Jednoduchý interpretační překladač

Náznak ukázky syntaxe a sémantiky pro projekt

Šárka Vavrečková

Ústav informatiky, FPF SU Opava

sarka.vavreckova@fpf.slu.cz

Poslední aktualizace: 8. ledna 2008

1 Syntaktické prvky

- matematické výrazy, proměnné (celočíselné), celá čísla,
- `begin`, `end`, `var`, `obdelnik`, `jestli`, `pak`, `cti`, `pis`,
- možné příkazy:
  - `var prom1, prom2, ...;`
    deklarace proměnných – příkaz `var` je vždy na začátku programu, ukončení středníkem je povinné, proměnné nemusí být žádné,
  - `obdelnik(x,y,s,v)`
    vykreslení obdélníka na souřadnice $[x, y]$ o šířce s a výšce v, parametry jsou výrazy,
  - `cti(prom), pis(vyraz),`
  - `begin ... end`
    složený příkaz, blok,
  - `jestli podminka pak prikaz`
    rozhodování, podmínka je `vyraz < vyraz` nebo `vyraz = vyraz`,
  - `prom := vyraz`
    přiřazovací příkaz, dvojsymbol `:=` se načte lexikálním analyzátorem jako symbol `r`,
- příkazy jsou *ukončeny* středníkem.

2 Popis jazyka

Sémantické prvky:

- proměnné musí být deklarovány na začátku programu, při deklaraci je proměnná zařazena do tabulky symbolů,
- protože je umožněno používat příkaz typu IF, příkazy mají dědičný atribut $prov$ (proveď), který je při nesplnění podmínky nastaven na $false$ a tato hodnota je také děděna (například u příkazu bloku),
- příkaz je proveden (interpretován) pouze tehdy, když má atribut $proved$ nastaven na $true$, jinak je pouze syntakticky rekurzívně rozvinut (pro kontrolu syntaktických chyb).

2.1 Syntaxe

$S \to DbTe.$	
$D \to vI$	deklarace
$I \to iJ$	
$I \to ;$	
$J \to ,I$	
$T \to PR$	posloupnost příkazů
$T \to \varepsilon$	
$R \to ;T$	
$P \to o(V,V,V,V)$	obdelnik
$P \to c(i)$	cti
$P \to p(V)$	pis
$P \to bTe$	blok příkazů
$P \to jMtP$	jestli ...
$P \to irV$	i := V

$V \to AB$	výraz
$B \to +AB$	
$B \to -AB$	
$B \to \varepsilon$	
$A \to CE$	
$E \to *CE$	
$E \to /CE$	
$E \to \varepsilon$	
$C \to n$	
$C \to i$	
$C \to (V)$	
$M \to VN$	podmínka
$N \to = V$	
$N \to < V$	

2.2 Kontrola, zda je LL(1)

Množiny Follow:

$FL(S) = \{\$\}$
$FL(D) = \{b\}$
$FL(I) = \{b\}$
$FL(J) = \{b\}$
$FL(T) = \{e\}$
$FL(R) = \{e\}$
$FL(P) = \{;\}$

$FL(V) = \{,,), ;, =, <, t\}$
$FL(A) = \{+, -, ,,), ;, =, <, t\}$
$FL(B) = \{,,), ;, =, <, t\}$
$FL(C) = \{*, /, +, -, ,,), ;, =, <, t\}$
$FL(E) = \{+, -, ,,), ;, =, <, t\}$
$FL(M) = \{t\}$
$FL(N) = \{t\}$

Dá se snadno ověřit, že se jedná o LL(1) gramatiku.

3 Sémantika

$S \rightarrow Db\ \{T.prov = true\}\ Te.$

$D \rightarrow vI$

$I \rightarrow i\ \{\text{Pridej}(i.nazev)\}\ J$
$I \rightarrow;$

$J \rightarrow, I$

$T \rightarrow \{P.prov = T.prov\}\ P\ \{R.prov = T.prov\}\ R$
$T \rightarrow \varepsilon$

