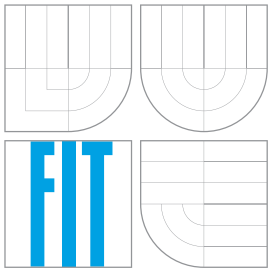
BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ



FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

REGULATED AUTOMATA SYSTEMS

REGULOVANÉ SYSTÉMY AUTOMATŮ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR
AUTOR PRÁCE
Bc. RADIM KRČMÁŘ

SUPERVISOR
VEDOUCÍ PRÁCE
prof. RNDr. ALEXANDER MEDUNA, CSc.

BRNO 2016

Abstract

This thesis defines and studies two new types of automata, cooperating distributed pushdown automata systems (CDPDAS) and parallel communicating pushdown automata systems (PCPDAS). CDPDAS and PCPDAS adapt the main concept of cooperating distributed grammar systems (CDGS) and parallel communicating automata systems (PCPDAS), respectively. CDPDAS are proven to have the same power as PDA and this thesis further explores the reason why CDPDAS do not increase power while CDGS do and introduces an automata system inspired by CDPDAS that does increase the power. PCGS have similar power as CDGS, but PCPDAS are equvalent with TM, which is proven by creating a communication protocol to access a second stack.

Abstrakt

Tato práce zavádí a studuje dva nové typy automatů, spolupracující distribuované systémy zásobníkových automatů (CDPDAS) a paralelní komunikující systémy zásobníkových automatů (PCPDAS), které jsou inspirovany spolupracujícími distribuovanými gramatickými systémy (CDGS), paralelními komunikujícími gramatickými systémy (PCGS) a jejich modifikacemi. CDGS používají bezkontextová pravidla a přesto zvyšují sílu nad úroveň bezkontextových gramatik, leč zavedení distribuované spolupráce k zásobníkovým automatům sílu nezvyšuje. Dokázána je schopnost simulovat distribuovanou spolupráci pouze za použití stavu. Práce z tohoto výsledku dále vychází a zavádí variantu CDPDAS lišící se od všech variant CDGS, která zvyšuje sílu na roveň Turingových strojů (TM). PCGS mají sílu podobnou s CDGS, ale jimi inspirované PCPDAS jsou ekvivalení s TM, což je dokázáno umožněním přístupu k druhému zásobníku pomocí neintuitivního komunikačního protokolu.

Keywords

formal languages, regulated automata, cooperating distributed pushdown automata systems, parallel communicating pushdown automata systems

Klíčová slova

formální jazyky, regulované automaty, spolupracující distribuované systémy zásobníkových automatů, paralelní komunikující systémy zásobníkových automatů

Reference

KRČMÁŘ, Radim. *Regulated Automata Systems.* Brno, 2016. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Meduna Alexander.

Regulated Automata Systems

Declaration

I declare that this thesis has been composed solely by myself under supervision of Alexander Meduna and that I have referenced all external sources that were used.

.....
Radim Krčmář
May 25, 2016

Contents

Chapter 1

Introduction

A formal language is a set of sequences of symbols, so called strings. Defining an infinite set by listing its elements is impossible and even finite languages easily exceed our capacities, so succinct finite models are used to represent formal languages. Grammars and automata are the two best studied models. Grammars are generative models that repeatedly rewrite a starting symbol into a sequence that belongs to the formal language. Automata are consuming models that decide membership of an input sequence in the formal language.

An application that drove the initial development of grammars and automata is parsing. Automata have an input string and are a therefore better fitting model for parsing, but grammars are more used and studied because they are arguably more intuitive. Applications worked around the flaw of grammars by creating a hybrid that excludes derivations that would certainly lead to a string that differs from the parsed one, or by defining the language using a grammar and converting it to an automaton for parsing.

Grammars sufficiently powerful to express any formal language are rarely used as their computational complexity is not guaranteed to fit into practical range. Context-free grammars strike good balance between computational complexity and coverage of useful formal languages, yet more powerful grammars are in demand. Context-sensitive grammars are already considered as impractical for being too powerful, so the most active field of research lies in between. A successful method of getting a language in the range is to use context-free rules and have a mechanism on top of them.

Regulated context-free grammars cover is a wide spectrum of mechanisms that restrict the set of applicable context-free rules in each derivation. This thesis studies two concepts that were applied to context-free grammars and converts them for pushdown automata as pushdown automata model the same same set of languages as context-free grammars.

The first concept of interest comes from Cooperating Distributed Grammar Systems, where multiple context-free grammars take turns to derive a shared sentential form and the grammar system defines how many derivations in a row can each grammar perform. Cooperating Distributed Grammar Systems are stronger than context-free grammars.

Cooperating Distributed Grammar Systems are similar to another regulated grammar, the regular-controlled grammar. Regular-controlled grammar controls order of context-free rules that are used in derivation with a regular language. Regular-controlled grammar is also more powerful than context-free grammar, but the application of regular control language

to pushdown automata yields the same power as pushdown automata. The power of Cooperating Distributed Grammar Systems is incomparable with Regular-controlled grammars and this thesis observes whether application of the concept from Cooperating Distributed Grammar Systems to pushdown automata does increase power.

The second concept of interest comes from Parallel Communicating Grammar Systems, where multiple context-free grammars generate separate sentential forms in a step synchronized manner and a grammar may include the sentential form of any other grammars into its own sentential form between steps. Parallel Communicating Grammar Systems are more powerful than context-free grammars and no similar modification of pushdown automata has been tried before this thesis.

1.1 Preliminaries

This chapter gives an overview of definitions and formalisms from [6, 7, 8] that are used in following chapters. This thesis assumes that the reader has basic knowledge of algebra and proof techniques.

1.1.1 Algebraic structures

Sets

A *set* is a collection of objects with one operation, *membership*. $x \in S$ denotes that object x belongs to a set S. $y \notin S \iff \neg(y \in S)$. A set is defined by all its members. A set without any member is $\emptyset$, $x \notin \emptyset$ for all x.

For example, a set A of integers from 1 to 5 can be defined as $1 \in A, 2 \in A, 3 \in A, 4 \in A, 5 \in A$. Enumeration soon becomes unreadable, so the same set can be defined as $A = \{1, 2, 3, 4, 5\}$. An ellipsis is used if the author deems a series as obvious, $A = \{1, \ldots, 5\}$. Another method that defines a set is using an expression that has to be satisfied, $S = \{x \mid \lambda(x)\}$, which is equivalent to $\forall x \in I, \lambda(x) \iff x \in S$, where $\lambda : I \to \mathfrak{B}$

Common derived infix operations (with two sets) are $\subseteq, =, \subset, \cup, \cap, \setminus$.

$$
\begin{aligned}
X \subseteq Y &\iff a \in X \to a \in Y \\
X = Y &\iff X \subseteq Y \wedge Y \subseteq X \\
X \subset Y &\iff X \subseteq Y \wedge X \neq Y \\
X \cup Y &\iff a \in X \vee a \in Y \\
X \cap Y &\iff a \in X \wedge a \in Y \\
X \setminus Y &\iff a \in X \wedge a \notin Y
\end{aligned}
$$

Set X is a *subset* of set Y, $X \subseteq Y$ if every element of X is also in Y. If $X \subseteq Y$ and $Y \subseteq X$, those sets are considered as *equal* or *identical* ($=$). A *proper subset* ($\subset$) is similar to subset, but doesn't allow sets to be identical. *Union* ($\cup$) results in a set that has elements that are in either set. *Intersection* ($\cap$) has elements that are in both sets. *Difference* ($\setminus$) has elements from the left set that aren't in the right set.

Two sets are said to be *disjoint* if $X \cap Y = \emptyset$. A set X is a *superset* of Y if Y is a subset of X.

Tuples and Sequences

A *pair*, (x, y), is an ordering of two objects from sets X and Y.

$$(x, y) \in X \times Y \Rightarrow x \in X \wedge y \in Y$$

$X \times Y$ is called a *Cartesian product*. A pair is a special case of *tuple* with two elements. An n-tuple has n elements from n sets,

$$(e_1, \ldots, e_n) \in e_1 \times \cdots \times e_n \Rightarrow e_1 \in S_1 \wedge \ldots \wedge e_n \in S_n$$

Parentheses and commas can be omitted when the result is not ambiguous, 1-tuple is a simple example and a more complicated omission looks like

$$x \in \mathbb{N} \times \{a, \ldots, z\} \times \mathbb{N}, x = (2, k, 16) = 2, k, 16 = 2k16$$

An n-tuple where all objects are from the same set is called a *sequence* of length n and we usually omit parentheses and commas as there is no ambiguity. Sequence is convenient for tuples of variable length. The length of a sequence x is $|x|$, e.g.

$$x \in \{a, \ldots, z\}^*, x = (a, l, p, h, a) = alpha \Rightarrow |x| = 5$$

A sequence of length 0 is denoted by ε and called *empty*. $\{a, \ldots, z\}^*$ denotes the set of sequences of all lengths made of symbols in $\{a, \ldots, z\}$; X^+ denotes a set of sequences with elements from X of lengths greater than zero, $X^+ = X^* \setminus \{\varepsilon\}$.

Binary Relations and Functions

A *binary relation* is $R \subseteq X \times Y$, where X and Y are sets. The domain of R is X, co-domain is Y, and range is $\{y \mid (x, y) \in R\}$. Binary relation $R \subseteq X \times Y$ is a *function* if $X = \{x \mid (x, y) \in R\}$ and for all $(a, b), (x, y) \in R, a = x \Rightarrow b = y$.

Writing conventions

An arrow symbol ($\rightarrow$) is used to simplify tuples that act as relations. E.g. if $X \subset A \times B \times C$, then $ab \rightarrow c \in X$ is used instead of $(a, b, c) \in X$.

Unambiguous type constraints are omitted. E.g. $abd \Rightarrow cd, ab \rightarrow c \in X, d \in D$ stands for $abd \Rightarrow cd, ab \rightarrow c \in X, a \in A, b \in B, c \in C, d \in D$.

1.1.2 Languages

A *symbol* is any differentiable object. A sequence of symbols is a *word* or *string*. A set of strings is a *language*.

An *alphabet* of a language a is a superset of a set of all symbols in language's strings. The set of symbols in a string α is $alph(\alpha)$.

A *power* operator, a^n, where a is a string and n is an integer denotes n concatenations of string a. $a^n = aa^{n-1}$ and $a^0 = \varepsilon$, so $|a^n| = n|a|$

occur(x, l) is the number of occurrences of symbols from set x, or symbol x, in string l, e.g. occur$(a, abac) = 2$, occur$(\{a, b\}, abac) = 3$.

Language family is a set of languages that share a defined property. The property used in this thesis is a model of the language. Language family described by model M is $\mathscr{L}(M)$.

1.1.3 Grammars

Describing a language by the set of it words would be very inconvenient, and impossible for infinite languages, which is why grammars are introduced. Grammar is a quadruple

$$G = (N, T, R, S)$$

where

- N is a finite alphabet of nonterminals
- T is a finite alphabet of terminals, $N \cap T = \emptyset$
- $R \subset (N \cup T)^+ \times (N \cup T)^*$ is a finite set of rewriting rules
- $S \in N$ is the starting symbol

A language L described by grammar G is denoted as $L(G)$, we also say that grammar G generates language L.

$$L(G) = L((N, T, R, S)) = \{w \mid S \Rightarrow_R^+ w, w \in T^*\}$$

Where *direct derivation*, $\Rightarrow_R$, is an application of a rewrite rule from R.

$$\alpha x \beta \Rightarrow_R \alpha y \beta \text{ iff } (x \to y) \in R$$

where $\alpha, \beta \in (N \cup T)^*$. $\Rightarrow_G$ or $\Rightarrow$ is used for the same meaning when it presents no ambiguity. $\Rightarrow^+$ is a transitive closure, $\Rightarrow^*$ is a reflexive transitive closure, and $\Rightarrow^k$, for $k \geq 0$, is the kth power of of the binary relation $\Rightarrow$. $S_G \Rightarrow_G^* x$ is a *derivation* of $x \in (N_G \cup T_G)^*$. x is *sentential form* and if $x \in T_G$, x is also a *sentence*.

Chomsky hierarchy

The *Chomsky hierarchy* uses restrictions on grammar rewrite rules to classify languages.

- *Unrestricted grammars* (U) follow the definition above and are generate the family of *recursively enumerable languages*, $\mathscr{L}$(U) or **RE**.
- *Context-sensitive grammars* (CS) have all rules in the following form,

  $$\alpha x \beta \to \alpha y \beta \text{ where } x \in N \text{ and } y, \alpha, \beta \in (N \cup T)^*$$

  Context sensitive grammars generate the family of *context-sensitive languages*, $\mathscr{L}$(CS).
- Rules of *Context-free grammars* (CF) are a subset of

  $$N \times (N \cup T)^*$$

  Context-free grammars generate the family of *context-free languages*, $\mathscr{L}$(CF).
- Rules of *Regular grammars* (REG) are a subset of

  $$(N \times T^* N) \cup (N \times T^*)$$

  Regular grammars generate the family of *regular languages*, $\mathscr{L}$(REG)

Matrix grammars

Matrix grammar is an example of a regulated grammar. Regulation is achieved with a sequence (matrix) of rules that have to be applied in specified order.

