Implementace a použití generátoru překladačů v prostředí Smalltalku

Petr Šaloun

1. Úvod

V prostředí počítačů se často setkáváme s potřebou převést strukturované textové údaje do buďto jiného textového tvaru, nebo do jiné vnitřní representace. Zvláštním případem takového překladu je překlad vstupního (zdrojového) textu do proveditelného strojového programu. Formalismus takového překladu je dobře zpracován. Je možno vytvořit generátory překladačů, jimž jako vstup slouží vhodné specifikační výrazy.

Cílem mého příspěvku je informovat o implementaci generátoru překladačů v prostředí Smalltalku/V [Dig92]. Jako příklad možného použití generátoru jsme použili úlohu zpracování vstupních pravidel neuronového expertního systému NEUREX.

Použití generátoru při tvorbě programového vybavení je velmi výhodné. Formální popis vstupní informace nám umožňuje i syntaktickou kontrolu její správnosti při zpracování. Navíc nám dává možnost odhalení případných chyb (konfliktů) již ve formálním popisu samém. Případná modifikace takového popisu prochází stejnou kontrolou správnosti. Generátor ovšem nekontroluje sémantické akce.

Vytvořený generátor je rozdělen do několika funkčních celků s různým stupněm propracovanosti grafického uživatelského rozhraní (dále jen GUI) i automatizovanosti činnosti. Podrobnost popisu je určena rozsahem vymezeným tomuto příspěvku.

2. Princip činnosti LALR(1) S-atributovaného překladu

Při překladu vykonává překladač syntaktické i sémantické akce. Vstupem pro překlad je posloupnost symbolů (a případně jejich hodnot) ve vstupním jazyce. Výsledkem překladu je posloupnost v jazyce výstupním. Vstupní jazyk je popsán gramatikou, která je použita pro tvorbu překladového stromu.

Bezkontextová gramatika je určena (konečnými) množinami terminálů, neterminálů, pravidel a počátečním symbolem). Pravidla mají na levé straně neterminál, pravá strana je (případně prázdná) posloupnost terminálů a neterminálů. Gramatika určuje typ překladu. Pro podrobnější informace můžeme použít například [MČM87].

Při zpracování vstupních symbolů jsou načteny a dále zpracovávány jejich (syntetizované) atributy. Syntetizované atributy můžeme vyhodnocovat při analýze zdola nahoru. Vhodným typem gramatiky, popisujícím jak dostatečně rozsáhlou třídu jazyků, tak vyžadujícím přijatelně rozsáhlé množství informací (paměti) je třída LALR(1) gramatik [ČBH93].

Syntetizované atributy symbolů pravé strany jsou umístěny na zásobníku. Hodnotu syntetizovaného atributu neterminálu levé strany pravidla získáme při redukci. Způsob výpočtu definujeme sémantickými pravidly.

Klíčovou vlastností syntaxí řízeného atributovaného překladu je skutečnost, že činností vykonávané překladačem je jen několik. Pokud pro jednoduchost neuvážíme výpočet hodnot atributů a výstupní objekty, jsou to: přesun, redukce, přijetí a případně chyba. Tyto činnosti vykonává překladač na základě **tabulky akcí**, definované pro každý **stav** a **vstupní symbol** (terminál). Po provedení činnosti je z **tabulky přechodů** určen následující stav (pro každý stav a neterminál). Překladač vykonává akce až do úspěšného konce, nebo chybového stavu.

3. Rozdělení generátoru

V předchozí kapitole jsme se přibližně dozvěděli:
Překladač je abstraktní stroj zpracovávající vstupní věty. Je řízený rozkladovou tabulkou, vykonává sémantické akce a produkuje výstup.

