Excel@FIT 2015
http://excel.fit.vutbr.cz

Issues related to parsing based on parallel communicating pushdown automata systems

Jakub Šoustar*

Abstract
In this paper we consider the use of parallel communicating pushdown automata systems (PCPA), that communicates their stacks on request, in parsing. We will present some issues that would accompany their use, and also present some possible ways to deal with those issues. New implicit transition mapping for pushdown automata, that are used as a components of a PCPA, will be presented. This new property is aimed at dealing with the issues, that are caused by new requirements which a PCPA poses on pushdown automata.

Keywords: parsing – parallel – communicating – pushdown – automata

Supplementary Material: Downloadable Code

*xsoust02@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

In [1] Erzsébet Csuhaj-Varjú et al introduced parallel communicating pushdown automata systems which on request communicate their stacks. These systems compose of several pushdown automata, each called a component, that work in parallel to accept a language. Because parallel communicating pushdown automata systems can be considered as a accepting counter part of generative parallel communicating grammar systems [2], they use similar communication strategy. When a component is in need of an information from another component (contents of its stack), it places an appropriate query symbol on top of its stack. This action forces a system to perform a communication step, which results in to the query symbol being replaced by the stack of a component that is associated to that symbol.

Carlos Martín-Vide et al also introduced parallel communicating finite automata systems that communicate via states in [3]. This paper will focus only on parallel communicating automata systems whose components are pushdown automata, because they are able to accept languages that are not context-free, as proved in [1].

Parallel communicating pushdown automata systems can be divided into four categories which were defined in [1]:

1. RCPCPA for returning centralized parallel communicating pushdown automata systems.
2. RPCPA for returning parallel communicating pushdown automata systems.
3. CPCPA for centralized parallel communicating pushdown automata systems.
4. PCPA for non-returning non-centralized parallel communicating pushdown automata systems.

When PCPA is returning, it means that after a component sends its stack to another, its stack is reset to its initial state. Non-returning PCPA maintains contents of its stack after communication. Centralized PCPA is a system where only one dedicated component can request communication as opposed to non-centralized systems where any component can request communication.

There are several properties of PCPA, that are of interest for parsing. One is their increased accepting power – we can use these systems to accept larger family of languages. Also, following their name, these systems are already defined as parallel – it's opening opportunities to use this parallelism in the process of parsing to, for example, increase their performance.

Currently, most of the programming languages are defined as context-free – mainly because they can be

easily analyzed, and formal models for generating and accepting such languages are well documented. It is also quite common that they contain syntactical structures, that are not entirely context-free, for example multiple assignment (`a, b = 1, 2`). These structures are usually handled on multiple levels of a compiler or an interpreter, such as syntactic and semantic analysis. We propose that PCPA do not replace the currently used models in parsing, but that they are used together – creating heterogeneous systems capable of handling cases like the multiple assignment.

2. Background

We assume that the reader is familiar with basic concepts of formal languages and automata theory, particularly notions of grammars and pushdown automata. More details can be found in [4].

Firstly, we will define parallel communicating pushdown automata systems as introduced in [1].

A parallel communicating pushdown automata system of degree n is a construct

$$\mathcal{A} = (V, \Delta, A_1, A_2, \ldots, A_n, K),$$

where

- V is the input alphabet,
- Δ is the alphabet of pushdown symbols,
- for each $1 \leq i \leq n, A_i = (Q_i, V, \Delta, f_i, q_i, Z_i, F_i)$ is a pushdown automaton with set of states Q_i, the initial state $q_i \in Q_i$, the alphabet of input symbols V, the alphabet of pushdown symbols Δ, the initial contents of stack $Z_i \in \Delta$, the set of final states $F_i \subseteq Q_i$, and the transition mapping f_i from $Q_i \times V \cup \{\varepsilon\} \times \Delta$ into the finite subsets of $Q_i \times \Delta^*$,
- $K \subseteq \{K_1, K_2, \ldots, K_n\} \subseteq \Delta$ is the set of query symbols.

The automata $A_1, A_2, \ldots, A_n$ are called components of the system $\mathcal{A}$.

If there exists only one component $A_i, 1 \leq i \leq n$, such that $(r, \alpha) \in f_i(q, a, A)$ with $\alpha \in \Delta^*, |\alpha|_K > 0$, for some $r, q \in Q_i$, $a \in V \cup \{\varepsilon\}$, $A \in \Delta$, then the system is said to be *centralized* and A_i is said to be *master* of the system. For the sake of simplicity, whenever a system is centralized its master is the first component.