A *matrix grammar with appearance checking* G is a triple

$$H = (G, M, W)$$

where

- $G = (N, T, P, S)$ is a context-free grammar
- $M \subseteq P^+$ is a finite set of matrices, which are sequences of derivation rules
- $W \subseteq P$ is the appearance checking set

The language generated by G is

$$L(G) = \{w \mid S \Rightarrow_H^* w, w \in T^*\}$$

where

$$\alpha \Rightarrow_H \beta \text{ iff } r_1 \cdots r_n \in M, \alpha \Rightarrow_{r_1,W} \cdots \Rightarrow_{r_n,W} \beta$$

$$xAz \Rightarrow_{A \to y,W} xyz$$

$$x \Rightarrow_{A \to y,W} x \text{ iff } A \notin alph(x) \wedge A \to y \in W$$

There is no derivation for rules that are not in the appearance checking set and cannot be applied and the matrix where this situation should arise will never be selected.

If $W = \emptyset$, then the grammar is called *matrix grammar*. The family of languages generated by matrix grammars, $\mathscr{L}$(M), is a strict superset of $\mathscr{L}$(CF), even though matrix grammars still use only context-free rewrite rules. $\mathscr{L}$(M) $\subset \mathscr{L}$(M with appearance checking).

ET0L grammars

Unlike previous grammars, every derivation of ET0L grammars rewrites all symbols in parallel.

For $k > 1$, an *ET0L grammar* G is a $k + 3$ tuple

$$G = (V, T, P_1, \ldots, P_k, s)$$

where

- V is a finite alphabet
- $T \subseteq V$ is a finite alphabet
- P_i is a finite subset of $V \times V^*$, for all $1 \leq i \leq k$
- $s \in V^+$ is the starting string

The language generated by G is

$$L(G) = \{w \mid s \Rightarrow_G^* w, w \in T^*\}$$

where

$$a_1 \cdots a_n \Rightarrow_G b_1 \cdots b_n \text{ iff } a_i \to b_i \in P_j \text{ for some } 1 \leq i \leq n, 1 \leq n \leq k$$

The family of languages generated by ET0L grammars, $\mathscr{L}$(ET0L), is strictly between $\mathscr{L}$(CF) and $\mathscr{L}$(CS).

1.1.4 Automata

Another common description of languages is through automata. An automaton is presented with a string in its working alphabet and decides whether it *accepts* the string. A set of all acceptable strings is the language of an automaton. Automatons have a storage of arbitrary symbols that is used for the decision and we differentiate automata based on the storage.

Finite state machine

An automaton that sequentially reads the string by symbols and has finite storage in a form of a state.

A *finite state machine* (FSM) M is a quintuple

$$M = (\Sigma, S, \delta, s, F)$$

where

- Σ is the finite input alphabet
- S is a set of states
- $\delta \subset S \times \Sigma \times S$ is a finite set of transition rules
- $s \in S$ is a starting state
- $F \subset S$ is a finite set of accepting states

The language accepted by M is

$$L(M) = L((\Sigma, S, \delta, s, F)) = \{w \mid w \in \Sigma^*, sw \Rightarrow_\delta^* f\varepsilon, f \in F\}$$

where *step*, $\Rightarrow_\delta$, is an application of a transition rule from δ.

$$si\alpha \Rightarrow_\delta u\alpha \text{ iff } si \to u \in \delta$$

where $\alpha \in \Sigma^*$. $\Rightarrow_M$ or $\Rightarrow$ is used for simplicity when the meaning is not ambiguous. sw and $f\varepsilon$ are *configurations*. $\Rightarrow^+$ is a transitive closure, $\Rightarrow^*$ is a reflexive transitive closure, and $\Rightarrow^k$, for $k \geq 0$, is the kth power of of the binary relation $\Rightarrow$. $s_M w \Rightarrow_M^* q_M x$

The family of languages accepted by FSM is the same family that regular grammars generate, $\mathscr{L}$(FSM) = $\mathscr{L}$(REG).

Pushdown automaton

Extend the finite state automaton with a stack. A *pushdown automaton* (PDA) M is a septuple

$$M = (\Sigma, S, \delta, s, F, \Gamma, p)$$

where

- Σ, S, s, and F are defined like in the finite automaton
- Γ is the stack alphabet
- $p \in \Gamma$ is the starting stack symbol
- δ is extended to a finite subset of $S \times \Gamma \times \{\Sigma \cup \{\varepsilon\}\} \times S \times \Gamma^*$

The language accepted by M is

$$L(M) = L((\Sigma, S, \delta, s, F, \Gamma, p)) = \{w \mid w \in \Sigma^*, (s, p, w) \Rightarrow_\delta^* (f, \gamma, \varepsilon), f \in F, \gamma \in \Gamma^*\}$$

where

$$(t, p\gamma, i\alpha) \Rightarrow_\delta (t', p'\gamma, \alpha) \text{ iff } tpi \to tp' \in \delta$$

and the rest of definitions is no different from the finite automaton.

The family of languages accepted by PDA is the same family that context-free grammars generate, $\mathscr{L}(\text{PDA}) = \mathscr{L}(\text{CF})$.

Non-standard pushdown symbol handling. This thesis uses a lot of rules that do not depend on the topmost stack symbol. To avoid many quantifications over all stack symbols like

$$\forall p \in \Gamma, spi \to typ$$

an ε symbol is used to signal that the topmost stack symbol is not changed and the rule is applicable for all possible symbols on the stack. The shortened version is

$$s\varepsilon i \to ty$$

and does not change semantics, so a rule with ε for stack symbol is not applicable when the stack is empty.

Turing Machine

Turing automaton, usually called Turing machine, extends the finite automaton by adding a tape. A *Turing machine* (TM) is a septuple

$$M = (\Sigma, S, \delta, s, F, \Gamma, b)$$

where

- Σ, S, s, and F a defined like in the finite automaton
- $\Sigma \subseteq \Gamma$ is the tape alphabet

- $b \in \Gamma$ is the blank tape symbol
- δ is changed to a finite subset of $S \times \Gamma \times S \times \Gamma \times \{L, R\}$

The language accepted by M is

$$L(M) = L((\Sigma, S, \delta, s, F, \Gamma, b)) = \{w \mid w \in \Sigma^*, (s, \epsilon, wb^\infty) \Rightarrow_\delta^* (f, \alpha, \beta), f \in F\}$$

where

$$(s, \alpha, x\beta) \Rightarrow_\delta (t, \alpha y, \beta) \text{ iff } sx \to tyR \in \delta$$

$$(s, \alpha n, x\beta) \Rightarrow_\delta (t, \alpha, nyb\beta) \text{ iff } sx \to tyL \in \delta$$

where $\alpha, \beta \in \Gamma^*$, $n \in \Gamma$.

The family of languages accepted by TM is the same family that unrestricted grammars generate, $\mathscr{L}(\text{TM}) = \mathscr{L}(\text{RE})$.

Linear bounded automaton

Follows the definition of a Turing automaton with one exception: size of the tape is a linear function of input word length. The language accepted by a *linear bounded automaton* (LBA) M is

$$L(M) = L((\Sigma, S, \delta, s, F, \Gamma, b)) = \{w \mid w \in \Sigma^*, (s, \epsilon, wb^{\lambda(length(w))}) \Rightarrow_\delta^* (f, \alpha, \beta), f \in F\}$$

where λ is a linear function.

The family of languages accepted by LBA is the same family that context-sensitive grammars generate, $\mathscr{L}(\text{LBA}) = \mathscr{L}(\text{CS})$.

Two-pushdown automaton

A two-pushdown automaton is an automaton with two stacks. A *two-pushdown automaton* (2-PDA) M is an octuple

$$M = (\Sigma, S, \delta, s, F, \Gamma, p_1, p_2)$$

The definition follows PDA, but initial symbol on the second stack is added and δ is a finite subset of $S \times \Gamma \times \Gamma \times \{\Sigma \cup \{\varepsilon\}\} \times S \times \Gamma^* \times \Gamma^*$

The language accepted by M

$$L(M) = \{w \mid w \in \Sigma^*, sp_1p_2w \Rightarrow_\delta^* f\gamma_1\gamma_2\varepsilon, f \in F, \gamma_1 \in \Gamma^*, \gamma_2 \in \Gamma^*\}$$

where

$$(t, p_1\gamma_1, p_2\gamma_2, i\alpha) \Rightarrow_\delta (t', p_1'\gamma_1, p_2'\gamma_2, \alpha) \text{ iff } (tp_1p_2i \to t'p_1'p_2') \in \delta$$

The family of languages accepted by 2-PDA is the same family that Turing Machines accept, $\mathscr{L}(\text{2-PDA}) = \mathscr{L}(\text{TM})$. *Basic Idea.* First stack simulates the tape under and left of the reading head and the second stack simulates tape right of the reading head.

Non-determinism

Non-determinism is crucial property of automata that guarantees acceptance if there exists a sequence of steps that leads to an accepting configuration, i.e. automaton M will always accept a word that belongs to $L(M)$. If there are two applicable rules, one that leads to an accepting configuration and one that does not, then transition with the latter rule is never taken. If there are multiple sequences of steps that lead to an accepting state, then non-determinism arbitrarily selects one. Every automaton in this thesis is non-deterministic.

Chapter 2

Cooperating distributed PDA systems

This chapter presents cooperating distributed grammar systems (CDGS) [11] and derives cooperating distributed PDA systems (CDPDAS) from their main concept. CDPDAS have power equal to PDA, but a related modification that raises power to the level of Turing Machines is introduced at the end.

2.1 Cooperating distributed grammar systems

A *cooperating distributed grammar system* G of degree k, $k \geq 1$ is a $k+4$ tuple

$$G = (N, T, S, D, P_1, \ldots, P_k)$$

where

- N, T, S are sets of non-terminals, terminals, and the starting symbol, respectively
- $D \in \{t\} \cup (\{\leq, =, \geq\} \times \mathbb{N})$ is the derivation mode
- P_i is a finite set of context-free rules of each component, $1 \leq i \leq k$

The language generated by G is

$$L(G) = \{w \mid S \Rightarrow_G^* w, w \in T^*\}$$

where

$$\alpha \Rightarrow_G \beta \text{ iff } \alpha \Rightarrow_{G_i}^D \beta$$

$\alpha \Rightarrow_{G_i}^D \beta$ is a derivation with context-free grammar of component i, $G_i = (N, T, S, P_i)$ and D is the derivation mode.

$$\alpha \Rightarrow_{G_i}^t \beta \text{ iff } \alpha \Rightarrow_{G_i}^* \beta, \text{alphabet}(\beta) \cap N_i = \emptyset, N_i = \{n \mid n \to x \in P_i\}$$
$$\alpha \Rightarrow_{G_i}^{=k} \beta \text{ iff } \alpha \Rightarrow_{G_i}^k \beta$$
$$\alpha \Rightarrow_{G_i}^{\leq k} \beta \text{ iff } \alpha \Rightarrow_{G_i}^j \beta, j \leq k$$
$$\alpha \Rightarrow_{G_i}^{\geq k} \beta \text{ iff } \alpha \Rightarrow_{G_i}^j \beta, j \geq k$$

Definition in [11] also has a $*$ derivation mode, but that derivation mode is equivalent to ≥ 0, so this thesis omits it.

2.1.1 Variants

Hybrid CDGS allow a separate derivation mode for every component and a hybrid CDGS G of degree k is has k derivation modes

$$G = (N, T, S, (P_1, D_1), \ldots, (P_k, D_k))$$

Derivation with component i uses D_i where non-hybrid would use D and are the same otherwise.

2.1.2 Power

Example 1. A language $T = \{a^n b^n c^n \mid n > 0\}$ is not context-free. A CD grammar system

$$G = (\{S, A, A', B, B'\}, \{a, b, c\}, S, = 2, P_1, P_2)$$

where

$$P_1 = \{S \to S, S \to AB, A' \to A, B' \to B\}$$
$$P_2 = A \to aA'b, B \to cB', A \to ab, B \to c\}$$

generates $L(G) = T$, therefore CDGS are stronger than context-free grammars.

Terminating derivation mode has the same power as context-free grammars for degree 1 and 2 and its power is equal to ET0L systems from degree 3 on. Derivation modes $\{=k, \geq k \mid k \leq 2\}$ are stronger than context-free from degree 2 and never surpass the power of matrix grammars. Other derivations modes have the same power as context-free grammars.

