THE DESIGN OF A COMBINED PARSER

Petr MACHATA, Bachelor Degree Programme (3)
Dept. of Information Systems, FIT, BUT
E-mail: xmacha31@stud.fit.vutbr.cz

Supervised by: Dr. Alexander Meduna

ABSTRACT

The goal of this paper is to outline various thoughts concerning the design of a parser, based on several parsing systems. The author picks two common parsing algorithms – recursive descend parser and precedence parser – and discusses design issues related to combining them into single parser. Attention is paid to a couple of problematic areas – the communication between parsing algorithms, syntax error handling and context sensitive restrictions.

1 KONCEPT

Algoritmy pro syntaktickou analýzu se principielně dělí na dvě skupiny – na analyzátory pracující shora dolů, a ty, které pracují zdola nahoru. Cílem této práce je prozkoumat možnosti kombinace těchto přístupů. Autor použije rozklad metodou shora dolů (jmenovitě rekurzivní sestup) pro zachycení struktury řetězce “z ptačí perspektivy”, a drobné detaily (např. aplikaci precedenčních pravidel ve výrazech) nechá na precedenčním analyzátoru, který pracuje metodou zdola nahoru.

K dalším možnostem využití tohoto konceptu patří případy, kdy jsou malé části rozkládaného řetězce nesmírně náročné na překlad. Pro získání maximální výkonnosti pak lze velkou část řetězce analyzovat rychle, a na drobné podřetězce pak pustit náročný (a tudíž pomalý) algoritmus. Nebo naopak, pokud se to vyplatí, rychlost rozkladu lze zvyšovat tím, že některé části necháme rozložit jednoduchým a rychlým algoritmem. (Což se ostatně běžně používá – pomyslete na lexikální vs. syntaktickou analýzu. . .)

2 NÁVRH

V následujících kapitolách bude řeč o problémech spojených se zvolenou metodou analýzy. Bude rozebrána komunikace mezi algoritmy, dále bude zmíněno několik detailů týkajících se úpravy gramatiky pro analyzátor, a nakonec zpracování chyb v analyzovaném řetězci.

2.1 KOMUNIKACE MEZI ALGORITMY

Algoritmy podílející se na syntaktické analýze spolu obousměrně komunikují. Jedná se o typický příklad komunikace klient-server, kdy nadřízený analyzátor (klient) úkoluje podřízený (server) příkazy “rozlož řetězec X”. Příkaz k předání řízení je pak prvním směrem komunikace, odpověď druhým. V odpovědi server vrací klientovi syntaktický strom určeného řetězce, nebo odpověď, že rozkladu nelze dosáhnout. To lze v prostředí programovacího jazyka implementovat prostým voláním funkce.

Do analyzační funkce se jako jeden z argumentů předává kontext, obsahující data společná celé analýze (např. tabulky symbolů, syntaktický strom). V prostředí OOP lze kontext implementovat pomocí třídy, v níž jsou analyzátory zapouzdřeny. Díky tomu má každý z algoritmů přístup k veškerým datům, která ke své činnosti potřebuje.

V případě rekurzivního sestupu je implementace tohoto konceptu přímočará, a serverem může být libovolný jiný rozkladový algoritmus. Funkce pro analýzu neterminálu může místo své obvyklé rozkladové činnosti volat server a vyhodnotit jeho výsledky.

Pokud precedenční analyzátor není výhradním serverem, tj. pokud se z něj stane klient a chce žádat o rozklad někoho třetího, vyžaduje to obohacení klasického algoritmu precedenční analýzy. V gramatice jsou vyznačeny dvojice klíčových slov, ohraničujících řetězec předávaný podřízené analýze. Analyzátor o nich ví, a pokud na počáteční slovo narazí, přepne se do zvláštního režimu, v němž hledá konec ohraničeného řetězce. Nalezený řetězec potom pošle na analýzu serveru. Algoritmus hledání řetězce ohraničeného klíčovými slovy je shodný pro rekurzivní sestup i precedenční analýzu. (pozn.: Navržený demonstrační jazyk počítá s tím, že precedenční analyzátor není výhradním serverem.)

2.2 GRAMATIKA

Pro účely kombinované analýzy lze gramatiku rozdělit na několik částí (tolik, kolik algoritmů se na analýze podílí). Startovací neterminál každé této gramatiky potom může být součástí jedné nebo více jiných gramatik (podmínky lze definovat přesněji a formálněji, je to však mimo rozsah tohoto příspěvku).