Naše úkoly jsou:
- vytvořit rozkladovou tabulku (má části popisující akce a přechody)
- definovat vstupní symboly a vyhodnocení jejich atributů
- definovat sémantické akce
- vytvořit obecnou část překladače

V následujících kapitolách postupně popíšeme jednotlivá řešení. Příkladem, na němž budeme jednotlivé etapy tvorby dokumentovat, je tato posloupnost vzorů pro NEUREX [Vo94]:

(37,42) (100,100) (50,50) (50,50) (0) > (0,0) (100,100) (0,0);
(37,37) (100,100) (100,100) (20,20) (0,0) > (100,100) (0,0) (0,0);
(100,100) (0,0) (100,100) (100,100) (100,100) > (0,0) (0,0) (100,100);

Tabulka 1. Vstupní údaje

Vzory popisují vstupní údaje a z nich vyvozená fakta. Vstupní skupina dvojic určujících minimum a maximum (možno zkrátit na jednu hodnotu) je po '>' následována závěry ukončenými ';'. Výsledkem našeho snažení bude překladač zpracovávající takový vstup a produkující skupinu objektů určujících předpoklady a tvrzení.

4. Tvorba rozkladové tabulky

Tvorba rozkladové tabulky je zdlouhavá činnost, již raději přenecháme programu. Musíme být ovšem schopni definovat gramatiku. Pro zadání gramatiky použijeme formalismus. Vstup je navíc rozdělen do sekcí:

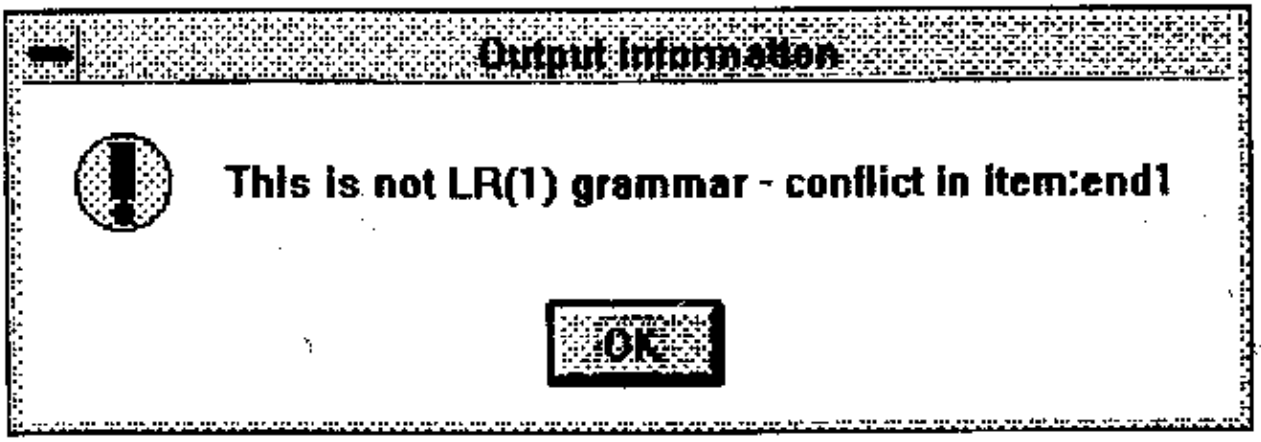
```
%GrammarStart%
Mar 27, 1995  22:19:47
%NullString
e
%Terminals
comma end imply lpar number rpar semi
%Nonterminals
Accept Accept' Item Items Pair Pairs
%Rules
Accept -> Items end
Accept' -> Accept
Item -> Pairs imply Pairs
Items -> e
Items -> Items Item semi
Items -> Items semi
Pair -> number
Pair -> number comma number
Pairs -> e
Pairs -> Pairs lpar Pair rpar
```

```
%StartSymbol
Accept'
%ExtendedSymbol
Accept'
%%
```