Hybrid CDGS of degree 4 and above are equivalent to matrix grammars with appearance checking, making them strictly stronger than non-hybrid CDGS [10].

2.2 PDA modifications

The main idea of CDGS is to have multiple separate grammars operating on the same string and a switch between them after a defined number of derivations.

A PDA has input, stack, state and transition rules. The CDPDAS will consist of multiple automata that operate on the same input and stack and switch after a defined number of derivations.

Definition 1. A *parallel communicating PDA system* M of degree k, $k \geq 1$, is a $k+4$ tuple

$$M = (\Sigma, \Gamma, Z, D, ((Q_1, \delta_1, s_1, F_1), \ldots, (Q_k, \delta_k, s_k, F_k))$$

where

- Σ, Γ, Z are defined as in PDA
- $D \in \{t\} \cup (\{\leq, =, \geq\} \times \mathbb{N})$ is the stepping mode as in CD grammar systems
- Q_i is a finite set of states of each component, $1 \leq i \leq k$
- $s_i \in Q_i$ is the starting state, $1 \leq i \leq k$
- $F_i \subseteq Q_i$ is a finite set of accepting states, $1 \leq i \leq k$
- $\delta_i \subset Q_i \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \times Q_i \times \Gamma^*$ is the finite transition relation, $1 \leq i \leq k$

The language accepted by M is

$$L(M) = \left\{ w \;\middle|\; (w, Z, (s_1, \ldots, s_k)) \Rightarrow_M^* (\varepsilon, \gamma, (t_1, \ldots, t_k)), \bigvee_{i=1}^{k} t_i \in F_i \right\}$$

where

$$\alpha \Rightarrow_M \beta \text{ iff } q_i p w \Rightarrow_{M_i}^* q_i' p' x'$$

$\Rightarrow M_i$ is a step with the PDA of component i, $M_i = (\Sigma, Q_i, \delta_i, s_i, F_i, \Gamma, Z)$, and D is the stepping mode.

$$(x, p, (q_1, \ldots, q_k)) \Rightarrow_{M_i}^{t} (x_1' x', p_1' p', (q_1, \ldots, q_i', \ldots, q_k)) \text{ iff } (q_i, p, w) \Rightarrow_{M_i}^* (q_i', p_1' p', x_1' x'),$$
$$\text{where } q_i' \in F_i \vee \forall (y, z) \in Q_i \times \Gamma^*, (q_i' x_1' p_1' \to yz) \notin \delta_i$$
$$(x, p, (q_1, \ldots, q_k)) \Rightarrow_{M_i}^{=k} (x', p', (q_1, \ldots, q_i', \ldots, q_k)) \text{ iff } q_i p w \Rightarrow_{M_i}^{k} q_i' p' x'$$
$$(x, p, (q_1, \ldots, q_k)) \Rightarrow_{M_i}^{\leq k} (x', p', (q_1, \ldots, q_i', \ldots, q_k)) \text{ iff } q_i p w \Rightarrow_{M_i}^{j} q_i' p' x', j \leq k$$
$$(x, p, (q_1, \ldots, q_k)) \Rightarrow_{M_i}^{\geq k} (x', p', (q_1, \ldots, q_i', \ldots, q_k)) \text{ iff } q_i p w \Rightarrow_{M_i}^{j} q_i' p' x', j \geq k$$

2.2.1 Power

CDPDAS has the same power as PDA, which will be proven by constructing CDPDAS from PDA and vice-versa.

Lemma 1. $L(PDA) \subseteq L(CDPDAS)$

Construction. For any PDA

$$M = (\Sigma, Q, \delta, s, F, \Gamma, Z)$$

construct CDPDAS

$$N = (\Sigma, \Gamma, Z, = 1, ((Q, \delta, s, F)))$$

Basic idea. CDPDAS of degree 1 with stepping mode = 1 trivially simulates PDA.

Proof.

$$L(M) = \{w \mid w \in \Sigma^*, sZw \Rightarrow_\delta^* f\gamma\varepsilon, f \in F, \gamma \in \Gamma^*\}$$

$$L(N) = \left\{ w \;\middle|\; (w, Z, (s)) \Rightarrow_N^* (\varepsilon, \gamma, (t_1)), \bigvee_{i=1}^{k} t_i \in F_i \right\}$$
$$= \{w \mid (w, Z, (s)) \Rightarrow_N^* (\varepsilon, \gamma, (f)), f \in F\}$$
$$L(M) = L(N) \text{ because } x \Rightarrow_\delta^{=1} y \text{ iff } x \Rightarrow_\delta y \qquad \square$$

Lemma 2. *$\mathscr{L}$(CDPDAS with terminating stepping mode) $\subseteq \mathscr{L}$(PDA)*

Construction. For any CDPDAS $M = (\Sigma, \Gamma, Z, t, ((Q_1, \delta_1, s_1, F_1), \ldots, (Q_k, \delta_k, s_k, F_k)))$, construct PDA $N = (\Sigma, Q, \delta, s, F, \Gamma, Z)$,

1. For all $1 \leq i \leq k$, do following steps
   (a) For all $h \in \Sigma \cup \{\varepsilon\}$, $q_i \in Q_i$, and $p \in \Gamma$, add $\langle \omega, q_i \rangle hp \to \langle h, q_i \rangle p$ into δ_i
   (b) Add all $\{\langle h, q_i \rangle \varepsilon p \to \langle \omega, q_i' \rangle \gamma \mid q_i hp \to q_i' \gamma \in \delta_i,\ h \neq \varepsilon\}$ into δ_i'
   (c) For all $h \in \Sigma \cup \{\varepsilon\}$, add all $\{\langle h, q_i \rangle \varepsilon p \to \langle h, q_i' \rangle \gamma \mid q_i \varepsilon p \to q_i' \gamma \in \delta_i\}$ into δ_i'
   (d) For all $(s, h, p, x, y) \in Q_i \times \Sigma \times \Gamma \times \Sigma \times \Gamma^*$, add $\langle h, s \rangle \varepsilon p \to \langle \langle h, s \rangle \rangle p$ into δ_i' if $shp \to xy \notin \delta_i$
   (e) For all $q_j \in Q_j$, $1 \leq j \leq k$, $\langle q_i, g \rangle hp \to \langle q_i', g' \rangle \gamma \in \delta_i'$, add $\langle i, g, q_1, \ldots, q_i, \ldots, q_k \rangle hp \to \langle i, g', q_1, \ldots, q_i', \ldots, q_k \rangle \gamma$ into δ
   (f) For all $q_j \in Q_j$, $1 \leq j \leq k$, $\langle q_i, g \rangle \varepsilon p \to \langle \langle q_i, g \rangle \rangle \gamma \in \delta_i'$, add $\langle i, g, q_1, \ldots, q_k \rangle \varepsilon p \to \langle j, g, q_1, \ldots, q_k \rangle p$ into δ and $\{\langle i, g, q_1, \ldots, q_i, \ldots, q_k \rangle, \langle i, g', q_1, \ldots, q_i', \ldots, q_k \rangle\}$ into Q
2. For all $h \in \Sigma \cup \{\varepsilon\}$ and $1 \leq i \leq k$, add $s_i hZ \to \langle i, \omega, s_1, \ldots, s_k \rangle Z$ into δ and add $\langle i, \omega, s_1, \ldots, s_k \rangle$ into Q
3. For all $q \in Q$, $q = \{i, h, q_1, \ldots, q_k\}$, add q into F if $\bigvee_{j=1}^{k} q_j \in F$

Basic idea. Introduce component index into state and modify all rules of original components to be applicable in states that have the appropriate index in state. Switching between components happens when the terminating condition is detected. To detect the terminating condition, we need to peek at the topmost stack symbol and the next input symbol. The topmost input symbol has to be remembered in the state, because an input symbol cannot be read twice. Rules that consume the input symbol only store the symbol in a state, the rest of rules consume the symbol from the state. ε-input step ambiguity is avoided because ε is the last symbol we ever read, so if it is read while some input is remaining, then the automaton doesn't accept the word.

Proof. Induction Basis. The first step for N is

$$sZw \Rightarrow_N \langle i, \omega, s_1, \ldots, s_k \rangle Zw$$

which contains the same information information as starting configuration of M,

$$(w, Z, (s_1, \ldots, s_k))$$

and also picks a component. If the word will be accepted, then non-determinism will pick from the same set of components as the first step of M would. If the word won't be accepted, then sets of possible components are also the same, because non-determinism can pick any component in both cases.

Induction step. A step of M with component i,

$$h\alpha, p\gamma, (q_1, \ldots, q_k) \Rightarrow_M \alpha, p'\gamma, (q_1, \ldots, q_i', \ldots, q_k)$$

where $h \neq \varepsilon$ implies a use of rule $q_i hp \to q_i' p' \in \delta_i$.

Configuration $(h\alpha, p\gamma, (q_1, \ldots, q_k))$ has 2 equivalent configurations in N,

1. $\langle i, \omega, (q_1, \ldots, q_k)\rangle, h\alpha, p\gamma$. The only step that N can perform is

   $$\langle i, \omega, (q_1, \ldots, q_k)\rangle, h\alpha, p\gamma \Rightarrow_N \langle i, h, (q_1, \ldots, q_k)\rangle, \alpha, p\gamma$$

   with a rule generated in 1a, which reaches the second equivalent configuration.

2. $\langle i, h, (q_1, \ldots, q_k)\rangle, \alpha, p\gamma$. Item 1b and 1e added rule

   $$\langle i, h, q_1, \ldots, q_i, \ldots, q_k\rangle \varepsilon p \to \langle i, \omega, q_1, \ldots, q_i', \ldots, q_k\rangle$$

   to δ if rule $q_i hp \to q_i' p'$ exists in δ_i. A step with the rule

   $$\langle i, h, (q_1, \ldots, q_k)\rangle, \alpha, p\gamma \Rightarrow_N \langle i, \omega, (q_1, \ldots, q_i', \ldots, q_k)\rangle, \alpha, p'\gamma$$

   reaches an equivalent configuration to M's $\alpha, p'\gamma, (q_1, \ldots, q_i', \ldots, q_k$.

A step of M with component i,

$$\alpha, p\gamma, (q_1, \ldots, q_k) \Rightarrow_M \alpha, p'\gamma, (q_1, \ldots, q_i', \ldots, q_k)$$

implies a use of rule $q_i \varepsilon p \to q_i' p' \in \delta_i$.

Configuration $(h\alpha, p\gamma, (q_1, \ldots, q_k))$ has $|\Gamma| + 1$ equivalent configurations in N. There are only two relevant classes, though

1. $\langle i, \omega, (q_1, \ldots, q_k)\rangle, h\alpha, p\gamma$, where $h \in \Sigma \cup \{\varepsilon\}$. The only permissible step

   $$\langle i, \omega, (q_1, \ldots, q_k)\rangle, h\alpha, p\gamma \Rightarrow_N \langle i, h, (q_1, \ldots, q_k)\rangle, \alpha, p\gamma$$

   will reach one configuration of the second class.

2. $\langle i, h, (q_1, \ldots, q_k)\rangle, \alpha, p\gamma$. Item 1c and 1e added rule

   $$\langle i, h, (q_1, \ldots, q_k)\rangle, \alpha, p\gamma \Rightarrow_N \langle i, h, (q_1, \ldots, q_i', \ldots, q_k)\rangle, \alpha, p'\gamma$$

   to δ if rule $q_i \varepsilon p \to q_i' p'$ exists in δ_i. A step with the rule

   $$\langle i, h, (q_1, \ldots, q_k)\rangle, \alpha, p\gamma \Rightarrow_N \langle i, \omega, (q_1, \ldots, q_i', \ldots, q_k)\rangle, \alpha, p'\gamma$$

   reaches an equivalent configuration.

If a next step with component i is not possible, then M non-deterministically switches to another component. A step with component i from configuration $h\alpha, p\gamma, (q_1, \ldots, q_k))$, where $h \in \Sigma \cup \{\varepsilon\}$, is not possible if and only if a rule of the form $q_i hp \to q_i' p'$ is not in δ_i.

A configuration of M where no steps with component i are possible has two equivalent configurations in N,

1. $\langle i, \omega, (q_1, \ldots, q_k)\rangle, h\alpha, p\gamma$. Like before, the only possible step reaches the second equivalent configuration by consuming and remembering the input

   $$\langle i, \omega, (q_1, \ldots, q_k)\rangle, h\alpha, p\gamma \Rightarrow_N \langle i, h, (q_1, \ldots, q_k)\rangle, \alpha, p\gamma$$

2. $\langle i, h, (q_1, \ldots, q_k)\rangle, \alpha, p\gamma$. There is no applicable rule in δ_i, so items 1b and 1e do not generate any rules in δ, but items 1d and 1f do generate k applicable rules of the form

$$\langle i, h, (q_1, \ldots, q_k)\rangle \varepsilon p \to \langle j, h, (q_1, \ldots, q_k)\rangle p$$

where $1 \leq j \leq k$. Non-determinism ensures that same sets of component can be switched to.