Gramatickou skladbu detailů, jež bude nakonec analyzovat cizí algoritmus, není nutno řešit na úrovni klientské gramatiky. Gramatika pro rekurzivní sestup jistě obsahuje neterminál *Expression*, ovšem jeho derivace již součástí této gramatiky nejsou. To je popsáno v úplně jiné gramatice, potenciálně úplně jiné mocnosti (v tomto případě se jedná o CFG pro precedenční analyzátor, což není obecná CFG).

Určité detaily gramatiky serveru však musí znát i klient. Například pokud jsou klíčovými slovy závorky, musí si být vědom jakým způsobem se párují, kde se mohou vyskytovat, atd.

2.3 KONTROLA SYNTAKTICKÝCH CHYB

První zodpovědností analyzátoru je zotavení z chyby – překladač nesmí havarovat, když narazí na neplatný řetězec. Musí svůj vnitřní stav uvést do konzistence a pokračovat v překladu, aby mohl potenciálně odhalit nějaké další chyby. Další zodpovědností je poskytnutí užitečných diagnostických hlášení. Obě tyto povinnosti spolu souvisejí, překladač který se nedokáže zotavit obvykle zahrne uživatele lavinou naprosto nesmyslných chybových hlášení.

Jedna z možností automatického zpracování chyb je syntaxí řízené zotavení [1] pro analýzu metodou zdola nahoru. Přestože se jedná o obecný algoritmus, programátor může do jazyka promítnou znalost chyb, jež v něm budou uživatelé typicky dělat. Udělá to už na úrovni gramatiky pomocí takzvaných chybových derivací, tj. derivací, které sice v gramatice jsou, ale reprezentují chybné řetězce. Mimo to algoritmus i pro ostatní případy dokáže odhadnout co měl autor rozkládaného řetězce na mysli, a podle nejpodobnějšího derivačního pravidla může chybové hlášení vygenerovat autormaticky (a, též důležité, upravit podle toho svůj vnitřní stav).

Metoda syntaxí řízeného zotavení je však poměrně křehká a vyžaduje zálohu ve formě hrubého zotavení, které bude fungovat vždy – tím může být např. prosté spolykání celé syntaktické jednotky. Mimo to ji nelze použít v RD analyzátoru, kde není žádný explicitní zásobník, s nímž by bylo možno pracovat.

Nejnadějněji vypadá možnost kombinace obou přístupů. V RD nám nezbude, než se spolehnout na ad-hoc řešení, v precedenční analýze však lze použít syntaxí řízené zotavení přímo.

2.4 KONTEXTOVÁ OMEZENÍ

Podle shora uvedených poznámek je chyba úzce svázána s gramatikou jazyka. Nicméně detaily jako typové informace, definovanost a inicializovanost proměnných lze v gramatice zachytit jen velice obtížně. Jednu možnost představují atributové gramatiky, jinou neformální definice sémantiky (“oba operandy musejí být stejného typu”).

Implementace kontextových kontrol však je vposledku stejná, a skutečně se jedná o dodatečná omezení uvalená na vytvořený derivační strom. Každé z derivačních pravidel je aplikovatelné vždy, bez ohledu na kontext, a jeho validita je kontrolována až dodatečně.

Kontextové podmínky lze testovat už ve chvíli tvorby uzlů syntaktického stromu (v případě OO implementace např. v konstruktorech), nebo je lze odložit a řešit je později, až je celý strom hotov. Záleží na struktuře jazyka, zda kontextové informace potřebuje pro konstrukci syntaktického stromu, odvozování dalšího derivačního či redukčního kroku, atd.

Také při kontrole kontextových omezení je potřeba produkovat chybová hlášení. Z tohoto důvodu potřebují jednotlivé uzly syntaktického stromu nést informaci o svém umístění ve zdrojovém kódu.

3 ZÁVĚR

V příspěvku byly nastíněny problémy se zvolenou metodou syntaktické analýzy, a jak to jen rozsah dovoluje, i návrh jejich konkrétního řešení. Vzhledem k brzkému stadiu projektu nejsou zatím známy experimentální výsledky.

Další práce na projektu zahrnuje zejména implementační činnost a srovnání výkonnosti s automaticky generovaným LALR parserem či čistým RD parserem.

LITERATURA

[1] Tremblay, J. P., Sorenson, P. G.: The Theory and Practice of Compiler Writing, Mcgraw-Hill College (May 1, 1985), ISBN: 0-070-65161-2