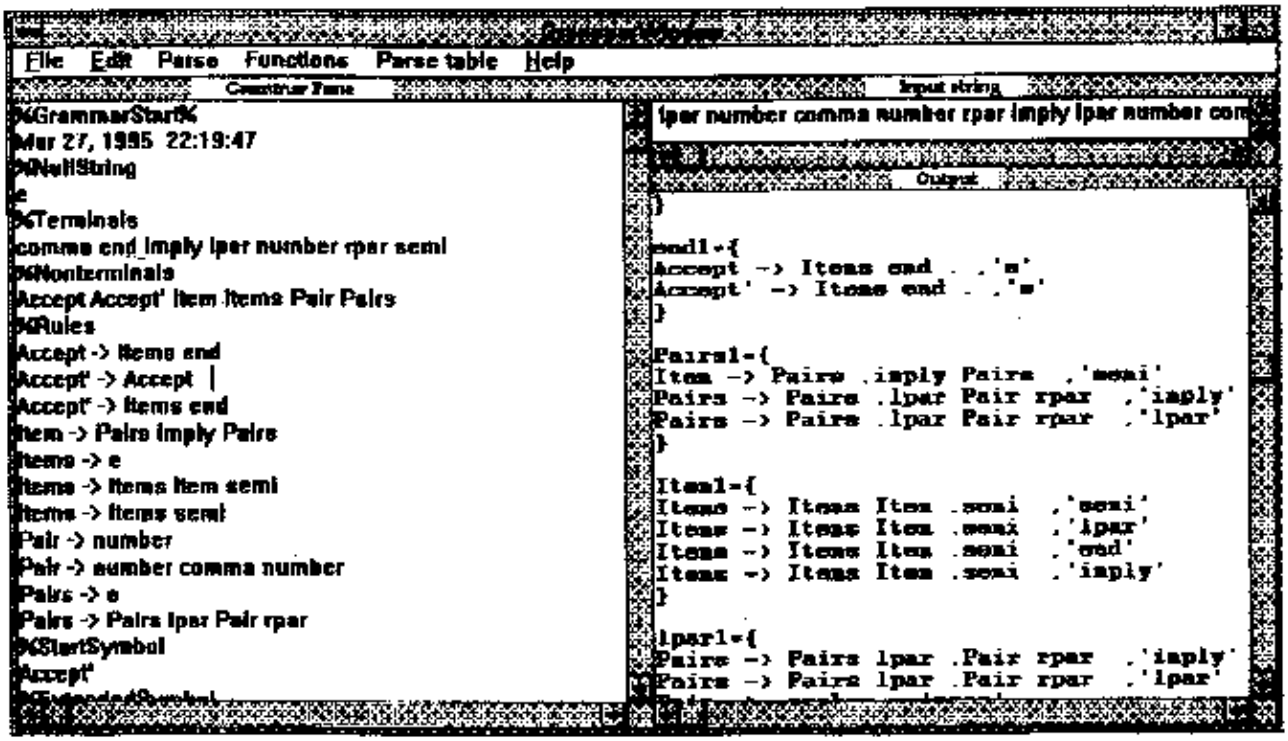
Tabulka 2. Určení gramatiky vstupního jazyka

V sekcích postupně definujeme prázdný řetězec, terminální a neterminální symboly, pravidla, počáteční a rozšířený symbol gramatiky. Pravidla pro určení gramatiky umožňují začlenit komentáře, zkrátit určení pravidel pro shodnou levou stranu atd.