When N reaches a configuration with ε in the state,

$$\langle i, \varepsilon, (q_1, \ldots, q_k)\rangle \alpha\gamma$$

then there are no rules that would consume the input or put other symbol in place of ε again, so α has to be ε for the input word to be accepted.

All possible transitions of M can be simulated by N, therefore $L(M) \subseteq L(N)$. There are no rules in N that could step into a configuration whose equivalent cannot be reached with rules in M, therefore $L(N) \subseteq L(M)$ and $L(N) = L(M)$. □

Lemma 3. *$\mathscr{L}$(CDPDAS with stepping mode in $\{\leq, =, \geq\} \times \mathbb{N}$) $\subseteq \mathscr{L}$(PDA)*

Construction. For $CDPDAS$ $M = (\Sigma, \Gamma, Z, D, ((Q_1, \delta_1, s_1, F_1), \ldots, (Q_k, \delta_k, s_k, F_k)))$, construct PDA $N = (\Sigma, Q, \delta, s, F, \Gamma, Z)$, by performing following steps for all $1 \leq i \leq k$,

1. $Q'_i = \{0, \ldots, n+1\} \times Q_i$
2. For all $0 \leq j < n$, where $(r, n) = D$, add $\{\langle j, q\rangle ip \to \langle j+1, q'\rangle\gamma \mid qip \to q'\gamma \in \delta_i\}$ into δ'_i.
3. If $(\geq, n) = D$, then add $\{\langle n, q\rangle ip \to \langle n, q'\rangle\gamma \mid qip \to q'\gamma \in \delta_i\}$ into δ'_i.
4. For all $\langle n, q_i\rangle ip \to n'q'_i\gamma \in \delta'_i$ and all $q_j \in Q_j$, $1 \leq j \leq k$, add $\langle i, n, q_1, \ldots, q_k\rangle \to \langle i, n', q_1, \ldots, q'_i, \ldots, q'_k\rangle$ into δ.
5. For all $1 \leq j \leq k$, all $q_m \in Q_m$, $1 \leq m \leq k$, and all $0 \leq n \leq l$, $(r, l) = D$, add $\langle i, n, q_1, \ldots, q_k\rangle\varepsilon\varepsilon \to \langle j, 0, q_1, \ldots, q_k\rangle\varepsilon$ into δ if the stepping mode D holds true for n. The value of D for n is $r(n, l)$ and will be written as predicate $D(n)$.
6. Add $s\varepsilon\varepsilon \to \langle i, 0, s_1, \ldots, s_k\rangle\varepsilon$ into δ.
7. For all $q_j \in Q_j$, $1 \leq j \leq k$, and all $0 \leq n \leq l$, $(r, l) = D$, add $\langle i, n, q_1, \ldots, q_k\rangle$ into F if any q_j is in F_j.

Basic idea. Remember the index of the simulated component and the relevant number of steps in the state. When the number of steps satisfies the stepping mode predicate, then it is possible to change the component and restart the step counter.

When M non-deterministically selects a component i, then it performs n steps with component i before selecting new component, where $D(n)$ is true. N keeps an explicit count of n, with respect to satisfiability of $D(n)$. The maximal useful value of n for stepping modes $\geq l$, $= l$, and $\leq l$ is $n = l$, because the value of $D(n)$ won't change for $n > l$. A finite storage for the number of steps is enough and N stops counting steps after step l, which makes the conversion feasible.

Proof. $L(M) \subseteq L(N)$ *Induction Basis.* M begins in configuration $(w, Z, (s_1, \ldots, s_k))$ and implicitly selects component i to perform the first derivation. The first step that N does is

$sZw \Rightarrow \langle i, 0, s_1, \ldots, s_k\rangle Zw$, where $1 \leq i \leq k$ is non-deterministically picked to match the selection of component that M does.

M accepts the word in configuration $(\varepsilon, \gamma, (q_1, \ldots, q_k))$, where any $q_i \in F_i$, $1 \leq i \leq k$. The equivalent configuration of N is $(\langle j, n, q_1, \ldots, q_k\rangle, \gamma\varepsilon)$ and construction step 7 adds $\langle j, n, q_1, \ldots, q_k\rangle$ info F iff any $q_i \in F_i$, $1 \leq i \leq k$.

Induction step. An n-th step with component i of M,

$$a\alpha, p\gamma, (q_1, \ldots, q_k) \Rightarrow_{M_i} \alpha, p'\gamma, (q_1, \ldots, q'_i, \ldots, q_k)$$

is done with a rule $q_i ap \to q'_i p'$ in δ_i. The equivalent step with automaton N is

$$\langle i, n-1, q_1, \ldots, q_k\rangle p\gamma a\alpha \Rightarrow_N \langle i, n, q_1, \ldots, q'_i, \ldots, q_k\rangle p'\gamma\alpha$$

and $q_i ap \to q'_i p' \in \delta_i$ implies $\langle i, n-1, q_1, \ldots, q_k\rangle ap \to \langle i, n, q_1, \ldots, q'_i, \ldots, q_k\rangle p'$, because of construction steps 2, 3, and 4.

After performing n steps in component i, and if $D(n)$ is true, M can do a 1st step with any other automaton j, $1 \leq j \leq k$

$$a\alpha, p\gamma, (q_1, \ldots, q_k) \Rightarrow_{M_j} \alpha, p'\gamma, (q_1, \ldots, q'_j, \ldots, q_k)$$

with a rule $q_j ap \to q'_j p'$ in δ_j. N simulates that with two steps that are made by construction rule 5 and rules 2, 3, and 4, respectively,

$$\langle i, n, q_1, \ldots, q_k\rangle, p\gamma, a\alpha \Rightarrow_N \langle j, 0, q_1, \ldots, q_k\rangle, p\gamma, a\alpha \Rightarrow_N \langle j, 1, q_1, \ldots, q'_j, \ldots, q_k\rangle, p'\gamma, \alpha$$

N accepts all words that M does.

$L(N) \subseteq L(M)$ *Induction basis.* First steps of N is $sZw \Rightarrow \langle i, 0, s_1, \ldots, s_k\rangle Zw$, which non-deterministically selects a new component. M starts with equivalent configuration $(w, Z, (s_1, \ldots, s_k))$ and select component i implicitly on the first step.

N accepts the word in configuration $(\langle j, n, q_1, \ldots, q_k\rangle, \gamma\varepsilon)$ if $\langle j, n, q_1, \ldots, q_k\rangle \in F$. The equivalent configuration of M is $(\varepsilon, \gamma, (q_1, \ldots, q_k))$, and construction step 7 adds $\langle j, n, q_1, \ldots, q_k\rangle$ info F iff any $q_i \in F_i$, $1 \leq i \leq k$.

Induction step. A step of automaton N

$$\langle i, n, q_1, \ldots, q_k\rangle, p\gamma, a\alpha \Rightarrow_N \langle i, n+1, q_1, \ldots, q'_i, \ldots, q_k\rangle, p'\gamma, \alpha$$

can be performed only if component i of M has a rule

$$q_i pa \to q'_i p'$$

Simulation with M did n steps in component i at that point and a rule to perform the following derivation exists, because the rule in N can only be contructed by steps 2, 3, 4, and 5 from the rule of component i. M steps into an equivalent configuration as N by performing

$$p\gamma, a\alpha, (q_1, \ldots, q_k) \Rightarrow_M p'\gamma, \alpha, (q_1, \ldots, q'_i, \ldots, q_k)$$

The same holds for

$$\langle i, n, q_1, \ldots, q_k \rangle, p\gamma, a\alpha \Rightarrow_N \langle i, n, q_1, \ldots, q'_i, \ldots, q_k \rangle, p'\gamma, \alpha$$

and

$$p\gamma, a\alpha, (q_1, \ldots, q_k) \Rightarrow_{M_i} p'\gamma, \alpha, (q_1, \ldots, q'_i, \ldots, q_k)$$

which happens after n steps of the same component in $\geq n$ derivation modes.

A step of automaton N

$$\langle i, n, q_1, \ldots, q_k \rangle, p\gamma, a\alpha \Rightarrow_N \langle j, 0, q_1, \ldots, q_k \rangle, p\gamma, a\alpha \Rightarrow_N \langle j, 1, q_1, \ldots, q'_j, \ldots, q_k \rangle, p'\gamma, \alpha$$

can only happen if n satisfies the derivation mode and component j of M has a rule

$$q_j pa \to q'_j p'$$

M did n steps in component i at that point and M can switch a component, because n satisfies the derivation mode. M reaches an equivalent configuration by performing

$$p\gamma, a\alpha, (q_1, \ldots, q_k) \Rightarrow_{M_j} p'\gamma, \alpha, (q_1, \ldots, q'_j, \ldots, q_k)$$

M accepts all words that N does. □

Theorem 1. $\mathscr{L}(CDPDAS) = \mathscr{L}(PDA)$

Proof. Follows from Lemma 1, Lemma 2, and Lemma 3. □

2.2.2 Variants

The definition of CDPDAS returned to the last state after a switch. A simple alternative is to always start with the starting symbol and a complex alternative is to define a switch-next-state relation.

The complex variant doesn't increase power. A proof would be almost identical to Lemma 2 and Lemma 3, so only the basic idea of a difference is given.

The variant includes a switch-next-state relation $\varrho \subseteq Q \times 2^Q$ into the relation returns a set of permissible states after a component switch for each state. The construction of PDA would add

$$\{q \to q' \mid q = \langle i, n, q_1, \ldots, q_k \rangle \in Q, D(n), q' \in \varrho(q)\}$$

into δ. The proven variant and the simple alternative are a subset of the complex one.

Hybrid CDPDAS M has a derivation mode for each compoent,

$$M = (\Sigma, \Gamma, Z, ((Q_1, \delta_1, s_1, F_1, D_1), \ldots, (Q_k, \delta_k, s_k, F_k, D_k))$$

The power of Hybrid CDPDAS remains at the level of PDA. *Basic idea.* Constructions in Lemma 2 and Lemma 3 have a set of steps for each component i, $1 \leq i \leq k$, that used derivation mode D. Replacing D with derivation mode D_i yields a proof for Hybrid CDPDAS.

2.3 Increasing power

The reason why CDPDAS did not increase power is because the modification added a finite amount of information and state can already express any finite amount of information. An unbounded storage is needed to increase power.

Consider a returning terminating derivation mode that has different accepting mechanism: If a component i terminates in a non-accepting state, then another component j is selected and begins stepping from the initial state s_j, the difference from returning CDPDAS is that the component i and its current state is stored for added to a sequence of stored component-state pairs. When a component j reaches an accepting state, then the last stored component is resumed from the stored state. The automaton halts and possibly accepts if there is no previously stored component.

Definition 2. CDPDAS+ M uses the same $k+3$ tuple as CDPDAS,

$$M = (\Sigma, \Gamma, Z, D, ((Q_1, \delta_1, s_1, F_1), \ldots, (Q_k, \delta_k, s_k, F_k))$$

but the language accepted by M is

$$L(M) = \{w \mid (w, Z, \varepsilon, \bot) \Rightarrow_M^* (\varepsilon, \gamma, \varepsilon, \top)\}$$

where

$$\begin{aligned}
(w, p, \Phi, \bot) \Rightarrow_M (w', p', \Phi', r) \text{ iff } & (w, p, (s_1, \ldots, s_k)) \Rightarrow_{M_i}^t (w', p', (s_1, \ldots, t_i', \ldots, s_k)) \\
(w, p, it_i\Phi, \top) \Rightarrow_M (w', p', \Phi', r) \text{ iff } & (w, p, (s_1, \ldots, t_i, \ldots, s_k)) \Rightarrow_{M_i}^t (w', p', (s_1, \ldots, t_i', \ldots, s_k)) \\
r = \top, \Phi' = \Phi \text{ iff } & t_i' \in F_i \\
r = \bot, \Phi' = it_i'\Phi \text{ iff } & t_i' \notin F_i
\end{aligned}$$

where $\Rightarrow_{M_i}^t$ is the terminating derivation transition from Definition 1.

Theorem 2. $\mathscr{L}(\textit{CDPDAS+}) = \mathscr{L}(\textit{2-PDA})$

This thesis focuses on modifications that are directly inspired from grammar systems and CDPDAS+ is not, so only a sketch proof is given.

Basic Idea. Any symbol that would be stored on the second stack is instead encoded inside a state that is stored in Φ. CDPDAS+ simulating 2-PDA has 1 component whose state contains state of 2-PDA and the topmost symbol of the second stack (starts with initial symbol). CDPDAS+ has unique stack symbols for each original stack symbol x, $\langle \Gamma, x \rangle$, and one unique symbol for each state t, $\langle \Sigma, t \rangle$.

