SYNTAX ANALYSIS BASED ON SEVERAL METHODS

Luděk Dolíhal
Master Degree Programme (2), FIT BUT
E-mail: xdolih00@stud.fit.vutbr.cz

Supervised by: Alexander Meduna
E-mail: meduna@fit.vutbr.cz

ABSTRACT

The main goal of this paper is to analyze the creation of the composite compiler. Composite compiler is in this case a system, which consists of more cooperating parts. In this particular case the syntax analyser consists of two parts. The work is focused on the description of such parts of the parser, on its cooperation and comunication. I will try to scatch the teoretical backgroun of this solution. This is to be done by gramatical systems.

1 ÚVOD

Počet překladačů neustále narůstá stejně jako narůstá počet programovacích jazyků. Programovací jazyky se dnes uplatňují ve stále více oblastech, a proto je potřeba také neustále rozšiřovat možnosti a způsoby, jak překladače vytvářet. V dnešní době už není v některých případech možné a v jiných případech vhodné rozložit daný jazyk pouze pomocí jednoho způsobu a proto vznikly syntaktické analyzátory, které jsou složeny z více částí. Takovýmto překladačem se zabývá i tato práce.

2 GRAMATICKÉ SYSTÉMY

V běžné teorii programovacích jazyků je výpočet prováděn jednou výpočetní jednotkou. To znamená, že jazyk je generován jednou gramatikou a přijímán jedním automatem. Nicméně v dnešní době hraje stále větší roli distribuované zpracování. Takovéto systémy si však nevystačí se základní teorií programovacích jazyků. Jsou potřeba prostředky pro zápis paralelismu, distribuování výpočtu, konkurence a jiných. Proto byly zavedeny gramatické systémy.

Gramatické systémy ve své podstatě představují představují množinu gramatik pracujících pohromadě. Pokud beží výpočet na více stanicích, jsou stanice synchronizovány pomocí protokolu, tak, aby produkovaly jeden jazyk. Je jasné, že klíčovou roli zde bude hrát právě souhra jednotlivých prvků množiny. Rozlišují se dva základní typy gramatických systémů.

- ♢ sekvenční
- ♢ paralelní

CD gramatické systémy [2] jsou systémy sekvenční. V tomto typu systémů mají všechny části společnou větnou formu. Aktivní je vždy pouze jedna gramatika a to ta, která právě pracuje na společné větné formě. V takovémto systému jsou nutné nějaké prvky, které udávají, kdy se právě aktivní gramatika stane neaktivní a předá řízení a větnou formu gramatice jiné. Které, o to se postará protokol. Příkladem takovýchto stop prvků mohou být například závorky, nebo if podmínka. Právě tento typ gramatického systému je v mém překladači implementován.

Naproti tomu PC gramatické systémy [2] jsou paralelní. Zde má každá gramatika svou vlastní větnou formu a mohou tedy všechny pracovat ve stejný okamžik, každá na své vlastní větné formě. V tomto typu gramatických systémů je nutný globální časovač. Zde je velmi podstatná existence takzvaných dotazovacích (query) symbolů. Jejich funkčnost je následující. Pokud část i vygeneruje symbol Q_j, pak aktuální větná forma komponenty j je zaslána části i, kde nahradí všechny výskyty Q_j.Více se lze o těchto systémech dozvědět zde [2].

3 SYNTAKTICKÝ ANALYZÁTOR

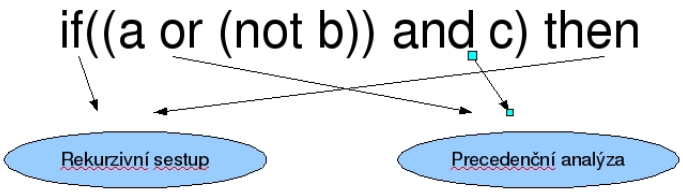
Samozřejmě, že překladač nesestává pouze ze syntaktického analyzátoru, ale konstrukce ostatních částí není ani zdaleka tak zajímavá. Proto nebudou v této práci příliš rozebírány.

Syntaktický analyzátor je typicky nejsložitější částí překladače. Jelikož se v dnešní době nejčastěji používá syntaxí řízený překlad, je kladen velký důraz na to, aby byl parser co nejrychlejší.

Metody překladu, lze rozdělit na dvě skupiny. Na skupinu algoritmů pracujících shora dolů a na skupinu algoritmů pracujících obráceně, tedy zdola nahoru. Obě dvě skupiny jsou popsány v literatuře [1].