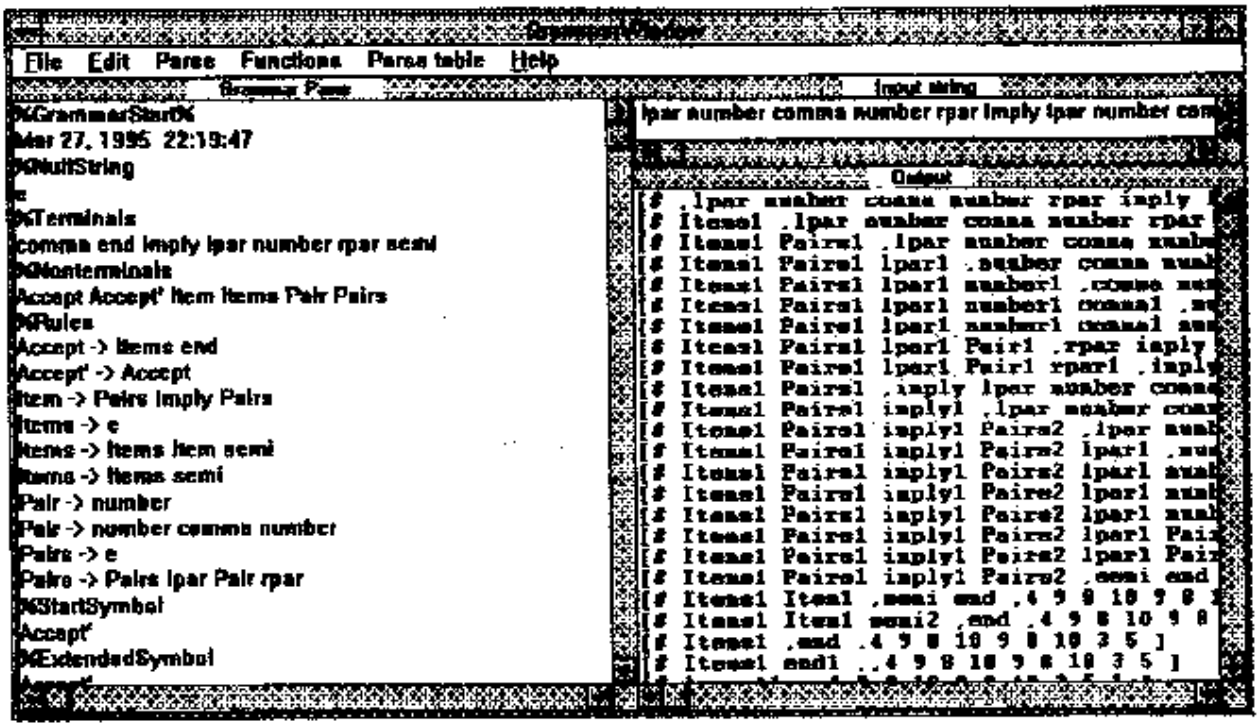
V komfortním prostředí můžeme testovat specifikovanou gramatiku na konflikty (přesun/redukce a redukce/redukce). Program nejen oznámí příslušný konflikt, ale určí i jeho výskyt. Díky možnému zobrazení množin LALR(1) položek se můžeme pokusit o jeho odstranění.



Obrázek 1. Chybové hlášení s konfliktní položkou



Obrázek 2. LR(1) výpis konfliktní položky



Obrázek 3. Testování vstupu

Výstupem je rozkladová tabulka, která je obvykle značně řídká. Proto je výhodné uchovávat ji v jiném tvaru, než je původní matice. Optimalizovaný tvar ukazuje tabulka 3.

```
state: 0
   action: #((all r3))
      goto: #((items 1));
state: 1
   action: #((e a) (semi s2) (imply r6) (left r6) (all error))
      goto: #((item 3) (pairs 4));
state: 2
   action: #((all r2))
      goto: #();
state: 3
   action: #((semi s5) (all error))
      goto: #();
state: 4
```

Tabulka 3. Optimalizovaný tvar rozkladové tabulky

5. Definice tokenů

Při zpracování vstupní informace vyvstává problém nestejně dlouhých řetězců pro různé vstupní symboly. Navíc musíme s některými terminály předávat i jejich hodnotu. Řešení je zavedení tokenů, nesoucích tyto informace.

```
Object subclass: #Token
  instanceVariableNames:
    ' symbol attribute '
  classVariableNames: ''
```

poolDictionaries: " !

Tabulka 4. Třída Token

Samotná třída token ovšem vstup nezpracuje. Tím se zabývá část překladače nazývaná **lexikální analyzátor**.