For rule of 2-PDA that pushes something onto the stack,

$$tip_1p_2 \to (t', p_1', p_2')$$

so $p_2' \neq \varepsilon$, CDPDAS+ has a rule

$$\langle t, p_2 \rangle ip_1 \to \langle P, p_{2_i} \rangle \langle \Gamma, p_{2_{i-1}}' \rangle \cdots \langle \Gamma, p_{2_1}' \rangle \langle \Sigma, t' \rangle p_1'$$

where $p_2' = p_{2_1} \cdots p_{2_i}$. p_{2_1} is the topmost symbol on the second 2-PDA stack after this 2-PDA rule. The special state P does not have any rule for $\langle \Gamma, x \rangle$, so if 2-PDA pushed

more on the second 2-PDA stack, our component will terminate in state $\langle P, p_{2_i} \rangle$, which is stored in Φ and starts from state $\langle S, Z_2 \rangle$. A rule

$$\langle S, Z_2 \rangle \varepsilon \langle \Gamma, x \rangle \to \langle P, x \rangle$$

stores the next symbol inside state and storage into Φ is repeated until $\langle \Sigma, t' \rangle$ is on top of the stack. $\langle P, p'_{2_2} \rangle \cdots \langle P, p'_{2_i} \rangle$ is prepended to Φ at that point. A rule

$$\langle P, p_2 \rangle \varepsilon \langle \Sigma, t' \rangle \to \langle t', p_2 \rangle$$

then enters simulation of the next 2-PDA rule.

If the 2-PDA does not push anything on stack,

$$t i p_1 p_2 \to (t', p'_1, \varepsilon)$$

CDPDAS+ has a rule

$$\langle t, p_2 \rangle i p_1 \to \langle Q, Z_2 \rangle \langle \Sigma, t' \rangle p_1$$

$Q \in F$, so CDPDAS+ returns to the state that was last stored in Φ. That state contains the last symbol on the second 2-PDA stack.

To accept correct words, initialization must begin by replacing the initial stack symbol Ω using a rule

$$\langle S, Z_2 \rangle \varepsilon \Omega \to \langle X, Z_2 \rangle \varepsilon Z_1$$

State X is not final and only has transitions when $\langle \Sigma, t' \rangle$ is on the stack, so $\langle X, Z_2 \rangle$ is the first, hence bottom-most, in Φ. If the machine ever returns to $\langle X, Z_2 \rangle$, it will be with $\langle \Sigma, t' \rangle$ on if t' was an accepting state of 2-PDA, CDPDAS+ will enter an accepting state. CDPDAS+ then halts.

There are few other corner cases, but the basic idea is hopefully clear.

Chapter 3

Parallel communicating PDA systems

This chapter presents parallel communicating grammar systems (PCGS) [12] and derives parallel communicating PDA systems (PCPDAS) from their main concept. Most PCPDAS variations have power equal to Turing machines and two variations denote infinite hierarchies of languages above context-free languages.

3.1 Parallel communicating grammar systems

PCGS connect n grammars into a system where components do every derivation step in parallel and can query the working string of other components in between.

A *parallel communicating grammar system* G of degree n is a $3+n$ tuple

$$G = (N, K, T, (S_1, P_1), \ldots, (S_n, P_n))$$

where

- N and T are disjunct sets of non-terminals and terminals
- $K = \{Q_1, Q_2, ..., Q_n\}$ is a set of query symbols, $K \cap (N \cup T) = \emptyset$
- $S_i \in N$ are starting symbols
- P_i is a set of rules of type $N \to (N \cup T \cup K)^*$

The language generated by G is

$$L(G) = \{w \mid S_1, \ldots, S_n \Rightarrow_G^+ w, \alpha_2 \ldots, \alpha_n, w \in T^*, \alpha_i \in (N \cup T)^*\}$$

where

$$\alpha_i \Rightarrow_{P_i} \beta_i$$

$$\gamma_i = \text{query}(i, \beta_1, \ldots, \beta_n)$$

$$\delta_i = \begin{cases} \gamma_i & \text{if for all } 1 \leq j \leq k, Q_i \notin alpha(\beta_j) \\ S_i & \text{otherwise} \end{cases}$$

$$\alpha_1, \ldots, \alpha_n \Rightarrow_G \gamma_1, \ldots, \gamma_n$$

$\Rightarrow_{P_i}$ is a derivation with context-free grammar $(N \cup K, T, P_i, S_i)$ and query$(i, \beta_1, \ldots, \beta_n)$ is a recursive substitution of all query symbols in $i+1$th.

$$\begin{aligned}
&\text{query}(i, \beta_1, \ldots, \beta_n) = \\
&\quad \text{while alph}(\beta_i) \cap K \neq \emptyset : \\
&\quad\quad \beta_i \leftarrow \text{replace}(\beta_i, (Q_1, \ldots, Q_n), (\beta_1, \ldots, \beta_n)) \\
&\quad \text{return } \beta_i
\end{aligned}$$

This definition of $\Rightarrow_G$ slightly differs from [12] in order to make clearer transition into automata. Both definitions are equivalent, because

- Γ always prefers c-steps to g-steps
- c-steps is performed if there is a query symbol in any sentential form
- c-steps do not terminate if there is a loop of query symbols
- c-steps only reset a component that is not queried in sequential c-steps

3.1.1 Variants

Centralised PCGS

Only the first component can produce query symbols in a Centralised PCGS. A PCGS

$$G = (N, K, T, (S_1, P_1), \ldots, (S_n, P_n))$$

is *centralised PCGS* iff for all $A \to x \in P_i$, where $2 \leq i \leq n$,

$$alph(x) \cap K = \emptyset$$

Non-Returning PCGS

Components that were queried don't return to their starting sentential form. The definition of *non-returning PCGS* G is identical to PCGS,

$$G = (N, K, T, (S_1, P_1), \ldots, (S_n, P_n))$$

Only the language generated by a non-returning PCGS is different,

$$L(G) = \{w \mid S_1, \ldots, S_n \Rightarrow_G^+ w, \alpha_2 \ldots, \alpha_n, w \in T^*, \alpha_i \in (N \cup T)^*\}$$

where

$$\begin{aligned}
\alpha_i &\Rightarrow_{P_i} \beta_i \\
\gamma_i &= \text{query}(i, \beta_1, \ldots, \beta_n) \\
\alpha_1, \ldots, \alpha_n &\Rightarrow_G \delta_1, \ldots, \delta_n
\end{aligned}$$

Prefix PC Grammar System

When a component i generates a query for component j, then component j may communicate any non-empty prefix of its sentential string. If the whole sentential form was communicated, then the sentential form returns to the initial symbol, otherwise the component j continues to work on the suffix.

3.1.2 Power

Known relations are [9].

$$\mathscr{L}(\text{CF}) \subset \mathscr{L}(\text{Centralised PCGS}) \subset \mathscr{L}(\text{PCGS}) \subset \mathbf{RE}, \mathscr{L}(\text{M}) \subset \mathscr{L}(\text{PCGS})$$

$$\mathscr{L}(\text{CF}) \subset \mathscr{L}(\text{Non-returning centralised PCGS}) \subset \mathscr{L}(\text{Non-returning PCGS}) \subset \mathscr{L}(\text{PCGS})$$

$$\mathscr{L}(\text{Prefix PCGS}) = \mathbf{RE}$$

Example 2. PCGS G generates a non-context free language $L(G) = \{a^n b^n c^n \mid n > 0\}$.

$$G = (\{A, B, C\}, \{Q_1, Q_2, Q_3\}, \{a, b, c\}, (A, P_1), (B, P_2), (C, P_3)$$

where

$$\begin{aligned} P_1 &= \{A \to aA, A \to aQ_2Q_3, B \to \varepsilon, C \to \varepsilon\} \\ P_2 &= \{B \to bB\} \\ P_3 &= \{C \to cC\} \end{aligned}$$

3.2 PDA modifications

The main idea of PCGS is an ability to include working state of other components. PCPDAS adapt this idea for stack, so every component of the system can obtain a copy of the whole stack of another component.

Definition 3. A *parallel communicating PDA system* M of degree n is $n + 3$ tuple

$$M = (\Sigma, \Gamma, K, (S_1, \delta_1, s_1, F_1, p_1), (S_2, \delta_2, s_2, F_2, p_2), \ldots, (S_n, \delta_n, s_n, F_n, p_n))$$

where

- Σ is the input alphabet
- Γ is the stack alphabet
- $K = \{Q_1, Q_2, \ldots, Q_n\}, K \cap \Gamma = \emptyset$ is a finite set of query symbols
- S_i is a finite set of states, for $0 < i \leq n$
- $s_i \in S_i$ is a starting state, for $0 < i \leq n$
- $F_i \subseteq S_i$ is a finite set of accepting states, for $0 < i \leq n$
- $p_i \in \Gamma$ is an initial stack symbol, for $0 < i \leq n$

- $delta_i$ is a set of transition rules, $S_i\Gamma(\Sigma \cup \{\varepsilon\} \to S_i(\Gamma \cup K)^*$, for $0 < i \leq n$

The language of M is

$$
\begin{aligned}
&L(M) = \\
&\quad L((\Sigma, \Gamma, K, (S_1, \delta_1, s_1, F_1, p_1), (S_2, \delta_2, s_2, F_2, p_2), \ldots, (S_n, \delta_n, s_n, F_n, p_n))) = \\
&\quad \{w \mid w \in \Sigma^*, (s_1 p_1 w, \ldots, s_n p_n w) \Rightarrow_M (f_1\gamma_1\epsilon, \ldots, f_n\gamma_n\epsilon), \exists f_i \in F_i, \gamma_i \in \Gamma\}
\end{aligned}
$$

where

$$
\begin{aligned}
&(s_1 p_1 \gamma_1 i_1 \alpha_1, \ldots, s_n p_n \gamma_n i_n \alpha_n) \Rightarrow_M (t_1 q_1 \gamma_1 \alpha_1, \ldots, t_n q_n \gamma_n \alpha_n) \\
&\quad \text{iff } (s_1 p_1 i_1 \to t_1 \kappa_i) \in \delta_1, \ldots, (s_n p_n i_n \to t_n \kappa_n) \in \delta_n \\
&\quad\quad \text{and terminates}(\kappa_1\gamma_1, \ldots, \kappa_n\gamma_n) \\
&\quad \text{where } q_i = \text{query}(i, \kappa_1\gamma_1, \ldots, \kappa_n\gamma_n)
\end{aligned}
$$

query function replaces query symbols and doesn't terminate if there is a loop, so terminates is used to avoid an undecidable non-terminating transition

```
terminates(α_1, ..., α_n) =
  for i in 1, ..., n :
    seen ← ∅
    while alph(α_i) ∩ K ≠ ∅ :
      if alph(α_i) ∩ seen ≠ ∅ :
        return false
      seen ← seen ∪ alph(α_i)
      α_i ← replace(α_i, (Q_1, ..., Q_n), (α_1, ..., α_n))
  return true
```

Example 3. The automaton M accepts $a^n b^n c^n$, $n \geq 0$.

$$M = (\{a, b, c\}, \{a, b, c, \#\}, \{Q_1, Q_2\}, (\{a_1, b_1, c_1, f\}, a_1, \delta_1, \{f\}, \#), (\{a_2, x_2\}, a_2, \delta_2, \emptyset, \#))$$

where

$$
\begin{aligned}
\delta_1 = \{&a_1\varepsilon\# \to f, a_1 a\# \to a_1 a\#, a_1 aa \to a_1 aa, a_1 ba \to b_1, b_1 ba \to b_1, b_1\varepsilon\# \to c_1 Q_2, \\
&c_1 ca \to c_1, c_1\varepsilon\# \to f\} \\
\delta_2 = \{&a_2 a\# \to a_2 a\#, a_2 aa \to a_2 aa, a_2 ba \to x_2 a, x_2 ba \to x_2 a, x_2 ca \to x_2 a\}
\end{aligned}
$$

Input word *aaaabbbbcccc* is accepted with following steps,

$$
\begin{aligned}
&(a_1\#aaaabbbbcccc, a_2\#aaaabbbbcccc) \Rightarrow_M (a_1 a\#aaabbbbcccc, a_2 a\#aaabbbbcccc) \Rightarrow_M \\
&(a_1 aa\#aabbbbcccc, a_2 aa\#aabbbbcccc) \Rightarrow_M (a_1 aaa\#abbbbcccc, a_2 aaa\#abbbbcccc) \Rightarrow_M \\
&(a_1 aaaa\#bbbbcccc, a_2 aaaa\#bbbbcccc) \Rightarrow_M (b_1 aaa\#bbbcccc, x_2 aaaa\#bbbcccc) \Rightarrow_M \\
&(b_1 aa\#bbcccc, x_2 aaaa\#bbcccc) \Rightarrow_M (b_1 a\#bcccc, x_2 aaaa\#bcccc) \Rightarrow_M \\
&(b_1\#cccc, x_2 aaaa\#cccc) \Rightarrow_M (c_1 aaaa\#cccc, x_2 aaaa\#cccc) \Rightarrow_M \\
&(c_1 aaa\#ccc, x_2 aaaa\#ccc) \Rightarrow_M (c_1 aa\#cc, x_2 aaaa\#cc) \Rightarrow_M (c_1 a\#c, x_2 aaaa\#c) \Rightarrow_M \\
&(c_1\#\varepsilon, x_2 aaaa\#\varepsilon) \Rightarrow_M (f\varepsilon\varepsilon, x_2 aaaa\#\varepsilon)
\end{aligned}
$$

3.2.1 Power

Example 3 shows that PCPDAS are stronger than PDA and PCPDAS are actually equivalent to Turing machines.

