--- c
|    +--- statements
|         +--- =
|         |    +--- b
|         |    +--- +
|         |         +--- a
|         |         +--- 2
|         +--- if
|         |    +--- >
|         |    |    +--- b
|         |    |    +--- 3
|         |    +--- =
|         |         +--- c
|         |         +--- -
|         |              +--- a
|         |              +--- 5
|         +--- return
|              +--- b
+--- main
     +--- variables
     |    +--- d
     +--- statements
          +--- =
          |    +--- d
          |    +--- fun1
          |         +--- 5
          +--- return
               +--- 0
```

Slika 1 – Primer ispisa AST

Semantička analiza

Za obavljanje semantičke analize, potrebno je odraditi *postorder* obilazak stabla po dubini. Prilikom obilaska svakog čvora se proverava njegova vrsta, te se u skladu sa tim obavljaju odgovarajuće semantičke provere. Semantička analiza se obavlja u dva prolaza. U prvom prolazu se vrši upis deklarisanih funkcija, promenljivih i parametara u tabelu simbola. Zahvaljujući tome, funkcije mogu biti navedene u proizvoljnom redosledu.

Prilikom obilaska čvora koji predstavlja deklaraciju funkcije, proverava se da li je funkcija sa datim imenom već deklarisana. Ako nije, ona se dodaje u tabelu simbola. Takođe, ukoliko je naziv date funkcije *main,* pomoću globalnog indikatora se naznači da u programu postoji *main* funkcija. Ako funkcija ima parametar, i on se dodaje u tabelu simbola. Pri tome se čuva i redni broj funkcije, jer različite funkcije mogu imati parametar sa istim nazivom.

Analogno tome, prilikom obilaska čvora koji predstavlja deklaraciju promenljive, proverava se da li je promenljiva sa datim imenom već deklarisana u okviru iste funkcije. Ako nije, ona se dodaje u tabelu simbola, sa rednim brojem funkcije kojoj pripada kao atributom. U okviru tabele simbola je dodata funkcija za dobavljanje simbola, koja osim naziva i vrste,

proverava i redni broj funkcije kojoj simbol pripada. Ona se poziva prilikom dobavljanja promenljivih i parametara. Implementacije ove funkcije je prikazana na isečku 4.

U slučaju da čvor predstavlja aritmetičku, relacionu ili operaciju dodele, proverava se da li su levi i desni operand istog tipa. Pošto operand može biti ponovo operacija, potrebno je da se tipovi traže rekurzivno dok se ne dođe do čvora koji je literal, promenljiva, parametar ili poziv funkcije. U okviru ovog procesa se tip propagira kroz stablo, tako što se za tip roditeljskog čvora postavlja tip potomaka. Ako u nekom koraku dođe do nepoklapanja tipova potomaka, prijavljuje se semantička greška.

Prilikom nailaska na promenljivu, parametar ili poziv funkcije, proverava se njihovo postojanje u tabeli simbola. Za promenljive i parametre se proverava i to da li su deklarisani baš u onoj funkciji u kojoj se nalaze. U slučaju poziva funkcije se proveravaju broj i tip argumenata.

Kada se dođe do čvora koji predstavlja povratnu vrednost funkcije, proverava se da li je tip povratne vrednosti odgovarajući.

```
int lookup_symbol_in_func(char *name, unsigned kind, unsigned func_index) {
  int i;
  for(i = first_empty - 1; i > FUN_REG; i--) {
    if(strcmp(symbol_table[i].name, name) == 0
       && (symbol_table[i].kind & kind) && symbol_table[i].atr1 == func_index)
       return i;
  }
  return -1;
}
```

Isečak 4 – Funkcija za pronalazak simbola koja uzima u obzir redni broj funkcije

Testovi

Deo projekta su i testovi kojima se proverava ispravnost implementiranih funkcionalnosti. Svi testovi su leksički i sintaksno ispravni, jer je u ovom projektu fokus bio na semantičkoj analizi.

Testovi naziva *test-ok*.mc* su i semantički ispravni, tj. oni prolaze ukoliko program ne prijavi nijednu grešku. U okviru njih su testirane sve podržane konstrukcije jezika. Oni obuhvataju i slučaj kada se promenljive i parametri istih naziva javljaju u različitim funkcijama, kao i slučaj kada se iz funkcije poziva funkcija koja je deklarisana nakon nje.

Testovi naziva *test-semerr*.mc* sadrže semantičke greške, tj. oni prolaze ukoliko program prijavi semantičku grešku. Prvih 5 testova testira deklarisanje (deklaracija main funckije, redeklaracija promenljive, redeklaracija funkcije, upotreba nedeklarisane promenljive, poziv nedeklarisane funkcije). Naredna 4 testa testiraju tipove (tip povratne vrednosti, tipove leve i desne strane operacije dodele, aritmetičkih i relacionih operacija, te tip argumenta). Poslednja dva testa testiraju slučajeve u okviru kojih se funkciji prosledi pogrešan broj argumenata.

Ideje za nastavak

Za proširivanje projekta, mogli bi se implementirati generisanje AST i semantička analiza za prošireni MiniC jezik. Na primer, mogla bi se uvesti podrška za globalne promjenljive, različite vrste petlji, nove tipove podataka i funkcije sa više parametara. U tom slučaju, trebalo bi uvesti nove vrste AST čvorova i implementirati dodatne semantičke provere. Takođe, mogla bi se dodati funkcionalnost grafičkog iscrtavanja čvora mimo ispisa u konzoli.

Literatura

Slajdovi sa predavanja
Compilers: Principles, Techniques and Tools
https://www.twilio.com/blog/abstract-syntax-trees
http://www.cs.ecu.edu/karl/5220/spr16/SFLIMP/Assn3/ast.html