CONTEXT DRIVEN LEXICAL ANALYZER

Peter Hatina
Bachelor Degree Programme (3), FIT BUT
E-mail: xhatin01@stud.fit.vutbr.cz

Supervised by: Martin Čermák
E-mail: icermak@fit.vutbr.cz

ABSTRACT

This paper is devoted to the principles of a lexical analysis and to a means of context resolution of a lexeme type, depending on its source code position. The lexical analysis and possible solution for context lexeme recognition, based on a multiple automata system, is described. Lexical analyzer functionality is extended to accept lexemes of two other languages.

1 ÚVOD

Prvá a zároveň jediná časť prekladača jazyka, lexikálny analyzátor, prichádza do styku so zdrojovým textom programu [1]. Jeho prvoradou úlohou je rozpoznať logicky oddelené postupnosti znakov, tvoriace lexémy. Lexém, ako reťazec znakov, reprezentuje istú konštrukciu v jazyku, pomocou ktorého sú vytvorené riadiace časti programu, príkazy, premenné, atď. Lexikálny analyzátor predstavuje prostredník medzi zdrojovým textom a analyzátorom, kontrolujúcim syntaktickú správnosť zapísaných konštrukcií v danom jazyku, pričom medzi sebou komunikujú pomocou tzv. tokenov (vnútorne reprezentované lexémy). Ďalšou z činností, ktorú lexikálny analyzátor vykonáva, je práca s tabuľkou symbolov (vkladanie nových symbolov). Okrem vlastnej činnosti, dochádza k preskakovaniu bielych miest a ignorovaniu komentárov v zdrojovom súbore. Analyzátory môžu sledovať aktuálny riadok programu, v ktorom sa nachádzajú a využiť túto informáciu pri hlásení chýb. Ak vstupný jazyk využíva preprocesor, je možné expandovať makrá týmto analyzátorom.

2 ROZBOR

Lexikálna analýza a samotný lexikálny analyzátor je postavený na základe konečného automatu (ďalej len KA, viď Definícia 2.1). Princíp analýzy pozostáva z čítania znaku zo vstupného súboru, pričom po prečítaní znaku sa vykoná prechod v konečnom automate podľa príslušného pravidla.

Definícia 2.1. Konečný automat je pätica $M = (Q, \Sigma, R, s, F)$, pričom platí:

- Q je konečná množina stavov
- Σ je vstupná abeceda

- R je konečná množina pravidiel v tvare $pa \to q$, kde $p,q \in Q$, a $\in \Sigma \cup \{\varepsilon\}$
- $s \in Q$ predstavuje počiatočný stav
- $F \subseteq Q$ je konečná množina koncových stavov

2.1 LEXIKÁLNY ANALYZÁTOR

Spôsob tvorby KA, ktorý prijíma vstupný jazyk, pozostáva z tvorby deterministických konečných automatov (ďalej DKA), prijímajúcich časti jazyka — jednotlivé lexémy, pričom platí, že DKA je KA, ktorý má pre aktuálnu konfiguráciu definovaný maximálne jeden možný prechod. Spojením jednotlivých DKA vznikne KA, ktorý umožňuje rozlíšenie typu lexému podľa konečného stavu automatu, prípadne vyhlásenie lexikálnej chyby. Lexikálny analyzátor postavený na tomto type KA neumožňuje rozpoznať typ lexému podľa miesta výskytu. Túto situáciu musí riešiť syntaktický analyzátor. Jedným z možných riešení tohto problému je zostaviť množinu automatov, kde každý z nich má odlišnú úlohu.

2.2 KONTEXTOM RIADENÝ LEXIKÁLNY ANALYZÁTOR

V zdrojovom texte programu sa vyskytujú lexémy, ktoré nesú viacero významov z hľadiska pozície v samotnom súbore. Pre odlíšnie významu lexémov je možné použiť *automatový systém* [2], tvoriaci množinu medzi sebou komunikujúcich KA (viď Riešenie). Jeden z automatov vykonáva základné rozlíšenie lexémov, zvyšok zavádza do analyzátora kontextovú závislosť a konečné priradenie typu lexému. Tento systém je schopný rozpoznať kontext, čiže vrátiť syntaktickému analyzátoru presný typ tokenu.