Lexikální analyzátor je při čtení znaků vstupního textu řízen konečným automatem. Ve Smalltalku je výhodné jeho řídící část definovat jako bázovou třídu. Její potomek pak bude rozlišovat jednotlivé lexikální jednotky podle konkrétní aplikace.

Část lexikálního analyzátoru, rozpoznávajícího symboly našeho příkladu ukazuje tabulka 5.

```
aSymbol = $,
   ifTrue: [
      state := #comma.
      output := output, (String with: aSymbol).
      token := Token symbol: #comma attribute: output.
      ^nil
   ].
aSymbol = $>
   ifTrue: [
      state :- #imply.
      output := output, (String with: aSymbol).
      token := Token symbol: #imply attribute: output.
      ^nil
   ].
state := #error.! !
```

Tabulka 5. Některé lexikální symboly

6. Činnost překladače

Činnost překladače byla obecně popsána ve druhé kapitole. Při její implementaci budeme postupovat podobně, jako v případě lexikální analýzy. Vytvoříme stroj realizující činnost LALR(1) syntaktického analyzátoru, viz [ČBH93], nebo [Ho90]. Při přesunech pracujeme s tokeny (tedy symboly i s jejich atributy), které jsou při redukcích použity pro výpočet atributu levé strany.

Potomek obecného překladače implementuje zprávy zodpovědné na zpracování sémantických akcí. Způsob volání je ve Smalltalku elegantní - můžeme jej provést například takto:

```
self perform: ((rules at: i) semantics) ...
```

Díky zapouzdření stačí určit zprávu, jež má být provedena. Při výpočtu samotném je na tokeny odvoláváme podle jejich umístění v pravidle gramatiky.

7. Připojení sémantických akcí

Sémantickou akci zavoláme snadno a elegantně. Místo, v němž ji určíme, je definice pravidel gramatiky. Připojení vymezeného identifikátoru zprávy za poslední symbol pravé strany je v našem případě postačující. Naše implementace ovšem neprovádí kontrolu shody

s definovanými zprávami (viz níže) překladové třídy. Zde jsme prostě ponechali nižší komfort obsluhy (i nižší bezpečnost).

Do překladové třídy musíme přidat definice zpráv sémantických akcí. To je činnost, jíž spojujeme dosud nezávislou tvorbu rozkladové tabulky s lexikální a syntaktickou analýzou.

8. Závěr

Popsaný generátor je ve své první verzi. To se projevuje zejména v nižší propracovanosti připojení sémantických akcí.

Při srovnání s "klasickými" generátory rodiny yacc+lex jsou jeho implementované jazykové možnosti chudší. Usoudili jsme, že případné změny a rozšíření provedeme až na základě zkušeností s používáním generátoru.

Přesto věříme, že generátor ušetří mnoho rutinní práce a poslouží jako základ pro výzkumné či komerční aplikace.

Literatura

[MČM87] Molnár, L., Češka, M., Melichar, B.: Gramatiky a jazyky. Alfa - SNTL, Bratislava - Praha, 1987

[ČBH93] Češka, M., Beneš, M., Hruška, T.: Překladače. Skriptum FEL VUT Brno, 1993

[Dig92] Smalltalk/V for Windows, Tutorial and Programming Handbook. Encyclopedia of Classes, Digitalk 1992

[Ho90] Holub, A. I.: Compiler Design in C. Prentice-Hall, 1990

[Vo94] Vondrák, I.: Neuronový expertní systém NEUREX. Dokumentace a program. Ostrava, 1991-94

Autor příspěvku:
RNDr. Petr Šaloun
odborný asistent
katedra informatiky K456
FEI VŠB-TU Ostrava
tř. 17. listopadu
708 33 Ostrava
tel: (069) 699 kl. 4213, 3263
e-mail: Petr.Saloun@vsb.cz