Projekt PJP 2004/2005

Autoři

Václav Lukáš – luk117
Vladimír Schäfer – sch110

1. Úvod

Projekt obsahuje dvě kompletní řešení zadání za použití různých postupů:

1) Rekurzivní sestup (sch110)
2) Vyhodnocování pomocí zásobníkového automatu (luk117)

Obě řešení používají společný lexikální analyzátor.

Postup vývoje včetně autorů jednotlivých částí kódů lze vysledovat ze CVS na adrese:
http://cvs.berlios.de/cgi-bin/viewcvs.cgi/xds/pjp/src/

Kompilace:
Po rozbalení spustíme příkaz „ant“. Projekt vyžaduje **Java SDK 1.5** eventuelně JRE 1.5

Spouštění:

- Pro použití rekurzivního sestupu:
```
java –jar pjp.jar a source
```

- Pro použití zásobníkového automatu:
```
java –jar pjp.jar b source
```

Testovací soubor:
Pro spuštění testovacího skriptu zadejte:
```
java –jar pjp.jar a data/testscript.txt
java –jar pjp.jar b data/testscript.txt
```

2. Lexikální analyzátor

Dynamicky se generuje ze souboru pravidel (rules.lex), který obsahuje popisy konečných automatů pro jednotlivé symboly. Počáteční stavy automatů symbolů jsou spojeny e přechody s novým globálním počátečním stavem – je vytvořen nedeterministický automat, který je následně převeden na deterministický.

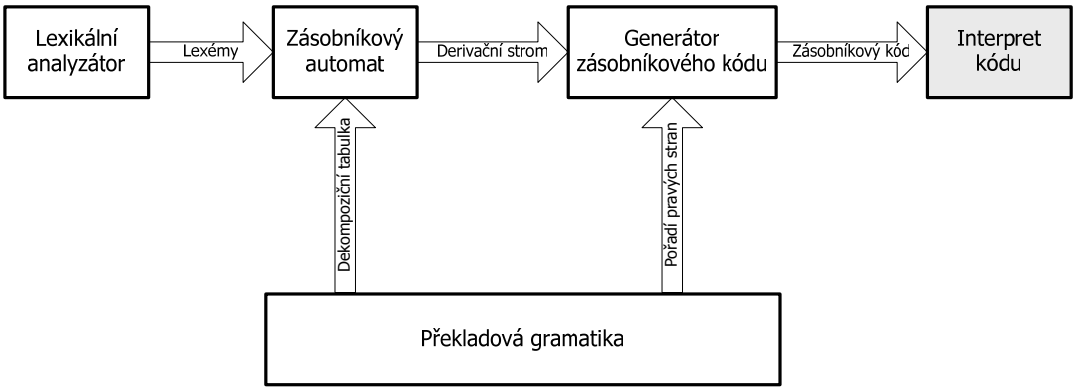
Zotavení je implementováno vynecháváním chybných znaků.

3. Rekurzivní sestup

První řešení realizuje překladač pomocí rekurzivního sestupu. Každému nonterminálu přísluší jedna metoda, vyhodnocování začíná od počátečního nonterminálu.

Je použito Hartmannovo schéma zotavení, kde množinu kontext tvoří terminály z množiny follow rozpracovaných nonterminálů.

4. Implementace pomocí zásobníkového automatu



Postup při sestavování zásobníkového kódu:

1. Načtení gramatiky, výpočet množin FIRST, FOLLOW a sestavení dekompoziční tabulky pro zásobníkový automat
2. Načtení a zpracování slova pro zásobníkový automat. Pro každé slovo (příkaz) je sestaven derivační strom
3. Převod derivačního stromu na instrukce a data (využije se překladové gramatiky) a sestavení zásobníku.

Takto sestavený kód se následně zpracuje klientskou třídou zděděnou od abstraktní třídy ClientStackCodeProcessor.

Toto řešení vyžaduje speciální překladovou gramatiku, která je uložena v souboru `main-grammar.txt`. Tato gramatika je vytvořena pro jediný příkaz, nedochází tedy k rekurentnímu zanořování jako u implementace rekurzivním sestupem. Pravá strana gramatiky obsahuje definice pořadí pravidel tak, jak se mají vyhodnocovat v derivačním stromu.

5. Použitá gramatika

STATEMENT	var DECLARATION ; STATEMENT | ident B ; STATEMENT | ; STATEMENT | ε
DECLARATION	ident D
D	, DECLARATION | : type
B	:= ASSIGN | (FUNC)
ASSIGN	= EXPR
FUNC	EXPR FUNC2;
FUNC2	, EXPR FUNC2; | ε
EXPR	E E1;
E	H T1;
H	- H | not H | H1 | stringval
H1	numberint | ident | numberdouble | true | false | (EXPR)
T1	* H T1 | / H T1 | mod H T1 | and H T1 | ε
E1	G E2
G	+ E G | - E G | concat E G | or E G | ε
E2	relation E G E2 | ε