Príklad: Nejednoznačné lexémy

```
int *ptr = &var;                    var = 7 & 3;
10 * *ptr;                          int &ref = var;
```

Riešenie: Majme automatový systém $N = \{M_1, M_2\}$, pričom úlohou automatu M_1 je čítanie vstupného súboru a základné rozpoznanie lexémov. Konečný automat M_2 je zavedený za účelom rozpoznať kontext. Ak KA M_1 vykoná sériu prechodov končiacu v jednom z koncových stavov, automat M_2 vykonáva prechod podľa rozpoznaného lexému automatom M_1. Po dosiahnutí koncového stavu automatu M_2 je možné presne určiť typ prečítaného lexému, prípadne série lexémov, ktoré možno zlúčiť.

3 ZHODNOTENIE

Navrhnutým spôsobom je možné rozlíšiť tri možné významy lexému „`*`“ — operátor násobenia, dereferenčný operátor a definíciu ukazateľa, rovnaká situácia platí aj pre lexém „`&`“.

Pre ukážku je možné zaviesť jazyk, ktorý je podmnožinou jazyka *C++* a je rozšírený o možnosť vkladania blokov kódu v jazyku symbolických inštrukcii *assembler* a funkcií zapísaných v jazyku tvoriaceho podmnožinu jazyka *Lisp*. Zo známych vlastností rozširujúcich jazykov je zrejmé, že v prípade *assembleru* bude lexém „`{`“ niesť iný význam — v spojení s kľúčovým slovom „`asm`“ sa jedná o začiatok bloku jazyka. Podobná situácia sa objavuje v prípade jazyka *Lisp*, kde lexém „`(`“ a „`)`“ (v tomto poradí) znamenajú pre syntaktický analyzátor začiatok a koniec zoznamu.

Ako ďalšie praktické rozšírenie lexikálneho analyzátora je možnosť prijímať sekvenciu lexémov od automatu M_1, predstavujúcu istú syntaktickú konštrukciu v danom jazyku a vrátiť ju ako jeden token, ktorý bude obsahovať vhodné parametre z lexémov tvoriacich sekvenciu.

Jedná sa v prvom rade o typy konštrukcií ako:

- definícia ukazateľa
  `int *pointer`
- definícia a inicializácia referencie
  `int &ref = variable`
- definícia poľa bez rozmerov
  `int array[] = {1, 2, 3}`
- definícia triedy
  `class SomeClass: public Base`
- začiatok kódu v jazyku *assembler*
  `asm {`
- funkcia v podmnožine jazyka *Lisp*
  `(defun(foo p1 p2 ... pN)(`

4 ZÁVER

Spomenutý jazyk, ktorý prijíma automatový systém má radu výhod, ktoré spočívajú v správnom priradení lexému jeho skutočnému významu. Systém bol rozšírený o schopnosť prijímať sekvencie lexémov a vyhodnocovať ich ako jeden token. Táto vlastnosť umožňuje zjednodušiť gramatiku syntaktického analyzátora, keďže konštruckia v jazyku je reprezentovaná jedným tokenom — isté syntaktické kontroly sa vykonajú už v časti lexikálnej analýzy. Keďže lexikálny analyzátor pracuje s tabuľkou symbolov, je možné presne vyhradiť priestor v pamäti pri definícii polí bez udaných rozmerov (táto povinnosť odpadá synktaktickému analyzátoru).

Jednou z nevýhod je spomalenie celej analýzy, z dôvodu zavedenia množiny automatov (viď Tabulka 1). K definitívnemu vráteniu presného typu tokenu je potrebné vykonať aspoň 2 prechody (v našom prípade, každý z automatov po jednom).

klasický lex		automatový systém		vstupný súbor	
čas	počet tokenov	čas	počet tokenov	riadky	slová
0.035s	31 561	0.056s	20 881	7 541	20 760
0.011s	10 263	0.022s	10 175	1 783	3 854
0.007s	3 784	0.008s	3 719	822	2 112
0.003s	1 011	0.005s	723	700	236
0.002s	187	0.005s	138	57	132

Tabulka 1: Časový priebeh analyzátorov

Poděkování: Tato práce vznikla částečně za podpory grantu VUT FIT, FIT-S-10-2 a specifického výzkumu MSM0021630528.

REFERENCE

[1] Aho, A. V.; Lam, M. S.; Sethi, R.; aj.: *Compilers – principles, techniques and tools*. Boston: Pearson, 2007, ISBN 0-321-49169-6, 1009 s.

[2] Čermák, M.: Systems of Formal Models and Their Application, In: Proceedings of the 14th Conference Student EEICT 2008, Brno, CZ, FEKT VUT, 2008, p. 164-166, ISBN 978-80-214-3615-2