PARSING OF CONTEXT-SENSITIVE LANGUAGES

Lukáš RYCHNOVSKÝ, Master Degree Programme (5)
Dept. of Information Systems, FIT, BUT
E-mail: xrychn01@stud.fit.vutbr.cz

Supervised by: Doc. Dušan Kolář

ABSTRACT

Parsing of context-sensitive languages is impossible with conventional parsing techniques for context-free languages. This article shows how to parse useful context-sensitive languages with Scattered Context Grammars.

1 ÚVOD

Tento text navazuje na článek Doc. Koláře a prof. Meduny Regulated Pushdown Automata [1]. Je zde citován výsledek a navržena modifikace regulovaných zásobníkových automatů pro praktické využití pro parsing context-sensitive jazyků.

2 DEFINICE

Definici zásobníkováho automatu a Turingova stroje, jejich konfigurace, přechody a další vlastnosti lze nalézt např. v [2].

Definice 2.1. Gramatika s roztroušeným kontextem (Scattered Context Grammar) (SCG) G je uspořádaná čtveřice (V,T,P,S), kde V je konečná množina symbolů, $T \subset V, S \in V \setminus T$ a P je konečná množina pravidel ve tvaru $(A_1,...,A_n) \rightarrow (w_1,...,w_n), n \geq 1, \forall A_i : A_i \in V \setminus T, \forall w_i : w_i \in V^*$.

Definice 2.2. Nechť $G = (V,T,P,S)$ je SCG. Nechť $(A_1,...,A_n) \rightarrow (w_1,...,w_n)$ je z P a pro $1 \leq i \leq n+1$ nechť $x_i \in V^*$. Pak píšeme

$$x_1A_1x_2A_2...x_nA_nx_{n+1} \Rightarrow x_1w_1x_2w_2...x_nw_nx_{n+1}$$

$\Rightarrow^*$ je reflexivní, tranzitivní uzávěr $\Rightarrow$.
Jazyk generovaný gramatikou G je definován jako $L(G) = \{w \in T^* | S \Rightarrow^* w\}$.

Definice 2.3. Regulovaný zásobníkový automat (Regulated Pushdown Automata) (RPDA) je zásobníkový automat $M = (Q,\Sigma,\Omega,R,s,S,F)$, kde množina pravidel R je rozšířena o labely a doplněný o kontrolní jazyk Ξ. Změny stavů automatu M generují posloupnost labelů $[\sigma_1...\sigma_n]$ která tvoří slovo z jazyka Ξ. Přesnou definici lze nalézt v [2].

3 KONTEXTOVÉ PRVKY V BEZKONTEXTOVÝCH JAZYCÍCH

V práci [2] Doc. Koláře je navržen postup převodu lineárně omezeného Turingova stroje (LBTM) na RPDA. Pomocí tohoto algoritmu můžeme tedy kontextové jazyky parsovat metodami známými pro bezkontextové jazyky za pomoci regulovaného zásobníkového automatu. Tento postup však i pro ty nejjednodušší kontextové jazyky dává obrovské množství pravidel, jak pro RPDA, tak pro jeho kontrolní jazyk. Například pro LBTM, který dekadické číslo na pásce zvětší o 1, dostáváme přibližně 300 pravidel v RDPA a přes 6 000 pravidel v kontrolním jazyce Ξ. Pro složitější LBTM je tento přístup velice nepohodlný.

Cílem tohoto článku je navrhnout rozšíření stávajících bezkontextových jazyků (Pascal, C/C#, Java, ...) o kontextové prvky přímo v jejích gramatikách. Zároveň zachovat jednoduchost a přímočarost parsingu.

Jako příklad bezkontextového jazyka nám poslouží jazyk ZAP03 [3], který je svojí syntaxí velice podobný Pascalu. Z gramatiky ZAP03 (přibližně 70 pravidel) budeme potřebovat následující fragment popisující definici proměnných

`DCL` $\rightarrow$ `TYP [:] [id] ID_LIST`
`ID_LIST` $\rightarrow$ `[,] [id] ID_LIST`
`ID_LIST` $\rightarrow \varepsilon$

přiřazovací příkaz

`PRIKAZ` $\rightarrow$ `[id] TEST PRIKAZ`
`TEST` $\rightarrow$ `[=] VYRAZ [;]`

a konečně použití proměnné v příkazu

`OPER` $\rightarrow$ `[id] TEST2`
`TEST2` $\rightarrow \varepsilon$.

Symboly v závorkách [] představují terminály.