Lemma 4. *$\mathscr{L}(TM) \subseteq \mathscr{L}(PCPDAS\ of\ degree\ 3)$*

Construction. For a Turing machine $M = (\Sigma, S, \delta, s, F, \Gamma, \Delta)$, construct a PCPDAS $N = (\Sigma, \Gamma \cup \{\Xi, \Omega, \mathcal{L}, \mathcal{R}\}, \{Q_1, Q_2, Q_3\}, (S \cup S_1, \delta_1, s_1, F, \Omega), (S_2, \delta_2, s_2, \emptyset, \Omega), (S_3, \delta_3, s_3, \emptyset, \Xi))$ by performing following steps

1. For all $x \in \Sigma$, add $s_1 x\varepsilon \to s_1 x$ into δ_1 and for all $i \in \{2, 3\}$, add $s_i x\varepsilon \to s_i\varepsilon$ into δ_i
2. Add $s_1\varepsilon\varepsilon \to \langle r, \Delta, 1\rangle_1\varepsilon, s_2\varepsilon\varepsilon \to \langle r, \Delta, 1\rangle_2\varepsilon, s_3\varepsilon\varepsilon \to \langle r, 1\rangle_3\varepsilon$ into $\delta_1, \delta_2, \delta_3$, respectively
3. For all $d \in \Gamma$, add the following into $\delta_1, \delta_2, \delta_3$, respectively
   (a) $\langle r, d, 1\rangle_1\varepsilon\varepsilon \to \langle r, d, 2\rangle_1\varepsilon, \langle r, d, 1\rangle_2\varepsilon\varepsilon \to \langle r, d, 2\rangle_2\varepsilon, \langle r, 1\rangle_3\varepsilon\varepsilon \to \langle r, 2\rangle_3 Q_1$
   (b) $\langle r, d, 2\rangle_1\varepsilon\varepsilon \to \langle r, d, 3\rangle_1 Q_3, \langle r, d, 2\rangle_2\varepsilon\varepsilon \to \langle r, d, 3\rangle_2 Q_3, \langle r, 2\rangle_3\varepsilon\varepsilon \to \langle r, 2\rangle_3\varepsilon$
   (c) $\langle r, d, 3\rangle_1\varepsilon\Omega \to \langle r, d, \Omega 1\rangle_1\varepsilon, \langle r, d, 3\rangle_2\varepsilon\Omega \to \langle r, d, \Omega 1\rangle)_2\varepsilon, \langle r, 3\rangle_3\varepsilon\Omega \to \langle r, \Omega 1\rangle_3\varepsilon$
   (d) For all $p \neq \Omega$, $\langle r, d, 3\rangle_1\varepsilon p \to \langle r, p, 4\rangle_1\varepsilon, \langle r, p, 3\rangle_2\varepsilon\varepsilon \to \langle r, p, 4\rangle_2\varepsilon, \langle r, 3\rangle_3\varepsilon\varepsilon \to \langle r, 4\rangle_3\varepsilon$
   (e) $\langle r, d, 4\rangle_1\varepsilon\Omega \to \langle r, d, \Omega 1\rangle_1\varepsilon, \langle r, d, 4\rangle_2\varepsilon\Omega \to \langle r, \Omega 1\rangle_2\varepsilon, \langle r, 4\rangle_3\varepsilon\Omega \to \langle r, \Omega 1\rangle_3\varepsilon$
   (f) For all $p \neq \Omega$, $\langle r, d, 4\rangle_1\varepsilon p \to \langle r, d, 5\rangle_1 p, \langle r, d, 4\rangle_2\varepsilon p \to \langle r, d, 5\rangle_2 p, \langle r, 4\rangle_3\varepsilon p \to \langle r, 5\rangle_3 p$
   (g) For all $p \neq \Xi$, $\langle r, d, 5\rangle_1\varepsilon p \to \langle r, d, 5\rangle_1\varepsilon, \langle r, d, 5\rangle_2\varepsilon p \to \langle r, d, 5\rangle_2\varepsilon, \langle r, 5\rangle_3\varepsilon p \to \langle r, 5\rangle_3\varepsilon$
   (h) $\langle r, d, 5\rangle_1\varepsilon\Xi \to \langle r, d, 6\rangle_1\varepsilon, \langle r, d, 5\rangle_2\varepsilon\Xi \to \langle r, d, 6\rangle_2\varepsilon, \langle r, 5\rangle_3\varepsilon\Xi \to \langle r, 5\rangle_3\Xi$
   (i) $\langle r, d, 6\rangle_1\varepsilon d \to \langle r, d, 1\rangle_1\varepsilon, \langle r, d, 6\rangle_2\varepsilon\varepsilon \to \langle r, d, 1\rangle_2 d, \langle r, 6\rangle_3\varepsilon\varepsilon \to \langle r, 1\rangle_3\varepsilon$
   (j) $\langle r, d, \Omega 1\rangle_1\varepsilon\Omega \to \langle r, d, \Omega 2\rangle_1\varepsilon, \langle r, \Omega 1\rangle_2\varepsilon\Omega \to \langle r, \Omega 2\rangle_2\varepsilon, \langle r, \Omega 1\rangle_3\varepsilon\Omega \to \langle r, \Omega 2\rangle_3\varepsilon$
   (k) $\langle r, d, \Omega 2\rangle_1\varepsilon\Xi \to \langle r, d, \Omega 3\rangle_1\varepsilon, \langle r, \Omega 2\rangle_2\varepsilon\Xi \to \langle r, \Omega 3\rangle_2\varepsilon, \langle r, \Omega 2\rangle_3\varepsilon\Xi \to \langle r, \Omega 3\rangle_3\Xi$
   (l) $\langle r, d, \Omega 3\rangle_1\varepsilon\Omega \to sd, \langle r, \Omega 3\rangle_2\varepsilon\varepsilon \to \langle 0\rangle_2\varepsilon, \langle r, \Omega 3\rangle_3\varepsilon\varepsilon \to \langle 0\rangle_3\varepsilon$
4. For all $qt \to q't'm \in \delta$, $\alpha = qtq't'm$, add the following into $\delta_1, \delta_2, \delta_3$, respectively
   (a) $q\varepsilon t \to \langle \alpha, 1\rangle_1 mt', \langle 0\rangle_2\varepsilon\varepsilon \to \langle 1\rangle_2\varepsilon, \langle 0\rangle_3\varepsilon\varepsilon \to \langle 1\rangle_3\varepsilon$
   (b) $\langle \alpha, 1\rangle_1\varepsilon\varepsilon \to \langle \alpha, 2\rangle_1\varepsilon, \langle, 1\rangle_2\varepsilon\varepsilon \to \langle 2\rangle_2\varepsilon, \langle 1\rangle_3\varepsilon\varepsilon \to \langle 2\rangle_3 Q_1$
   (c) $\langle \alpha, 2\rangle_1\varepsilon m \to \langle \alpha, 3\rangle_1 Q_3, \langle 2\rangle_2\varepsilon\varepsilon \to \langle 3\rangle_2 Q_3, \langle 2\rangle_3\varepsilon\varepsilon \to \langle 3\rangle_3\varepsilon$
   (d) $\langle \alpha, 3\rangle_1\varepsilon m \to \langle \alpha, 4m\rangle_1\varepsilon, \langle 3\rangle_2\varepsilon m \to \langle 4m\rangle_2\varepsilon, \langle 3\rangle_3\varepsilon m \to \langle 4m\rangle_{,3}\varepsilon$
5. For all $qt \to q't'\mathcal{L} \in \delta$, $\alpha = qtq't'\mathcal{L}$, add the following into $\delta_1, \delta_2, \delta_3$, respectively
   (a) $\langle \alpha, 4\mathcal{L}\rangle_1\varepsilon t' \to \langle \alpha, 5\mathcal{L}\rangle_1\varepsilon, \langle 4\mathcal{L}\rangle_2\varepsilon t' \to \langle t', 5\mathcal{L}\rangle_2\varepsilon, \langle 4\mathcal{L}\rangle_3\varepsilon t' \to \langle 5\mathcal{L}\rangle_3\varepsilon$
   (b) $\forall x \neq \Xi$, $\langle \alpha, 5\mathcal{L}\rangle_1\varepsilon x \to \langle \alpha, 5\mathcal{L}\rangle_1\varepsilon, \langle t', 5\mathcal{L}\rangle_2\varepsilon x \to \langle t', 5\mathcal{L}\rangle_2\varepsilon, \langle 5\mathcal{L}\rangle_3\varepsilon x \to \langle 5\mathcal{L}\rangle_3\varepsilon$
   (c) $\langle \alpha, 5\mathcal{L}\rangle_1\varepsilon\Xi \to \langle \alpha, 6\mathcal{L}\rangle_1\varepsilon, \langle t', 5\mathcal{L}\rangle_2\varepsilon\Xi \to \langle t', 6\mathcal{L}\rangle_2\varepsilon, \langle 5\mathcal{L}\rangle_3\varepsilon\Xi \to \langle 6\mathcal{L}\rangle_3\Xi$

(d) $\langle \alpha, 6\mathcal{L}\rangle_1 \varepsilon t' \to \langle \alpha, 7\mathcal{L}\rangle \varepsilon, \langle t', 6\mathcal{L}\rangle_2 \varepsilon\varepsilon \to \langle 7\mathcal{L}\rangle t', \langle 6\mathcal{L}\rangle_3 \varepsilon \Xi \to \langle 7\mathcal{L}\rangle \Xi$

(e) $\forall x \in \Gamma,\ \langle \alpha, 7\mathcal{L}\rangle_1 \varepsilon x \to q'x, \langle \alpha, 7\mathcal{L}\rangle_2 \varepsilon\varepsilon \to \langle 0\rangle_2 \varepsilon, \langle \alpha, 7\mathcal{L}\rangle \varepsilon\varepsilon \to \langle 0\rangle_3 \varepsilon$

6. For all $qt \to q't'\mathcal{R} \in \delta$, $\alpha = qtq't'\mathcal{R}$, add the following into $\delta_1, \delta_2, \delta_3$, respectively

(a) $\forall x \neq \Xi,\ \langle \alpha, 4\mathcal{R}\rangle_1 \varepsilon x \to \langle \alpha, 4\mathcal{R}\rangle_1 \varepsilon, \langle 4\mathcal{R}\rangle_2 \varepsilon x \to \langle 4\mathcal{R}\rangle_2 \varepsilon, \langle 4\mathcal{R}\rangle_3 \varepsilon x \to \langle 4\mathcal{R}\rangle_3 \varepsilon$

(b) $\langle \alpha, 4\mathcal{R}\rangle_1 \varepsilon \Xi \to \langle \alpha, 5\mathcal{R}\rangle_1 Q_2 \Xi, \langle 4\mathcal{R}\rangle_2 \varepsilon \Xi \to \langle 5\mathcal{R}\rangle_2 \Xi, \langle 4\mathcal{R}\rangle_3 \varepsilon \Xi \to \langle 5\mathcal{R}\rangle_3 Q_2 \Xi$

(c) $\langle \alpha, 5\mathcal{R}\rangle_1 \varepsilon\varepsilon \to \langle \alpha, 6\mathcal{R}\rangle_1 \varepsilon, \langle 5\mathcal{R}\rangle_2 \varepsilon \Xi \to \langle 6\mathcal{R}\rangle_2 Q_3, \langle 5\mathcal{R}\rangle_3 \varepsilon\varepsilon \to \langle 6\mathcal{R}\rangle_3 \varepsilon$

(d) $\langle \alpha, 6\mathcal{R}\rangle_1 \varepsilon \Xi \to \langle \alpha, 7\mathcal{R}\rangle_1 \varepsilon, \langle 6\mathcal{R}\rangle_2 \varepsilon \Xi \to \langle 7\mathcal{R}\rangle_2 \varepsilon, \langle 6\mathcal{R}\rangle_3 \varepsilon \Xi \to \langle 7\mathcal{R}\rangle_3 \varepsilon$