4 SPOLUPRÁCE OBOU METOD

Hlavní metodou rozkladu zdrojového kódu je rekurzviní sestup. Pomocí něj jsou rozkládány veškeré příkazy, které se ve zdrojovém souboru vyskytnou. Narazí-li tato metoda na libovolný výraz je řízení předáno precedenčnímu analyzátoru, který jej rozloží a ověří tak zda je výraz validní. Poté je řízení navráceno zpět rekurzivnímu sestupu. Celou situaci pěkně popisuje následující obrázek.



Obrázek 1: Příklad rozkladu podmínky

Rekurzivní sestup tedy začne se zpracovávíním zdrojového kódu a překládá jej dokud nenarazí na výraz.

Překládáním se v tomto případě rozumí lexikální, syntaktická a sémantická kontrola. Lexikální analyzátor je napsán ručně i kdyz bylo možné použít například lex. V programu je uplatněn takzvaný syntaxí řízený překlad, kdy si syntaktický analyzátor volá analyzátor lexikální v případě, že potřebuje další token a řídí i činnost sémantického analyzátoru.

Rekurzivní sestup tedy zpracovává zdrojový kód, nechává si posílat tokeny pokud to zrovna potřebuje a pokud je kód validní, tak si volá sémantický analyzátor, aby kód ověřil i po sémantické stránce. Syntaktický analyzátor vlastně předá k sémantické analýze gramaticky správnou instrukci a sémantický analyzátor ověří, zda jsou kompatibilní typy, zda jsou proměnné deklarované a podobně. Pokud instrukce projde i sémantickým analyzátorem je vygenerován jeden prvek seznamu, který je tvořen instrukcemi v mezikódu. Takováto instrukce nese informace o operaci, která je v jejím rámci prováděna a také nese informace o jednotlivých operandech.

Pokud narazí rekurzivní sestup na výraz je řízení předáno precedenční analýze. Precedenční analýza potřebuje ke své práci nezpracovanou část zdrojového kódu, to znamená že je nutné předat deskriptor souboru spoléčně s pozicí na níž se zrovna nacházíme, dále potřebuje ukazatel na seznam instrukcí v tříadresném kódu, aby mohla také generovat instrukce. Při precedenční analýze vlastně redukujeme výrazy, které máme na zásobníku, dokud nám ná zásobníku něco zůstává. Každá takováto redukce znamená syntaktickou a sémantickou kontrolu zredukovaného výrazu a vygenerování jednoho prvku do seznamu intrukcí v tříadresném kódu. Po zredukování celého obsahu zásobníku je řízení předáno zpět rekurzivnímu sestupu společně s posledním operandem, který nám na zásobníku zůstal. Ten je později využit rekurzivním sestupem.

Takto probíhá překlad za ideálních podmínek. Pokud se ovšem v programu vyskytne chyba, tak je potřeba nejakým způsobem na daný problém reagovat. Nejjednoduším způsobem je ukončit překlad. Další možností je upozornit na vzniklou chybu a pokračovat v překladu. V rámci mého projektu je řešeno zotavení z chyb. Pokud program narazí na chybu, tak automaticky přeskočí vstup, který zbývá dočíst do konce řádku a s překladem pokračuje až na řádku následujícím. Samozřejmostí je vypsání chybové hlášky společně s řádkem na němž jsme chybu nalezli.

Ještě bych rád podotknul pár slov k sémantické analýze funkcí. Ta je totiž na rozdíl od zbytku kódu relativně komplikovaná. U funkcí totiž musíme uchovávat dosti informací. Například návratový typ, počet a typ parametrů a podobně. Všechny tyto údaje jsou v mém případě uchovávány ve speciální struktuře. Prvky této struktury jsou vytvářeny když narazíme na definici funkce a jsou později využívány pokud je funkce volána.

5 ZÁVĚR

V této práci byl diskutován syntaktický analyzátor sestávající z více částí. Od kombinace několika metod překladu si slibuji především rychlost překladače. V neposlední řadě nelze tomuto řešení upřít určitou eleganci.

REFERENCE

[1] Tremblay, J.P. and Sorenson, P.G.: The Theory and Practise of Compiler Writing, New York, McGraw-Hill , ISBN 0-070-65161-2

[2] Rozenberg, G. and Salomaa, A.: Handbook of formal languages Vol. 2 Linear Modeling: Background and application, Berlin, Springer, ISBN 3-540-60648-3