Yet Another Parser Generator
Informace pro uživatele

Režim make

V tomto režimu může uživatel předložit programu soubor s definicí nějaké gramatiky a nechat si ji zkompilovat do binárních dat, pomocí kterých lze později rychle inicializovat lexer/parser duo a parsovat text v gramatikou popsaném jazyce.

yapg make *inputFile* [**-o** *outputFile*] [**-l** *logFile*] [**-f**]

Yapg načte gramatiku z inputFile, zkompiluje ji a uloží výstup do outputFile (pokud nebyl outputFile určen, data se zapíší do inputFile.par). Pokud nebude určen logFile, program v případě existence LALR(1) konfliktů v parseru vypíše do souboru inputFile.html všechny stavy automatu a odůvodnění nalezených konfliktů. Pokud bude logFile určen explicitně, bude do něj program zapisovat podobu automatu i v případě úspěšného vytvoření parseru. Uživatel může také nastavením optionu **-f** vynutit výpočet LALR(1) lookahead množin pro všechny konfliktní itemy parseru, tj. i pro ty, jejichž konflikty by bylo možno vyřešit s pomocí SLR(1) lookahead množin. Obecně se tak prodlouží výpočet programu, výstupní parser by však měl přesněji popisovat chyby ve zpracovávaném vstupu (konkrétně řečeno bude schopen přesněji určit množinu očekávaných symbolů).

Jak má vypadat specifikace gramatiky předkládaná režimu make?

V první části jsou obsaženy data pro lexer popisující podobu terminálních symbolů. Uživatel může nejprve zadefinovat jméno terminálního symbolu, který v případě, že bude lexerem nalezen, nebude předložen parseru, ale bude zahozen. Takovýto odpadový terminál si může uživatel zadefinovat na začátku specifikace následujícím způsobem

%null *identifikátor*

Lexer scanuje tokeny pomocí .NETího Regex enginu, u kterého se dá nastavit několik různých režimů (multiline, ignorecase..., viz. .NETí dokumentace). Tyto optiony může uživatel nastavit globálně pro celý lexer použitím regexí konstrukce **(?***optiony***)**, která je zdokumentována a popsána v .NETí dokumentaci v sekci o Miscellaneous constructs. Tato část je stejně jako definice odpadového terminálu nepovinná.

Dále následuje definice terminálních symbolů.

*identifikátor***=***regulární_výraz*

Tímto se zadaný regulární výraz přiřadí k řečenému terminálu. Každý terminál k sobě může mít přiřazen více než jeden regulární výraz. Na pořadí regulárních výrazu záleží, jelikož lexer bude hledat výše popsané regulární výrazy dříve a vrátí token s terminálem, kterému přísluší nejvyšší regulární výraz, jež bylo možno z dané pozice najít. Regulární výrazy smí používat veškeré vlastnosti, o kterých se píše v .NETí dokumentaci, jen by se měly vyvarovat používání pojmenované skupiny, jejichž jména jsou tvaru **__***i*, kde *i* je celé číslo, neměly by používat shodně pojmenované skupiny ve dvou různých regulárních výrazech a zároveň se nemůžou spolehnout na číslování skupin.

Nyní už se můžeme pustit do „neterminální“ úrovně jazyka. Další povinnou součástí gramatiky je definice počátečního symbolu, která vypadá následovně.

%start *identifikátor*

Parser se pak bude snažit najít složkový stromek, jež bude mít tento neterminální symbol v kořeni.

Zbytek gramatiky už tvoří pouze přepisovací pravidla. Přepisovací pravidla musí být zapsány ve standardní notaci BNF, pouze je uživatel povinen ukončit každé přepisovací pravidlo středníkem, jinak by nešel zápis v BNF notaci parsovat rozumnou LR(1) gramatikou. Prázdný řetězec na pravé straně se nijak explicitně neznačí, pouze se rovnou ukončí pravidlo středníkem, případně se napíše ořítko, pokud mají následovat další varianty pro pravou stranu. Veškeré identifikátory mohou sestávat z tzv. slovních znaků (\w neboli a-zA-Z0-9). Gramatika může obsahovat jednořádkové komentáře, které vždy začínají znakem **#**.

Režim run

V tomto režimu si může nechat uživatel rozparsovat text v jazyce popsaném nějakou zkompilovanou gramatikou.

yapg run *parserFile* [**-x** *xmlOutputFile*]* [**-a** *asciiTreeOutputFile*]*

Program načte z *parserFile* data lexeru a parseru v binární podobě (vytvořená předem pomocí režimu make anebo uživatelským programem využívajícím **yapg** jako knihovnu) a inicializuje lexer a parser. Program pak očekává na standardním vstupu text, který po přečtení rozparsuje. XML reprezentaci složkového stromu uloží do souborů *xmlOutputFile* a ASCII „obrázkovou“ reprezentaci do souborů *asciiTreeOutputFile*. Místo názvu souboru lze použít **-**, což značí standardní výstup. Pokud uživatel neurčí žádný výstupní soubor, program implicitně vypisuje „ASCII“ reprezentaci složkového stromu na standardní výstup.