Programski prevodioci

05 Semantička analiza

Fakultet tehničkih nauka, Novi Sad
22-23/Z
Dunja Vrbaški

Nakon semantičke analize – znamo da je program ispravan.

Završen je prednji deo kompajlera (front-end) koji ne zavisi od ciljnog jezika.

Nakon analize (leksička, sintaksna, semantička) dolazi na red sinteza (generisanje koda).

Gruba, opšta, najjednostavnija podela na faze.

Bison → korisničke akcije definisane uz pravila gramatike.

Ovakve akcije se mogu iskoristiti za proveru semantičkih zahteva ali i dalje, za generisanje koda. Time se proces prevođenja završava u parseru.

Koristićemo ovaj pristup pri realizaciji kompajlera.

Jednostavno za implementaciju, ali ograničenih mogućnosti.

Tokenima se dodeljuju vrednosti/atributi na osnovu vrednosti pročitanih leksema.
Pojmovima se dodeljuju vrednosti/atributi na osnovu semantičkih pravila.

miniC - semantička pravila

Opseg vidljivosti i važenja (scope)

- globalni identifikatori
  - imena funkcija
  - vidljivost: od mesta definicije do kraja programskog teksta

- lokalni identifikatori
  - definisani u okviru funkcija
  - parametri i lokalne promenljive
  - vidljivost: od mesta definicije do kraja funkcije (bloka?)

- identifikatori mogu biti korišćeni samo nakon definicije (deklaracije)

Šta je: definicija; deklaracija; deklaracija je ujedno i definicija?

Jednoznačnost

- svi globalni identifikatori moraju biti međusobno različiti
- svi lokalni identifikatori iste funkcije moraju biti međusobno različiti
- lokalni identifikatori raznih funkcija mogu biti identični
- lokalni identifikatori i globalni identifikatori mogu biti identični

```
int main(int p) {
  int x;
  int y;
  int x;
}
int f(int p) {
  int x;
  int y;
  int p;
}
int f() {
}
```

- ako postoje identični globalni identifikatori i lokalni identifikatori neke funkcije, tada van te funkcije važe globalni, a unutar nje lokalni identifikatori

- rezervisane reči smeju da se koriste samo u skladu sa svojom ulogom i na globalnom i na lokalnom nivou

- standardni identifikator main je rezervisan samo na globalnom nivou

```
int x;
int f() {
  int x;
  x = 0;
}
int f1(int p) {
  int y;
  int else;
  y = 1;
}
unsigned f2() {
  int main;
  main = 9;
}
```

Tipovi

- leva i desna strana iskaza dodele moraju imati isti tip

- tipovi parametara i odgovarajućih argumenata funkcije moraju biti identični

```
int f(unsigned p) {
  int x;
  int y;
  x = y;
  x = p;
  p = p + p;
}
```

```
int inc(int p) {
  return p + 1;
}
int f() {
  int x;
  x = inc(3);
  x = inc();
  x = inc(3u);
}
```

- tip izraza return iskaza neke funkcije i tip povratne vrednosti funkcije moraju biti identični

- u istom relacionom izrazu identifikatori moraju biti istog tipa

- podrazumeva se da je tip literala int ako nije eksplicitno naznačeno da je tip unsigned

```
int main() {
  int x;
  return 5u;
  return 5;
}
int f() {
  int x;
  unsigned y;
  x = 3;
  y = 3u;
  if(y < 0)
    x = 0;
  else
    x = 1;
}
```

micko - realizacija semantičke analize

Već smo spominjali deo mehanizma koji će nam omogućiti realizaciju:

flex

Globalna promenljiva `yylval`

bison

`$$` - vrednost pojma sa leve strane
`$i` – vrednost i-tog pojma ili simbola sa desne strane

Novo:

tabela simbola

...

Tabela simbola

- struktura podataka koja sadrži informacije o identifikatorima (imenima)

- tipovi, informacije, scope
  - micko: vrsta simbola (fun, param, var, ...), njihovi tipovi (INT, UINT) i dodatne informacije

- “simbol” – terminologija: odnosi se na konkretan identifikator, nije token

- izgrađuje se i može se menjati u toku analize
  - kad se naiđe na ime dodaje se u TS
  - kad se dobije nova informacija može se menjati

- često se pretražuje

micko - implementacija TS

```
int f(unsigned p) {
    int a;
    unsigned b;
    b = 3u;
    b = b + p;
    a = 8;
}
```

name	*kind*	*type*	*atr1*	*atr2*
f	FUN	INT	1	UINT
p	PAR	UINT	1	-
a	VAR	INT	1	-
b	VAR	UINT	2	-
3	LIT	UINT	-	-
8	LIT	INT	-	-

Implementacija:

- pronaći odgovarajuću strukturu podataka (brzo dodavanje, pristup/pretraživanje)
- odlučiti koji atributi se pamte
- definisati funkcije: dodavanje simbola, menjanje simbola, traženje simbola, brisanje....

micko: symtab.h, symtab.c

name	*kind*	*type*	*atr1*	*atr2*
f	FUN	INT	1	UINT
p	PAR	UINT	1	-
a	VAR	INT	1	-
b	VAR	UINT	2	-
3	LIT	UINT	-	-
8	LIT	INT	-	-

Elementi tabele simbola

primer na ovom kursu; micko

`name` – string simbola (leksema)
`kind` - vrsta simbola (FUN, VAR, PAR, LIT, REG)
`type` - tip simbola (INT, UINT)
`atr1` - atribut 1
- za lokalnu promenljivu - redni broj promenljive
- za parametar - redni broj parametra
- za funkciju – broj parametara
- za ostale simbole - nije definisano

