BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2023
http://excel.fit.vutbr.cz

Alternative deep pushdown automata and their aplications

Radovan Klembara*

Abstract
The goal of my thesis is to design new versions of deep pushdown automata, their analysis and aplication in informatics. These new versions should have better properties than basic deep pushdown automata. I have created two new versions of automata. The idea behind modificatons is to add parallel processing of expansive transitions. For each version I have created algorithms, which can be used to converse these versions to basic deep pushdown automata. Furhtermore I have created algorithms for converting basic deep pushdown automata back to these new versions. Proposed alternative versions of deep pushdown automata are faster than their basic version thanks to the paralelism. Through their analysis I have found out that their strength is same as strength of basic deep pushdown automata. Solution of this thesis allows designing parallel version of deep pushdown automaton instead of basic deep pushdown automaton, that can result in faster and more efective design. It is possible to use these version in bioinformatics in searching for DNA repetitions, RNA psudoknots and more.

*radovan@klembara.pro, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Theoretical informatics gives many models that accept languages, but the number of alternative formall grammars is far greater than number of automatons. This fact convinced me to create new versions of automata. I have decided to modify deep pushdown automata because they are as strong as n-limited state grammars and they have been created by simple idea of deeper access to the pushdown.

The goal is to modify deep pushdown automata to make them faster and easier to use. The new automata have to be at least properly defined. Additionally it should be proven that they have better properties than the basic deep pushdown automata. Then it is required to find possible their applications in bioinformatics.

I have created two modifications, by adding parallelism to deep pushdown automata. First solution can make number of simultaneous expansions maximally equal to depth of automata. The other solution specifies number of parallel expansions to depth of automata. For these automata I have created algorithms that conver them to basic version and vice versa. These automata are as strong as deep pushdown automata and can be used for example in searching DNA repetitions.

2. Deep pushdown automata

The deep pushdown automata were presented in Acta Informatica by professor Meduna. The idea behind deep pushdown automata is that they allow access to symbols that are deeper inside the pushdown than the top. Other than that the deep pushdown automaton works exactly like any other pushdown automaton.

Deep pushdown automaton (PD_n) is defined as a septuple $M = (Q, \Sigma, \Gamma, R, s, S, F)$, where Q is a finite set of states, Σ is an input alphabet, Γ is a pushdown alphabet, $\Sigma \subseteq \Gamma$, $\Gamma \setminus \Sigma$ contains a special *bottom* symbol denoted by $\#$, R is finite relation, $R \subseteq (\mathbb{N}^+ \times Q \times (\Gamma \setminus (\Sigma \cup \{\#\})) \times Q \times (\Gamma \setminus \{\#\})^+) \cup (\mathbb{N}^+ \times Q \times \{\#\} \times Q \times (\Gamma \setminus \{\#\})^* \{\#\})$, s is start state, $s \in Q$, S is the start pushdown symbol, $S \in \Gamma$, F is a set of final states, $F \subseteq Q$.

The finite relation R is also refered as a set of rules. The pop rules of the automaton are implicit. Instead of $(m, q, A, p, v) \in R$, we write $mqA \to pv$. If rules of automaton M access symbols located maximally in depth n then we can denote the automa-

ton as M_n. For every $n \geq 1$, ${}_{deep}\mathbf{PD}_n$ denotes the family of languages defined by the deep pushdown automata of depth i, where $1 \leq i \leq n$. Let **CF** and **CS** denote the families of context-free and context-sensitive languages, respectively. Then the expresion ${}_{deep}\mathbf{PD}_1 = \mathbf{CF}$ is valid. Deep pushdown automata create infinite hierarchy of language families. This hierarchy is equivalent to hierarchy created by n-limited state grammars. This means that the expresion $\mathbf{CF} = {}_{deep}\mathbf{PD}_1 \subset {}_{deep}\mathbf{PD}_2 \subset \cdots \subset {}_{deep}\mathbf{PD}_n \subset {}_{deep}\mathbf{PD}_\infty = \mathbf{CS}$ is valid [1].

3. Parallel deep pushdown automata

First alternative deep pushdown automata introduces parallelism. Atomaton works just like basic deep pushdown automaton with a small change in expansion rules. The modification allows automata to simultaneously replace up to n non-input symbols, where n is depth of automaton.

Parallel deep pushdown automaton (PPD_n) is defined as a septuple $M = (Q, \Sigma, \Gamma, R, s, S, F)$, where elements are defined in the same way as in PPD_n except for R. R is finite relation, $R \subseteq (Q \times (\Gamma \setminus (\Sigma \cup \{\#\}))^n \times Q \times (\Gamma \setminus \{\#\})^+)^n$. The rules are writen as $q(A_1, A_2, \ldots, A_k) \to p(\alpha_1, \alpha_2, \ldots, \alpha_k)$.