Nyní upravíme gramatiku následovně. Vezměme pravidla jazyka ZAP03 a výše uvedené nahraďme těmito SC pravidly:

`(DCL, S')` $\rightarrow$ `(TYP [:] [id] ID_LIST, D)`
`(ID_LIST, S')` $\rightarrow$ `([,] [id] ID_LIST, D)`
`(ID_LIST)` $\rightarrow (\varepsilon)$

přiřazovací příkaz

`(PRIKAZ, D)` $\rightarrow$ `([id] TEST PRIKAZ, DL)`
`(TEST)` $\rightarrow$ `([=] VYRAZ [;])`

a konečně použití proměnné v příkazu

`(OPER, D)` $\rightarrow$ `([id] TEST2, DR)`
`(TEST2)` $\rightarrow (\varepsilon)$.

Parsing nyní bude probíhat následovně: startovní symbol bude SS$'$, parsing bude probíhat standardní cestou, až do chvíle, kdy bude zpracováván nonterminál DCL. V tomto okamžiku se S$'$ přepíše na D, což znamená, že proměnná jménem [id] byla definována. Při použití této proměnné na levé, resp. pravé straně přiřazení je D přepsáno na DL, resp. DR. Pokud je proměnná [id] použita bez definice, dojde k chybě při parsování, neboť S$'$ se nemůže přepsat na DL ani DR. Po ukončení parsování se původní startovací symbol přepíše na ε a v derivaci zbude posloupnost $D\{LR\}^*$. Pokud obdržíme pouze symbol D, pak [id] byl nadefinován, ale nebyl použit, v případě DL^+ víme, že [id] byl nadefinován, ale nebyl použit pro čtení, pouze pro zápis. V obou případech lze oznámit "Parse warning".

Tento postup umožňuje zpracování pouze jedné proměnné [id]. Není však problém uvedený postup rozšířit na libovolný počet proměnných.

Protože byl původní jazyk LL1 a uvedenou modifikací nebylo žádné pravidlo přidáno, bude i tento jazyk mít jednoznačné derivace.

Je tedy vidět, že pomocí jednoduchého SC rozšíření bezkontextového jazyka, můžeme dostat gramatiku, která již při parsingu reaguje na použití nedefinovaných proměnných, nepoužití definovaných proměnných nebo výskyt proměnných, jejichž hodnota není v kódu použita. Pomocí stejně jednoduchých kontextových rozšíření můžeme například v jazyce typu Java kontrolovat, že abstraktní metoda není zároveň označena jako final. Dalším příkladem může být zavedení klíčového slova OBSERVER, které bude vylučovat modifikaci proměnné this.

4 FORMÁLNÍ VERIFIKACE PROGRAMŮ

Dalším rozšířením jazyka ZAP03 bude přidání tzv. preconditions resp. postconditions. Jedná se o množinu logických formulí, jejichž platnost požadujeme při vstupu výpočtu do programu či metody, resp. při opouštění. Například při výpočtu odmocniny z x lze precondition formulovat jako $\{x >= 0\}$ a při výpočtu y = sin(x) lze postconditin zapsat jako $\{(y <= 1) \wedge (y >= -1)\}$. Příkazy jazyka nám pak definují postupně transformace na množině preconditions. Například přiřazení V := E definuje transformaci $\{P[E/V]\}V := E\{P\}$, kde V je proměnná, E je výraz, P je precondition a P[E/V] značí nahrazení všech výskytů V v P za E. Transformace precondition dalšími příkazy lze najít v [4].

Cílem našeho snažení je mít jazyk a v něm program, kde z preconditions a transformací definovaných jednotlivými příkazy budou přímo odvoditelné postconditions. Takovýto program potom přímo v sobě obsahuje formální důkaz své správnosti.

V jazyce ZAP03 by toho bylo možno dosáhnout přidáním klíčového slova PRE pro definici precondition, POST pro postcondition a přidáním transformačních předpisů k jednotlivým pravidlům gramatiky. Na počátku parsingu by se zpracovaly preconditions a při průchodu programem by se automaticky aplikovaly příslušné transformace. Pokud by po ukončení parsingu transformované preconditions odpovídaly postconditions, parsování proběhlo v pořádku.

Při formulování preconditions a postconditions však platí jisté podmínky. Postconditions musí být přímo transformací preconditons, neboť jinak narážíme na Geodelovu větu. Postconditions mohou být pravdivé, ale mohou být z transformovaných preconditions nedokazatelné.

REFERENCE

[1] Meduna, A., Kolář, D.: Regulated Pushdown Automata, Acta Cybernetica, 14/2000.

[2] Kolář, D.: Pushdown Automata: New Modifications and Transformations, 2004.

[3] http://www.fit.vutbr.cz/study/courses/ZAP/public/project/

[4] Gordon, J. C. M.: Programming Language Theory and its Implementation, 1998.