(e) $\forall p,\ \langle \alpha, 6\mathcal{R}\rangle_1 \varepsilon p \to \langle \alpha, p, 7\mathcal{R}\rangle_1 \varepsilon, \langle 6\mathcal{R}\rangle_2 \varepsilon p \to \langle 7\mathcal{R}\rangle_2 \varepsilon, \langle 6\mathcal{R}\rangle_3 \varepsilon p \to \langle 7\mathcal{R}\rangle_3 \varepsilon$

(f) $\forall x \neq \Xi, p \in \Gamma,\ \langle \alpha, p, 7\mathcal{R}\rangle_1 \varepsilon x \to \langle \alpha, p, 7\mathcal{R}\rangle_1 \varepsilon, \langle 7\mathcal{R}\rangle_2 \varepsilon x \to \langle 7\mathcal{R}\rangle_2 \varepsilon, \langle 7\mathcal{R}\rangle_3 \varepsilon x \to \langle 7\mathcal{R}\rangle_3 \varepsilon$

(g) $p \in \Gamma,\ \langle \alpha, p, 7\mathcal{R}\rangle_1 \varepsilon \Xi \to \langle \alpha, p, 8\mathcal{R}\rangle_1 \varepsilon, \langle 7\mathcal{R}\rangle_2 \varepsilon \Xi \to \langle 8\mathcal{R}\rangle_2 \varepsilon, \langle 7\mathcal{R}\rangle_3 \varepsilon \Xi \to \langle 8\mathcal{R}\rangle_3 \Xi$

(h) $p \in \Gamma,\ \langle \alpha, p, 8\mathcal{R}\rangle_1 \varepsilon\varepsilon \to \langle \alpha, 9\mathcal{R}\rangle_1 p, \langle 8\mathcal{R}\rangle_2 \varepsilon p \to \langle 9\mathcal{R}\rangle_2 \varepsilon, \langle 8\mathcal{R}\rangle_3 \varepsilon \Xi \to \langle 9\mathcal{R}\rangle_3 \Xi$

(i) $\langle \alpha, 9\mathcal{R}\rangle_1 \varepsilon\varepsilon \to q'\varepsilon, \langle \alpha, 9\mathcal{R}\rangle_2 \varepsilon\Omega \to \langle 0\rangle_2 \Delta\Omega, \langle 9\mathcal{R}\rangle_3 \varepsilon \Xi \to \langle 0\rangle_3 \Xi$

(j) $\forall x \neq \Omega, \langle \alpha, 9\mathcal{R}\rangle_1 \varepsilon\varepsilon \to q'\varepsilon, \langle 9\mathcal{R}\rangle_2 \varepsilon x \to \langle 0\rangle_2 x, \langle 9\mathcal{R}\rangle_3 \varepsilon \Xi \to \langle 0\rangle_3 \Xi$

7. add all states that occur in δ_1, δ_2, and δ_3 into S_1, S_2, and S_3, respectively

Basic idea. Simulate every step of M by performing synchronized steps in N with three components. Two components simulate the tape. The first component has the symbol under the head and symbols to the left. The second component has symbols to the right. The first component selects rules and other components read the chosen rule and do necessary steps to simulate it. The third component is there to provide temporary storage that allows synchronization.

Induction basis. Before simulating the first step of M, N must prepare an equivalent configuration. The initial configuration of M has the form

state	tape
s	$\underline{a}bc\cdots z\Delta\cdots$

its equivalent in N would have this configuration

component	state	pushdown	input
1	s	a	ε
2	$\langle 0\rangle_2$	$bc\cdots yz\Delta\Omega$	ε
3	$\langle 0\rangle_3$	Ξ	ε

Component 1 has the part of tape left of the head and the head, 2 has the part of tape right of the head. The initial configuration of N is (the order of components is always the same, so their indexes are omitted)

state	pushdown	input
s_1	Ω	$abc\cdots yz$
s_2	Ω	$abc\cdots yz$
s_3	Ξ	$abc\cdots yz$

The first step is to add Δ symbol to component 2, to simulate the infinite part of the tape, (column headers are omitted if the meaning of columns hasn't changed)

$$\begin{array}{c|l|l} \langle r, \Delta, 1\rangle_1 & \Omega & abc\cdots yz \\ \langle r, \Delta, 1\rangle_2 & \Delta\Omega & abc\cdots yz \\ \langle r, 1\rangle_3 & \Xi & abc\cdots yz \end{array}$$

The next part is reading the input. Component 1 pushes symbols from input onto the stack. The rest just discard read symbol in order to keep synchronized. After finishing two cycles, the configuration will look like,

$$\begin{array}{c|l|l} \langle r, b, 1\rangle_1 & ba\Omega & c\cdots yz \\ \langle r, b, 1\rangle_2 & \Delta\Omega & c\cdots yz \\ \langle r, 1\rangle_3 & \Xi & c\cdots yz \end{array}$$

Components continue to read the input, The input is never read afterwards, so while there is a non-deterministic epsilon rule to the next phase. All components must read the whole input in order for the word to be accepted, thus the synchronization is guaranteed in all cases that matter. The desired configuration has been reached,

$$\begin{array}{c|l} s & a \\ \langle 0\rangle_2 & bc\cdots yz\Delta\Omega \\ \langle 0\rangle_3 & \Xi \end{array}$$

Induction step.

A step of M,

$$q \mid \overrightarrow{\alpha}\, a\underline{t}b\beta\Delta\cdots \Rightarrow_M q' \mid \overrightarrow{\alpha}\, \underline{a}t'b\beta\Delta\cdots$$

had to be done with a rule $qt \to q't'\mathcal{L}$. An equivalent sequence of steps in N would be

$$\begin{array}{c|l} q & ta\overleftarrow{\alpha} \\ \langle 0\rangle_2 & b\beta\Delta\Omega \\ \langle 0\rangle_3 & \Xi \end{array} \Rightarrow_N^* \begin{array}{c|l} q' & a\overleftarrow{\alpha} \\ \langle 0\rangle_2 & t'b\beta\Delta\Omega \\ \langle 0\rangle_3 & \Xi \end{array}$$

$\overleftarrow{\alpha}$ is reversed $\overrightarrow{\alpha}$, both are finite strings.

The sequence begins when component 1 non-deterministically starts simulating rule $qt \to q't'\mathcal{L}$, $\rho = qtq't'\mathcal{L}$.

$$\Rightarrow_N \begin{array}{c|l} \langle \rho, 1\rangle_1 & \mathcal{L}t'a\overleftarrow{\alpha} \\ \langle 1\rangle_2 & b\beta\Delta\Omega \\ \langle 1\rangle_3 & \Xi \end{array}$$

Component 1 pushed the direction on stack, this will allow component 2 to tell which rule is being simulated. component 2 will query the symbol, but because the stack of component 1 has unknown length, component 3 must be used to remain synchronized.

$$\Rightarrow_N \begin{array}{c|l} \langle \rho, 2\rangle_1 & \mathcal{L}t'a\overleftarrow{\alpha} \\ \langle 2\rangle_2 & b\beta\Delta\Omega \\ \langle 2\rangle_3 & \mathcal{L}t'a\overleftarrow{\alpha}\Xi \end{array} \Rightarrow_N \begin{array}{c|l} \langle \rho, 3\rangle_1 & \mathcal{L}t'a\overleftarrow{\alpha}\Xi t'a\overleftarrow{\alpha} \\ \langle 3\rangle_2 & \mathcal{L}t'a\overleftarrow{\alpha}\Xi b\beta\Delta\Omega \\ \langle 3\rangle_3 & \mathcal{L}t'a\overleftarrow{\alpha}\Xi \end{array}$$

Component 3 copies the stack of component 1 and then components 1 and 2 copy the stack of component 3. The direction is now on top of all stacks, so components pop and remember it.

$$\Rightarrow_N \begin{array}{c|l} \langle \rho, 4\mathcal{L}\rangle_1 & t'a\overleftarrow{\alpha}\Xi t'a\overleftarrow{\alpha} \\ \langle 4\mathcal{L}\rangle_2 & t'a\overleftarrow{\alpha}\Xi b\beta\Delta\Omega \\ \langle 4\mathcal{L}\rangle_3 & t'a\overleftarrow{\alpha}\Xi \end{array}$$

Component 2 now knows that the head is moving left and that it should remember the first symbol. The rest of the queried stack is not interesting and has the same length, so all components pop a symbol in each step to reach Ξ.

$$\Rightarrow_N \begin{array}{c|l} \langle \rho, 5\mathcal{L}\rangle_1 & a\overleftarrow{\alpha}\Xi t'a\overleftarrow{\alpha} \\ \langle t', 5\mathcal{L}\rangle_2 & a\overleftarrow{\alpha}\Xi b\beta\Delta\Omega \\ \langle 5\mathcal{L}\rangle_3 & a\overleftarrow{\alpha}\Xi \end{array} \Rightarrow_N^* \begin{array}{c|l} \langle \rho, 5\mathcal{L}\rangle_1 & \Xi t'a\overleftarrow{\alpha} \\ \langle t', 5\mathcal{L}\rangle_2 & \Xi b\beta\Delta\Omega \\ \langle 5\mathcal{L}\rangle_3 & \Xi \end{array}$$

Ξ is not needed anymore and final two steps simulate a move of the head and enter a configuration to simulate a next rule of M.

$$\Rightarrow_N \begin{array}{c|l} \langle \rho, 6\mathcal{L}\rangle_1 & t'a\overleftarrow{\alpha} \\ \langle t', 6\mathcal{L}\rangle_2 & b\beta\Delta\Omega \\ \langle 6\mathcal{L}\rangle_3 & \Xi \end{array} \Rightarrow_N \begin{array}{c|l} \langle \rho, 7\mathcal{L}\rangle_1 & a\overleftarrow{\alpha} \\ \langle 7\mathcal{L}\rangle_2 & t'b\beta\Delta\Omega \\ \langle 7\mathcal{L}\rangle_3 & \Xi \end{array} \Rightarrow_N \begin{array}{c|l} q' & a\overleftarrow{\alpha} \\ \langle 0\rangle_2 & t'b\beta\Delta\Omega \\ \langle 0\rangle_3 & \Xi \end{array}$$

Components couldn't have reached any other configuration after the rule was decided in the first step of the sequence.

A step of M,

$$q \mid \overrightarrow{\alpha}\, a\underline{t}b\beta\Delta\cdots \Rightarrow_M q' \mid \overrightarrow{\alpha}\, at'\underline{b}\beta\Delta\cdots$$

has to be done with a rule $qt \to q't'\mathcal{R}$. An equivalent sequence of steps in N would be

$$\begin{array}{c|l} q & ta\overleftarrow{\alpha} \\ \langle 0\rangle_2 & b\beta\Delta\Omega \\ \langle 0\rangle_3 & \Xi \end{array} \Rightarrow_N^* \begin{array}{c|l} q' & bt'a\overleftarrow{\alpha} \\ \langle 0\rangle_2 & \beta\Delta\Omega \\ \langle 0\rangle_3 & \Xi \end{array}$$

Component 2 must again learn the direction,

$$\begin{array}{c|l} q & ta\overleftarrow{\alpha} \\ \langle 0\rangle_2 & b\beta\Delta\Omega \\ \langle 0\rangle_3 & \Xi \end{array} \Rightarrow_N \begin{array}{c|l} \langle \rho, 1\rangle_1 & \mathcal{R}t'a\overleftarrow{\alpha} \\ \langle 1\rangle_2 & b\beta\Delta\Omega \\ \langle 1\rangle_3 & \Xi \end{array} \Rightarrow_N \begin{array}{c|l} \langle \rho, 2\rangle_1 & \mathcal{R}t'a\overleftarrow{\alpha} \\ \langle 2\rangle_2 & b\beta\Delta\Omega \\ \langle 2\rangle_3 & \mathcal{R}t'a\overleftarrow{\alpha}\Xi \end{array}$$

$$\Rightarrow_N \begin{array}{c|l} \langle \rho, 3\rangle_1 & \mathcal{R}t'a\overleftarrow{\alpha}\Xi t'a\overleftarrow{\alpha} \\ \langle 3\rangle_2 & \mathcal{R}t'a\overleftarrow{\alpha}\Xi b\beta\Delta\Omega \\ \langle 3\rangle_3 & \mathcal{R}t'a\overleftarrow{\alpha}\Xi \end{array} \Rightarrow_N \begin{array}{c|l} \langle \rho, 4\mathcal{R}\rangle_1 & t'a\overleftarrow{\alpha}\Xi t'a\overleftarrow{\alpha} \\ \langle 4\mathcal{R}\rangle_2 & t'a\overleftarrow{\alpha}\Xi b\beta\Delta\Omega \\ \langle 4\mathcal{R}\rangle_3 & t'a\overleftarrow{\alpha}\Xi \end{array}$$

After learning the direction, the rest of the stack has no use and is popped all the way to Ξ,

$$\Rightarrow_N^* \begin{array}{c|l} \langle \rho, 4\mathcal{R}\rangle_1 & \Xi t'a\overleftarrow{\alpha} \\ \langle 4\mathcal{R}\rangle_2 & \Xi b\beta\Delta\Omega \\ \langle 4\mathcal{R}\rangle_3 & \Xi \end{array}$$