$R \rightarrow;\ \{T.prov = R.prov\}\ T$

$P \rightarrow \{V_0.prov = V_1.prov = V_2.prov = V_3.prov = P.prov\}\ o(V_0, V_1, V_2, V_3)$
$\{\text{if } P.prov \text{ then KresliObd}(V_0.val, V_1.val, V_2.val, V_3.val)\}$
$P \rightarrow c(i\ \{\text{if } P.prov \text{ then Zmen}(i.Nazev, \text{NactiZeVstupu})\}\ )$
$P \rightarrow p(\ \{V.prov = P.prov\}\ V\ \{\text{if } P.prov \text{ then Vypis}(V.val)\}\ )$
$P \rightarrow b\ \{T.prov = P.prov\}\ Te$
$P_0 \rightarrow j\ \{M.prov = P_0.prov\}\ Mt$
$\{\text{if } P_0.prov \text{ and } M.bool \text{ then } P_1.prov = true \text{ else } P_1.prov = false\}\ P_1$
$P \rightarrow ir\ \{V.prov = P.prov\}\ V\ \{\text{if } P.prov \text{ then Zmen}(i.nazev, V.val)\}$

$V \rightarrow \{A.prov = B.prov = V.prov\}$
$A\ \{\text{if } V.prov \text{ then } B.m = A.val\}\ B\ \{\text{if } V.prov \text{ then } V.val = B.val\}$

$B_0 \rightarrow \{A.prov = B.prov = V.prov\}$
$+\ A\ \{\text{if } B_0.prov \text{ then } B_1.m = B_0.m + A.val\}\ B_1\ \{\text{if } B_0.prov \text{ then } B_0.val = B_1.val\}$
$B_0 \rightarrow \{A.prov = B.prov = V.prov\}$
$-\ A\ \{\text{if } B_0.prov \text{ then } B_1.m = B_0.m - A.val\}\ B_1\ \{\text{if } B_0.prov \text{ then } B_0.val = B_1.val\}$
$B \rightarrow \varepsilon\ \{\text{if } B.prov \text{ then } B.val = B.m\}$

$A \rightarrow \{C.prov = E.prov = A.prov\}$
$C\ \{\text{if } A.prov \text{ then } E.m = C.val\}\ E\ \{\text{if } A.prov \text{ then } A.val = E.val\}$

$E_0 \rightarrow \{C.prov = E_1.prov = E_0.prov\}$
$*\ C\ \{\text{if } E_0.prov \text{ then } E_1.m = E_0.m * C.val\}\ E\ \{\text{if } E_0.prov \text{ then } E_0.val = E_1.val\}$
$E_0 \rightarrow \{C.prov = E_1.prov = E_0.prov\}$
$/\ C\ \{\text{if } E_0.prov \text{ then } E_1.m = E_0.m/C.val\}\ E_1\ \{\text{if } E_0.prov \text{ then } E_0.val = E_1.val\}$
$E \rightarrow \varepsilon\ \{\text{if } E.prov \text{ then } E.val = E.m\}$

$C \rightarrow n\ \{\text{if } C.prov \text{ then } C.val = n.lex\}$
$C \rightarrow i\ \{\text{if } C.prov \text{ then } C.val = \text{ DejHodn}(i.lex)\}$
$C \rightarrow (\ \{V.prov = C.prov\}\ V)\ \{\text{if } C.prov \text{ then } C.val = V.val\}$

$M \rightarrow \{V.prov = N.prov = M.prov\}$
$V\ \{\text{if } M.prov \text{ then } N.m = V.val\}\ N\ \{\text{if } M.prov \text{ then } M.bool = N.bool\}$

$N \rightarrow =\ \{V.prov = N.prov\}\ V\ \{\text{if } N.prov \text{ then } N.bool = (N.m == V.val)\}$
$N \rightarrow <\ \{V.prov = N.prov\}\ V\ \{\text{if } N.prov \text{ then } N.bool = (N.m < V.val)\}$

4 Implementace

Použijeme rekurzivní sestup, vytvoříme tyto funkce (procedury):

- `S, D, I, J, T(prov: boolean), R(prov: boolean), P(prov: boolean),`
- `V(prov: boolean; var val: integer),`
  `A(prov: boolean; var val: integer),`
- `B(prov: boolean; m: integer; var val: integer),`
  `E(prov: boolean; m: integer; var val: integer),`
- `C(prov: boolean; var val: integer),`
- `M(prov: boolean; var bool: integer),`
- `N(prov: boolean; m: integer; var bool: integer),`
- `Pridej, Zmen, DejHodn` – pro práci s tabulkou,
- `KresliObd` – pro vykreslení obdélníka,
- `NactiZeVstupu, Vypis` – pro práci se vstupem a výstupem.