3.1 Simulation of parallel automaton by basic automaton

The basic deep pushdown automaton can simulate simultaneous expansions by sequence of sub-rules. Where one parallel rule that replaces k non-input symbols must be simulated by k sub-rules. Additionally these sub-rules have to be used in order in which the deepest accessing rule has to be used first and the shallowest is used last. This is required because in other way the more shallow expansion could push other non-input symbols beyond the maximal depth. That would mean that it wouldn't be possible to use last sub-rule. To create PD_n that simulates PPD_n it is possible to use Algorithm 1.

3.2 Simulation of basic automaton by parallel automaton

The PPD_n can simulate PD_n by replacing non-input symbols with themselves exept for one symbol. For each PD_n rule $mqA \to pv$, PPD_n replaces each non-input symbol in smaller depth than m with itself, while it can expand mth symbol to v. To create PPD_n that simulates some PD_n it is possible to use Algorithm 2. For every PD_n it is possible to create PPD_n that accepts same language as shown higher. Algorithm 2 proofs that even for every PPD_n it is possible to create PD_n that accepts same language. Thanks to this, it is valid to say that family of languages generated by parallel deep pushdown automaton (${}_{deep}\mathbf{PPD}_n$) with depth n is equivalent to ${}_{deep}\mathbf{PD}_n$.

4. Tuple parallel deep pushdown automata

This alternative version is variation of PPD_n. Tuple parallel deep pushdown automata specifies number of simultaneous expansions t in all rules uniformly. Another difference is that this automata has starting pushdown string instead of symbol.

Tuple parallel deep pushdown automaton ($TPPD_n$) is defined as a septuple $M = (Q, \Sigma, \Gamma, R, s, \Omega, F)$, where elements are defined in the same way as they are in PPD_n except for R and Ω. R is finite relation, $R \subseteq (Q \times (\Gamma \setminus (\Sigma \cup \{\#\}))^n \times Q \times (\Gamma \setminus \{\#\})^+)^n$, where $n \in \mathbb{N}^+$ is depth of automaton. Ω is the start pushdown string, $\Omega \in ((\Gamma \setminus \Sigma) \setminus \#)^i$, where $i \geq n$. The rules are written as $q(A_1, A_2, \ldots, A_n) \to p(\alpha_1, \alpha_2, \ldots, \alpha_n)$.

4.1 Simulation of tuple parallel automaton by basic automaton

The idea behind simulation of $TPPD_n$ by PD_n is similar to the simulation of PPD_n by PD_n. The simultaneously expanded non-input symbols are expanded one by one from the deepest to the shallowest. The basic automaton has to expand its starting symbol to starting string of $TPPD_n$ right at the start. To create PD_n that simulates some $TPPD_n$ it is possible to use Algorithm 3.

4.2 Simulation of basic automaton by tuple parallel automaton

The reversed simulation also copies the idea from PPD_n. The $TPPD_n$ rules expand non-input symbols to itself, but the one that was reffered by PD_n. $TPPD_n$ works over n non-input symbols so it creates Ω with at least n "filling" symbols. These symbols are used as as empty spaces for basic rules accessing non-input symbols shallower than the depth of automaton. To create such automaton it is possible to use Algorithm 4. For every PD_n it is possible to create $TPPD_n$ that accepts same language as shown higher. Algorithm 4 proofs that even for every $TPPD_n$ it is possible to create PD_n that accepts same language. Thanks to this, it is valid to say that family of languages generated by tuple parallel deep pushdown automaton (${}_{deep}\mathbf{TPPD}_n$) with depth n is equivalent to ${}_{deep}\mathbf{PD}_n$.

5. Applications of alternative versions

Thanks to the parallel aproach it is easier to use alternative versions in cases when it is needed to do some expansion on two or more different places. This comes handy in bioinformatics. For example the DNA is responsible for passing genetic information coded inside itself. The molecule of DNA is made of nukleotides (adenin, guanin, cytosin, thymin), which are connected in single line. Thanks to this DNA can be represented by a string created over a set of the first letters of each nucleotide. The DNA can contain repetitions of sub-strings which can lead to instability of DNA, therefore it is important to know whether the DNA contains repetitions. For this it is possible to use PPD_n.

References

[1] Alexander Meduna. Deep pushdown automata. *Acta Informatica*, 2006(98):114–124, 2006.