Configuration of parallel communicating pushdown automata system $\mathcal{A}$ is a $3n$-tuple

$$(s_1, x_1, \alpha_1, s_2, x_2, \alpha_2, \ldots, s_n, x_n, \alpha_n)$$

where for $1 \leq i \leq n$,

- $s_i \in Q_i$ is the current state of the component A_i,
- $x_i \in V^*$ is the remaining part of the input word which has not yet been read by A_i,
- $\alpha_i \in \Delta^*$ is the stack of component A_i, its first symbol being the topmost symbol.

There are two variants of transitions on the set of configurations of $\mathcal{A}$, defined in the following way:

1. $(s_1, x_1, B_1\alpha_1, s_2, x_2, B_2\alpha_2, \ldots, s_n, x_n, B_n\alpha_n)$ $\vdash (p_1, y_1, \beta_1, p_2, y_2, \beta_2, \ldots, p_n, y_n, \beta_n)$, where $B_i \in \Delta$, $\alpha_i, \beta_i \in \Delta^*$, $1 \leq i \leq n$, iff one of the following two conditions hold:
   (a) $K \cap \{B_1, B_2, \ldots, B_n\} = \emptyset$ and $x_i = a_i y_i, a_i \in V \cup \{\varepsilon\}, (p_i, \beta_i') \in f_i(s_i, a_i, B_i), \beta_i = \beta_i'\alpha_i, 1 \leq i \leq n$,
   (b)
   - for all i, $1 \leq i \leq n$ such that $B_i = K_{j_i}$ and $B_{j_i} \notin K$, $j \geq 1$, $\beta_i = B_{j_i}\alpha_{j_i}\alpha_i$,
   - for all other r, $1 \leq r \leq n$, $\beta_r = B_r\alpha_r$, and
   - $y_t = x_t$, $p_t = s_t$, for all t, $1 \leq t \leq n$.
2. $(s_1, x_1, B_1\alpha_1, s_2, x_2, B_2\alpha_2, \ldots, s_n, x_n, B_n\alpha_n)$ $\vdash_r (p_1, y_1, \beta_1, p_2, y_2, \beta_2, \ldots, p_n, y_n, \beta_n)$, where $B_i \in \Delta$, $\alpha_i, \beta_i \in \Delta^*$, $1 \leq i \leq n$, iff one of the following two conditions hold:
   (a) $K \cap \{B_1, B_2, \ldots, B_n\} = \emptyset$ and $x_i = a_i y_i, a_i \in V \cup \{\varepsilon\}, (p_i, \beta_i') \in f_i(s_i, a_i, B_i), \beta_i = \beta_i'\alpha_i, 1 \leq i \leq n$,
   (b)
   - for all i, $1 \leq i \leq n$ such that $B_i = K_{j_i}$ and $B_{j_i} \notin K$, $j \geq 1$, $\beta_i = B_{j_i}\alpha_{j_i}\alpha_i$, and $\beta_{j_i} = Z_{j_i}$,
   - for all other r, $1 \leq r \leq n$, $\beta_r = B_r\alpha_r$, and
   - $y_t = x_t$, $p_t = s_t$, for all t, $1 \leq t \leq n$.

Whenever a component has a query symbol on top of its stack, communication must take place. During communication no regular transitions can be performed. Before stack can be communicated, its topmost symbol must not be a query symbol. This can lead to a circular query which will block further action of the system.

After communication, the stack contents of the sending component remains unchanged in the case of relation $\vdash$, whereas it becomes initial pushdown symbol in the case of relation $\vdash_r$. A parallel communicating pushdown automata system whose computations are based on relation $\vdash$ is said to be *non-returning*; if its computations are based on relation $\vdash_r$ it is said to be *returning*. The language accepted by parallel communicating pushdown automata system $\mathcal{A}$ is defined

as

$$L(\mathcal{A}) = \{x \in V^* | (q_1, x, Z_1, \ldots, q_n, x, Z_n) \vdash^* (s_1, \varepsilon, \alpha_1, \ldots, s_n, \varepsilon, \alpha_n), s_i \in F_i, 1 \leq i \leq n\},$$
$$L(\mathcal{A}) = \{x \in V^* | (q_1, x, Z_1, \ldots, q_n, x, Z_n) \vdash_r^* (s_1, \varepsilon, \alpha_1, \ldots, s_n, \varepsilon, \alpha_n), s_i \in F_i, 1 \leq i \leq n\},$$

where $\vdash^*$ and $\vdash_r^*$ denote the reflexive and transitive closure of $\vdash$ and $\vdash_r$ respectively.