Atribut *prov* a jeho testování lze vynechat, pokud nepoužijeme rozhodovací příkaz.

4.1 Tabulka Symbolů pro uložení proměnných

Možnosti implementace jsou různé, jedna z nejjednodušších je třeba spojový seznam.

```
type
  TNazev = string[15];    // omezeni delky nazvu promenne
  PPolozkaTab = ^TPolozkaTab;
  TPolozkaTab = record
    nazev: TNazev;
    hodnota: integer;
    dalsi: PPolozkaTab;
  end;

var
  Tabulka: PPolozkaTab;  // ukazatel na prvni prvek tabulky

procedure InicializujTabulku(var tab: PPolozkaTab);
procedure ZnicTabulku(var tab: PPolozkaTab);

procedure Najdi(var ukaz_tab: PPolozkaTab; nazev: TNazev);
// v ukazateli ukaz_tab vrati ukazatel na tu polozku tabulky, ve ktere
// je promenna s danym nazvem, kdyz v seznamu neni, vrati nil

procedure Pridej(var tab: PPolozkaTab; nazev: TNazev; hodnota: integer);
// jestlize je tabulka prazdna, vytvori prvni polozku s temito udaji,
```

```
// jestlize ne, najde dane misto v tabulce a prida zaznam
// kdyby uz polozka existovala, hlasi semantickou chybu

procedure Zmen(var tab: PPolozkaTab; nazev: TNazev; hodnota: integer);
// overi, zda polozka s timto nazvem existuje, kdyz ne, hlasi
// semantickou chybu, kdyz ano, provede zmenu hodnoty

function DejHodn(tab: PPolozkaTab; nazev: TNazev): integer;
// kdyz polozka s timto nazvem neexistuje, hlasi semantickou chybu
```

První parametr těchto funkcí můžeme vynechat, pokud budeme mít jen jedinou (globální) tabulku. Jestliže chceme použít blokovou strukturu a odlišovat různé úrovně lokálních proměnných, navíc přidáme dynamický zásobník, do kterého budeme řadit tabulky bloků, a vyhledávací proceduru upravíme tak, aby začala vyhledávat na vrcholu zásobníku a pokračovala v něm dále.

Jinou podobnou implementací by byl binární strom, ve kterém by se navíc zjednodušilo a zrychlilo vyhledávání. Rozdíl je jenom v uspořádání položek – místo v seznamu by byly v binárním stromu, do datového typu `TPolozkaTab` dáme místo odkazu na následující položku dva odkazy (levý a pravý potomek).

4.2 Implementace procedury `P`

Nejdřív uvedeme základní tvar procedury, dále „vnitřek“ pro jednotlivá pravidla:

```
procedure P(prov: boolean);
var v1, v2, v3, v4: integer; b: boolean; s: NazevProm;
begin
  case vstupni_sym.typ of
  ...               // jednotlive hodnoty typu symbolu
  else chyba(.....);
  end;              // case
end;
```

$P \rightarrow \{V_0.prov = V_1.prov = V_2.prov = V_3.prov = P.prov\}\ o(V_0, V_1, V_2, V_3)$
$\quad \{\text{if } P.prov \text{ then KresliObd}(V_0.val, V_1.val, V_2.val, V_3.val)\}$

```
SYM_OBD: begin
  expect(SYM_OBD);
  expect(SYM_LZAV);
  V(prov,v1);  expect(SYM_CARKA);
  V(prov,v2);  expect(SYM_CARKA);
  V(prov,v3);  expect(SYM_CARKA);
  V(prov,v4);  expect(SYM_RZAV);
  if prov then KresliObd(v1,v2,v3,v4);
end;
```

$P \rightarrow c(i$ {if $P.prov$ then Zmen$(i.Nazev,$ NactiZeVstupu)})

```
SYM_CTI: begin
  expect(SYM_CTI); expect(SYM_LZAV);
  if vstupni_sym.typ = SYM_ID then s := sym.atribstr;
  expect(SYM_ID);
  if prov then begin
    NactiZeVstupu(v1);
    Zmen(s,v1);
  end;
  expect(SYM_RZAV);
end;
```

$P \rightarrow p($ $\{V.prov = P.prov\}$ V {if $P.prov$ then Vypis$(V.val)$})

```
SYM_PIS: begin
  expect(SYM_PIS); expect(SYM_LZAV);
  V(prov,v1);
  if prov then Vypis(v1);
  expect(SYM_RZAV);
end;
```