`atr2` – atribut 2
- za funkcije - tip parametra
- za ostale simbole - nije definisano

name	*kind*	*type*	*atr1*	*atr2*
f	FUN	INT	1	UINT
p	PAR	UINT	1	-
a	VAR	INT	1	-
b	VAR	UINT	2	-
3	LIT	UINT	-	-
8	LIT	INT	-	-

Registri i konstante - dodati radi uniformnosti.
Inicijalizacija tabele – prih *n* mesta rezervisano za registre. Prisutni su sve vreme u tabeli.
O registrima kasnije (uticaj ciljne arhitekture)

```
int main() {
  int a;
}
```

name	*kind*	*type*	*atr1*	*atr2*
main	FUN	INT	0	-
a	VAR	INT	1	-

```
int main(int p) {
  int a;
  unsigned x;
}
```

name	*kind*	*type*	*atr1*	*atr2*
main	FUN	INT	1	INT
p	PAR	INT	1	-
a	VAR	INT	1	-
x	VAR	UINT	2	-

- Ceo naš prevodilac se realizuje u jednom prolazu – tako se i naša TS popunjava i koristi
- Ponaša se kao stek
- Globalni identifikatori (imena funkcija) se dodaju u TS
- Lokalni identifikatori se dodaju odmah nakon odgovarajućeg globalnog
- Kada se završi prolaz kroz funkciju lokalni identifikatori se brišu

- Doseg imena

- *informacije o identifikatorima potrebne i kasnije (dijagnostika)? - ne sme se raditi fizičko brisanje*

```
unsigned f(unsigned x) {
  return x + x;
}
int main() {
  unsigned a;
  a = f(5u);
  a = f(1u, a);   //error
}
```

name	*kind*	*type*	*atr1*	*atr2*
f	FUN	UINT	1	UINT
main	FUN	INT	-	-

Tabela simbola posle analize obe funkcije.

defs.h

```
//tipovi podataka
enum types { NO_TYPE, INT, UINT };

//vrste simbola (moze ih biti maksimalno 32)
enum kinds { NO_KIND = 0x1, REG = 0x2, LIT = 0x4,
             FUN = 0x8, VAR = 0x10, PAR = 0x20 };
```

symtab.h

```c
#ifndef SYMTAB_H
#define SYMTAB_H

// Element tabele simbola
typedef struct sym_entry {
   char *   name;          // ime simbola
   unsigned kind;          // vrsta simbola
   unsigned type;          // tip vrednosti simbola
   unsigned atr1;          // dodatni attribut simbola
   unsigned atr2;          // dodatni attribut simbola
} SYMBOL_ENTRY;

// Vraca indeks prvog sledeceg praznog elementa.
int get_next_empty_element(void);

// Vraca indeks poslednjeg zauzetog elementa.
int get_last_element(void);

// Ubacuje novi simbol (jedan red u tabeli)
// i vraca indeks ubacenog elementa u tabeli simbola
// ili -1 u slucaju da nema slobodnog elementa u tabeli.
int insert_symbol(char *name, unsigned kind, unsigned type,
                  unsigned atr1, unsigned atr2);

// Ubacuje konstantu u tabelu simbola (ako vec ne postoji).
int insert_literal(char *str, unsigned type);

// Vraca indeks pronadjenog simbola ili vraca -1.
int lookup_symbol(char *name, unsigned kind);
```

```c
// set i get metode za polja tabele simbola
void     set_name(int index, char *name);
char*    get_name(int index);
void     set_kind(int index, unsigned kind);
unsigned get_kind(int index);
void     set_type(int index, unsigned type);
unsigned get_type(int index);
void     set_atr1(int index, unsigned atr1);
unsigned get_atr1(int index);
void     set_atr2(int index, unsigned atr2);
unsigned get_atr2(int index);

// Brise elemente tabele od zadatog indeksa
void clear_symbols(unsigned begin_index);

// Brise sve elemente tabele simbola.
void clear_symtab(void);

// Ispisuje sve elemente tabele simbola.
void print_symtab(void);
unsigned logarithm2(unsigned value);

// Inicijalizacija tabele simbola.
void init_symtab(void);
```

semantic.y

```
function
  : _TYPE _ID
      {
        fun_idx = lookup_symbol($2, FUN);
        if(fun_idx == NO_INDEX)
          fun_idx = insert_symbol($2, FUN, $1, NO_ATR, NO_ATR);
        else
          err("redefinition of function '%s'", $2);
      }
    _LPAREN parameter _RPAREN body
      {
        clear_symbols(fun_idx + 1);
        var_num = 0;
      }
  ;
```

Drugačije realizacije tabele simbola

- stek stekova kao niz (linearna lista) – jednostavna za implementaciju, lako dodavanje, ali neefikasno pretraživanje
- bolja struktura – heš tabele za svaki doseg (stek heš tabela)

- TS za main/global scope + TS za svaki scope → stablo čiji su čvorovi TS (zapisi odgovaraju aktivacionim slogovima kod generisanja koda)

- Češće dinamička struktura (ne niz fiksne dužine, kao kod nas)

- Elementi su uniformni (npr struktura), ali mogu imati dinamičke vrednosti
    - (ime ili atributi nisu fiksne veličine već su pokazivači)

- povezivanje sa međureprezentacijom (AST)

- OP - nasleđivanje, višestruko nasleđivanje - problemi

Pitanja i zadaci

Razmisliti o mogućim strukturama za tabelu simbola.

Šta je statička, a šta dinamička provera tipova? Kakvu proveru mi imamo?
type checking

Kakve veze imaju apstraktna sintaksna stabla i programski jezik Lisp?