We use the following notations

RCPCPA(n) for returning centralized parallel communicating pushdown automata systems of degree n,

RPCPA(n) for returning parallel communicating pushdown automata systems of degree n,

CPCPA(n) for centralized parallel communicating pushdown automata systems of degree n,

PCPA(n) for parallel communicating pushdown automata systems of degree n.

3. Some PCPA related issues for parsing

In this section, we will discuss some issues that would be introduced into the process of parsing by using the parallel communicating pushdown automata systems. Also, the notations $card(M)$, meaning the number of elements of the set M, and $|M|_E$, meaning the number of occurrences of the element E in the set M, will be used. At first, we will look at the way the amount of non-determinism in systems components affects the whole system. After that, we will focus on the difference between regular pushdown automata and those used as a components in a PCPA. More specifically, we will focus on differences in the ways they reach their accepting state, and on changes in their behavior after they reach it.

3.1 General non-determinism in components of a PCPA

Without any further restrictions, any component can enter a configuration (s, x, α), $s \in Q$, $x \in V^*$, $\alpha \in \Delta^*$ such that $card(f(s, x, \alpha)) > 1$. That is, it can perform more than one transition from its current configuration.

Effect of this issue is even more heightened in situations, where it occurs in more than one component of a PCPA. In fact, the number of configurations a $pcpa(n)$ can transition into, is equal to

$$\prod_{i=0}^{n} card(f_i(s_i, x_i, \alpha_i)),$$

and the set of all possible configurations system can enter is given by

$$f_1(s_1, x_1, \alpha_1) \times f_2(s_2, x_2, \alpha_2) \times \cdots \times f_n(s_n, x_n, \alpha_n).$$

As we can see, even small amount of non-determinism in the systems components can have serious impact on whole system. When designing some PCPA, one must have in mind this issue, as it could possibly render the whole system too inefficient for any practical use.

3.2 Accepting state of a PCPA and accepting states of its components

A regular pushdown automaton and a component of a PCPA treat the input word differently. Regular pushdown automaton will continue reading the input word and either accept it or reject it – the input word either belongs to the language accepted by the pushdown automaton or it does not.

As we can see from the definition of a PCPA in 2, each of its components has to accept the input word, otherwise the PCPA as a whole can not accept it. But unlike the regular pushdown automaton, a component of a PCPA can use only portions of the input word to actually perform some computations, while the rest of the input word can serve as a synchronization mechanism. This is the reason behind the accepting power of a PCPA – its components analyze the input word, each one in its own way, and then they together produce a result.

Now, because each component of a PCPA can work differently to the others, it also means that they all can enter their accepting states after a different amount of performed transitions. But at that point, where regular pushdown automaton would successfully accept the input word and halt, a component must be able to continue working. If a component was to reach its accepting state, and it would not be able to perform any transition from such configuration, it could wrongfully cause the PCPA to reject the input word.

Reader can imagine that this issue can be actually quite easily solved using ε-transitions – a component could enter an infinite cycle where it would transition into the same configuration. This would cause the component to remain in its accepting state, and it would also remove the risk of unintentionally causing a system to reject the input word.

While there is formally nothing wrong with this solution, adding a new transition, especially ε-transition, could possibly introduce additional non-determinism into the system. Such situation could complicate the process of parsing, as we presented in 3.1. This issue also applies to every component of every PCPA, showing a limitation in the definition of pushdown automata in the context of parallel communicating pushdown automata systems.

4. Handling the previously presented issues

This section will present some possible solutions to the issues presented in 3. We will start with the general non-determinism in a PCPA and present two main approaches to this issue. At the end of this section, new type of transition for components of parallel communicating pushdown automata systems will be presented. This new transition allows the component to handle issues shown in 3.2, without posing additional requirements on its design or the threat of introducing non-determinism.

4.1 Dealing with general non-determinism in a PCPA

This issue can be dealt with by taking two approaches. System can either embrace non-determinism, and thus increase its accepting power at the expense of performance, or it can treat it as erroneous state.