$P \rightarrow b$ $\{T.prov = P.prov\}$ Te

```
SYM_BEGIN: begin
  expect(SYM_BEGIN);
  T(prov);
  expect(SYM_END);
end;
```

$P_0 \rightarrow j$ $\{M.prov = P_0.prov\}$ Mt
{if $P_1.prov$ and $M.bool$ then $P_1.prov = true$ else $P_1.prov = false$} P_1

```
SYM_JESTLI: begin
  expect(SYM_JESTLI);
  M(prov,b);
  expect(SYM_PAK);
  P(prov and b);
end;
```

$P \rightarrow ir$ $\{V.prov = P.prov\}$ V {if $P.prov$ then Zmen$(i.nazev, V.val)$}

```
SYM_ID: begin
  s := sym.atribstr;
  expect(SYM_ID);
  expect(SYM_PRIRAD);
  V(prov,v1);
  if prov then Zmen(s,v1);
end;
```

5 Příkazy cyklů

U příkazů cyklů je třeba vyřešit jeden důležitý problém – zpravidla je nutné se vracet v kódu. To lze řešit dvěma způsoby v závislosti na tom, jak je naprogramován lexikální analyzátor:

1. Jestliže je lexikální analyzátor v samostatném průchodu a tedy syntaktický analyzátor má k dispozici výstup lexikální analýzy vcelku (třeba jako dynamický seznam), provádí návrat samotný syntaktický analyzátor a blok kódu uvnitř cyklu včetně podmínky je vyhodnocován lexikální analýzou pouze jednou.
2. Pokud je lexikální a syntaktický analyzátor ve společném průchodu a tedy syntaktický analyzátor nemá přístup k celému svému zdroji najednou, musí provést navracení lexikální analyzátor, a to přímo ve zdrojovém souboru. Můžeme postupovat například takto:

   - kdykoliv narazí na klíčové slovo určující cyklus, třeba `while`, uloží svou pozici – vytvoří „zarážku“,
   - když syntaktický analyzátor narazí na konec cyklu, vyhodnotí, zda se má tento cyklus znovu provést,
     - pokud ano, požádá lexikální analyzátor o návrat k nejbližší zarážce (tj. od nejvnitřnějšího cyklu),
     - pokud ne, požádá lexikální analyzátor o zrušení nejbližší zarážky,
   - je třeba ošetřit i takové cykly, které se ve skutečnosti neprovedou ani jednou (to se může stát u cyklů typu `while`).

   Nevýhodou je nutnost provádět lexikální analýzu obsahu cyklu a podmínky opakovaně pro všechny průchody. To se sice dá řešit vytvořením „dočasného mezikódu“ vnitřku cyklu, ale toto řešení není zrovna transparentní, už proto, že cyklus může být třeba v rozsahu celého programu a také cykly mohou být navzájem vnořené, čímž se dostáváme k prvnímu bodu tohoto seznamu a ztrácíme výhodu zařazení obou fází překladu do společného průchodu.

První případ se řeší zachycením ukazatele do seznamu výstupu lexikální analýzy, druhý vytvořením funkcí pro komunikaci s lexikálním analyzátorem s tím, že je nutné vyřešit samotné navracení ve vstupu (soubor by mohl být načten jako stream, ve kterém se lze snadněji pohybovat).

Ukážeme si náznak řešení pro příkaz while v případě použití prvního postupu (lexikální analyzátor je v samostatném průchodu), implementace druhého postupu by byla podobná (práce se zarážkami).

Pravidlo včetně sémantiky bude vypadat takto:

$P_0 \rightarrow w$ $\{M.prov = P_0.prov,$ UlozUmisteni, repeat, NactiUmisteni$\}$
Md $\{P_1.prov = P_0.prov$ and $M.bool\}$
P_1 {until not($P_1.prov$ and $M.bool$}

Programový kód (úsek v proceduře `P`) vypadá takto:

```
SYM_WHILE: begin
  expect(SYM_WHILE);
  pom_sym := zpracovavany_symbol;   // zachytime momentalni pozici v kodu
  repeat
    zpracovavany_symbol := pom_sym; // vratime se na zachycenou pozici
    M(prov,b);
    expect(SYM_DO);
    P(prov and b);
  until not (prov and b);
end;
```