4.1.1 Embracing non-determinism

Taking this approach can greatly increase the accepting power of a system. In fact, it was proven in [1], that a $PCPA(n)$ equals the family of recursively enumerable languages. For practical use, such accepting power could be unnecessarily strong. On the other hand, it could also allow the system to accept languages like

$$\{xx \mid x \in V^+\}$$

or

$$\left\{a^{2^n} \mid n \geq 1\right\}.$$

Decision whether to use this increased power or not, should be based upon analyzing the intended use of such system. The impact of non-determinism on the performance of a PCPA will grow with the length of the input word. When mainly analyzing relatively small strings, it's likely that the benefit, in form of the increased accepting power, would outweigh the performance cost. Also, for some concrete systems, additional heuristics could be used to reduce the amount of possible configurations that the system could enter.

The demonstration application that is a part of this paper, is mainly designed as a proof of concept. The application simulates the work of a PCPA defined by the user. There are no specific restrictions for those systems, though complicated or non-deterministic systems can cause serious performance issues.

4.1.2 Discarding non-determinism

Let $A = (Q, V, \Delta, f, q, Z, F)$ be a pushdown automaton. If $card(f(s, x, \alpha)) \leq 1$ for each $s \in Q$, $x \in V \cup \{\varepsilon\}$, $\alpha \in \Delta$, A is said to be a deterministic pushdown automaton.

By posing requirement that each component of a system is deterministic, we can easily see that the whole system would be also deterministic. For any configuration, there would be either none or one possible configuration that the system could transition into.

4.2 Introducing a new implicit transition mapping for components of a PCPA

Let $\mathcal{A} = (V, \Delta, A_1, \ldots, A_n, K)$ be a $PCPA(n)$ and let $A_i = (Q_i, V, \Delta, f_i, q_i, Z_i, F_i)$ be a component $i, 1 \leq i \leq n$ of $\mathcal{A}$. We define this new property for the components of $\mathcal{A}$ as

Definition 1. Let $(s_i, a_i x_i, B_i \alpha_i)$, where $s_i \in Q_i, a_i \in V \cup \{\varepsilon\}, x_i \in V^*, B_i \in \Delta, \alpha_i \in \Delta^*$, be a configuration of A_i. Then f_i implicitly contains transition mapping $f_i(s_i, a_i, B_i) = \{(s_i, B_i \alpha_i)\}$, iff all the following conditions hold:

- $B_i \notin K$,
- $(s_i, a_i x_i, B_i \alpha_i)$ is a configuration, in which A_i is in an accepting state, and
- $f_i(s_i, a_i, B_i) = \emptyset$.

This implicit transition mapping allows the component of a PCPA to remain in a configuration in which the component is also in an accepting state, allowing the component to reach its accepting state and giving it a ability to remain in such state. Reader can see that this transition acts as the **NOP** instruction known from assembly languages. Main benefit of this type of transition is that while it acts the same as the transition mapping $f_i(s_i, \varepsilon, \alpha_i) = \{(s_i, \alpha_i)\}$, it does so with lower priority – this transition can only be performed if no other can.

Use of this new property has two main benefits. It simplifies the design of a PCPAs components, by removing the need to explicitly specify transitions, that would allow the component to work even after it has logically fulfilled its function. Removal of those transitions can also reduce the amount of non-determinism in the component, acting as an additional benefit.

5. Conclusions

We have discussed some of the issues that would parallel communicating pushdown automata systems introduce into the process of parsing, and also looked at how these issues could be approached. New implicit transition mapping for the components of a PCPA was introduced as a solution for the difference between pushdown automata that are part of a PCPA, and those that are not.

We will end with a few ideas to investigate in future. Would it be possible to enhance a PCPA with a additional control mechanism, that would be able to reduce the amount of possible configurations, that a non-deterministic PCPA could enter? Also, at the moment, the system must perform transition as a whole, so that the communication step can be performed properly if needed. Would it be possible to detect the points where communication takes place in advance? This would allow the components to work more independently by relaxing the requirement to wait for others after each transition.

References

[1] Erzsébet Csuhaj-Varjú, Carlos Martín-Vide, Victor Mitrana, and György Vaszil. Parallel communicating pushdown automata systems. *International Journal of Foundations of Computer Science*, 11(04):631–650, 2000.

[2] Gh. Păun and L. Sântean. Parallel communicating grammar systems: the regular case. *An. Univ. Bucureşti, Ser. Matem.-Inform.*, 38(2):55–63, 1989.

[3] Carlos Martín-Vide, Alexandru Mateescu, and Victor Mitrana. Parallel finite automata systems communicating by states. *International Journal of Foundations of Computer Science*, 13(05):733–749, 2002.

[4] Grzegorz Rozenberg and Arto Salomaa, editors. *Handbook of Formal Languages*, volume 1-3. Springer-Verlag Berlin, 1997.