Component 1 needs to know the symbol on top of component's 2 stack,

$$\Rightarrow_N \begin{array}{c|l} \langle \rho, 5\mathcal{R}\rangle_1 & t'a\overleftarrow{\alpha} \\ \langle 5\mathcal{R}\rangle_2 & b\beta\Delta\Omega \\ \langle 5\mathcal{R}\rangle_3 & b\beta\Delta\Omega\Xi \end{array} \Rightarrow_N \begin{array}{c|l} \langle \rho, 6\mathcal{R}\rangle_1 & b\beta\Delta\Omega\Xi t'a\overleftarrow{\alpha} \\ \langle 6\mathcal{R}\rangle_2 & b\beta\Delta\Omega\Xi b\beta\Delta\Omega \\ \langle 6\mathcal{R}\rangle_3 & b\beta\Delta\Omega\Xi \end{array} \Rightarrow_N \begin{array}{c|l} \langle b, \rho, 7\mathcal{R}\rangle_1 & \beta\Delta\Omega\Xi t'a\overleftarrow{\alpha} \\ \langle 7\mathcal{R}\rangle_2 & \beta\Delta\Omega\Xi b\beta\Delta\Omega \\ \langle 7\mathcal{R}\rangle_3 & \beta\Delta\Omega\Xi \end{array}$$

and stacks are popped all the way to Ξ.

$$\Rightarrow_N^* \begin{array}{c|l} \langle b,\rho,7\mathcal{R}\rangle_1 & \Xi t' a \overleftarrow{\alpha} \\ \langle 7\mathcal{R}\rangle_2 & \Xi b\beta\Delta\Omega \\ \langle 7\mathcal{R}\rangle_3 & \Xi \end{array}$$

Final steps will simulate the movement of the head and prepare to simulate a next rule.

$$\Rightarrow_N \begin{array}{c|l} \langle b,\rho,8\mathcal{R}\rangle_1 & t' a \overleftarrow{\alpha} \\ \langle 8\mathcal{R}\rangle_2 & b\beta\Delta\Omega \\ \langle 8\mathcal{R}\rangle_3 & \Xi \end{array} \Rightarrow_N \begin{array}{c|l} \langle b,\rho,9\mathcal{R}\rangle_1 & bt' a \overleftarrow{\alpha} \\ \langle 9\mathcal{R}\rangle_2 & \beta\Delta\Omega \\ \langle 9\mathcal{R}\rangle_3 & \Xi \end{array} \Rightarrow_N \begin{array}{c|l} q' & bt' a \overleftarrow{\alpha} \\ \langle 0\rangle_2 & \beta\Delta\Omega \\ \langle 0\rangle_3 & \Xi \end{array}$$

Proof. A full rigorous proof would be long, tedious, and less helpful than the semi-formal basic idea, so only cases that were not covered in the basic idea are proven.

Moving the head left while on the first cell. Let a turing machine M be in configuration $q \mid \underline{a}\beta\Delta\cdots$, where all possible rules move to left, $qt \to q't'\mathcal{L}$. M halts, because it cannot go before the first cell. The configuration of M is equivalent to the following configuration of simulating PDA N,

$$\begin{array}{c|l} q & t \\ \langle 0\rangle_2 & \beta\Delta\Omega \\ \langle 0\rangle_3 & \Xi \end{array}$$

And will go through the sequence shown in basic idea until

$$\Rightarrow_N^* \begin{array}{c|l} \langle qtq't',6\mathcal{L}\rangle_1 & t \\ \langle t',6\mathcal{L}\rangle_2 & \beta\Delta\Omega \\ \langle 6\mathcal{L}\rangle_3 & \Xi \end{array} \Rightarrow_N \begin{array}{c|l} \langle qtq't',7\mathcal{L}\rangle_1 & \\ \langle 7\mathcal{L}\rangle_2 & t'\beta\Delta\Omega \\ \langle 7\mathcal{L}\rangle_3 & \Xi \end{array}$$

State $\langle qtq't',7\mathcal{L}\rangle$ is not final and the stack is empty at that point, so the component is stuck and does not accept the word.

Infinite tape to the right. Unlike the tape of Turing machine, the stack that represents tape on the right does have end with infinitely many Δ symbols. When a Turing machine M uses a rule $qt \to q't'\mathcal{R}$ on the rightmost visited cell, the transition between two configurations is

$$q \mid \overrightarrow{\alpha}\underline{t}\Delta\cdots \Rightarrow_M q' \mid \overrightarrow{\alpha}t'\underline{\Delta}\Delta\cdots$$

A simulating PDA N does the equivalent transition,

$$\begin{array}{c|l} q & t\overleftarrow{\alpha} \\ \langle 0\rangle_2 & \Delta\Omega \\ \langle 0\rangle_3 & \Xi \end{array} \Rightarrow_N^* \begin{array}{c|l} q & \Delta t\overleftarrow{\alpha} \\ \langle 0\rangle_2 & \Delta\Omega \\ \langle 0\rangle_3 & \Xi \end{array}$$

The stepping sequence is the same as provided in basic idea until component 2 reaches state $\langle 9\mathcal{R}\rangle_2$.

$$\Rightarrow_N^* \begin{array}{c|c|c} \langle b,\rho,8\mathcal{R}\rangle_1 & t'\overleftarrow{\alpha} & \\ \langle 8\mathcal{R}\rangle_2 & \Delta\Omega & \Rightarrow_N \\ \langle 8\mathcal{R}\rangle_3 & \Xi & \end{array} \begin{array}{c|c|c} \langle b,\rho,9\mathcal{R}\rangle_1 & \Delta t'\overleftarrow{\alpha} & \\ \langle 9\mathcal{R}\rangle_2 & \Omega & \Rightarrow_N \\ \langle 9\mathcal{R}\rangle_3 & \Xi & \end{array} \begin{array}{c|c|c} & q' & \\ & \langle 0\rangle_2 & \Delta\Omega \\ & \langle 0\rangle_3 & \Xi \end{array}$$

A special case in state $\langle 9\mathcal{R}\rangle_2$ recognizes the end of tape symbol Ω and pushes Δ before it, simulating an infinite tape filled with Δ.

Desynchronization. The construction introduces one source of non-determinism when reading the input, which can cause desynchronization. If the input word does belong to the language, then all components read it whole, because reading is only done in very first steps and a the word is not accepted if any component did not read the whole input. If any component reads ε prematurely, then the word is not accepted regardless of what happens next. □

Theorem 3. *$\mathscr{L}$(PCPDAS of rank 3) = $\mathscr{L}$(TM)*

Proof. Follows from Lemma 4 and Church-Turing thesis. □

Theorem 4. *$\mathscr{L}$(PCPDAS of rank 2) = $\mathscr{L}$(TM)*

Proof. PCPDAS gets stuck if two components query each other during one step, so exchange of stack contents must take at least two steps. Construction in Lemma 4 uses a third component to read a stack of other component while maintaining synchronization, but a more complicated communication protocol can do that with just two components.

Construction in Lemma 4 simulates TM by using a configuration that whose first component has an arbitrary stack α and the second component β, $|\alpha| = a$, $|\beta| = b$, while the third one has one Ξ symbol on stack. PCPDAS of rank 3 does the following sequence resembling the following one to communicate stack between first two components.

$$\begin{array}{c} \alpha \\ \beta \\ \Xi \end{array} \Rightarrow \begin{array}{c} \alpha \\ \beta \\ \alpha\Xi \end{array} \Rightarrow \begin{array}{c} \alpha\Xi\alpha \\ \alpha\Xi\beta \\ \alpha\Xi \end{array} \Rightarrow^a \begin{array}{c} \Xi\alpha \\ \Xi\beta \\ \Xi \end{array} \Rightarrow \begin{array}{c} \alpha \\ \beta \\ \Xi \end{array} \Rightarrow \begin{array}{c} \beta\Xi\alpha \\ \beta\Xi\beta \\ \beta\Xi \end{array} \Rightarrow^b \begin{array}{c} \Xi\alpha \\ \Xi\beta \\ \Xi \end{array} \Rightarrow \begin{array}{c} \alpha \\ \beta \\ \Xi \end{array}$$

Synchronization is trivial as prefix of the stack has same length, so each step of $\Rightarrow^*$ consumes one symbol on all stacks. The situation is more complicated for rank 2 PCPDAS. Two components can employ a communication protocol where the second component queries the first, then the first queries the second, and then the second queries the first again to gain a copy of both stacks into both components and enough information to synchronize. The full protocol that is equivalent to what rank 3 PCPDAS does is

$$\begin{array}{c} \alpha \\ \beta \end{array} \Rightarrow \begin{array}{c} \alpha \\ \alpha\Xi\beta \end{array} \Rightarrow \begin{array}{c} \alpha\Xi\beta\Xi\alpha \\ \alpha\Xi\beta \end{array} \Rightarrow \begin{array}{c} \alpha\Xi\beta\Xi\alpha \\ \alpha\Xi\beta\Xi\alpha\Xi\alpha\Xi\beta \end{array} \Rightarrow^{3a+b+4} \begin{array}{c} \alpha \\ \beta \end{array}$$

The second component consumes $3a + b + 4$ symbols in $3a + b + 4$ steps only by consuming one symbol each step, but the second component can consume only $a + b + 2$ symbols, so it spends 3 steps on every symbol from α and then two extra steps to remain synchronized with the second component.

Each step of TM simulation that rank 3 PCPDAS did in Lemma 4 can be done with rank 2 PCPDAS by performing $2a$ extra steps. □

Theorem 5. *$\mathscr{L}(PDA) = \mathscr{L}(PCPDAS\ of\ rank\ 1) \subset \mathscr{L}(PCPDAS\ of\ rank\ 2) = \mathscr{L}(TM)$*

Proof. $\mathscr{L}$(PDA) = $\mathscr{L}$(PCPDAS of rank 1) is a simple identity and the rest follows from Example 3, Theorem 4, and Theorem 3 □

3.2.2 Variants

Agreeing PCPDAS

The accepting condition of PCPDAS was to have any component its final state, which is inspired by PCGS, where only the first component generated the word of the language. Agreeing PCPDAS need all components in final state to accept the word. Agreeing PCPDAS and PCPDAS are equivalent as all states of second component in Lemma 4 could be accepting.

Returning PCPDAS

In returning PC Grammar Systems, the sentential form returns to the starting symbol and sentential form is the only information that can be returned. PCPDAS have have input, stack, and state, which comes together to 8 combinations.

1. Nothing is reset. This variant is also called non-returning PCPDAS, and the last section proved that maximal power is equal to Turing machines.
2. State is reset. The maximal power is equal to the variation where nothing is reset, which is equal to Turing machines.

   Proof. Basic idea. Perform 2 steps for each step of the original component. First step pops the pushdown symbol and sets it as the state. Second step performs an action of the original machine, but instead of switching state to the next state, it pushes the state as on top of what the original action would do. Query and therefore reset can only happen in step 2 and the next step would read the state encoded in the topmost stack symbol anyway, so a reset does not change the situation.

   There are two special cases

   - the initial stack symbol. There can be only one starting symbol, so the starting state has to recognize this special situation and perform an action that reads the initial stack symbol and pushes two in its place – the original initial starting symbol and the original initial state on top of it.
   - query copies one extra symbol, so components that did query must skip it and queried components have to do nothing for one step. This can be done, because all components know when queries happen and who does them. □
3. Stack is reset. The maximal power is equal to the variation where nothing is reset.

Proof. Basic idea. Introduce 3 more components that will serve as temporary storage for multiplication of input. All components are counting steps (in state) and on every 4th step, these components will query the stack of 3 original components. And in the 5th, original components will query extra components to get their stack back. No other query can happen on the 5th step as all other queries that original components do are postponed to happen only on every 4 steps. The stack is duplicated and not lost and synchronization is preserved even with postponenement, because all components in Lemma 4 were querying at well known steps. □

4. Stack and State is reset. The maximal power is equal to Turing machine.

   Proof. Basic idea. First restore the stack with method when only stack is reset and then restore state with method when only state is reset.

   The state was used to keep count of steps, but because the reset always happens at a set point in the count, we can encode that step index to the initial state. □

5. Input is reset. To attain power equal to the non-resetting variant, a non-standard modification has to be made.

   The problem is that accepting condition requires that the whole input was read, but the component has no way of knowing that it read the whole input, because it has to do an ε transition to move on and that transition can also be performed while there still is remaining input. The proof when nothing is reset relies on a fact that all components synchronize on the length of input and the input-reading component never touch the input after that, so non-determinism makes sure that components are synchronized if the string can be accepted.

   The input-reading component has to be queried, which resets the input in this variation. Without further changes, that would mean that no input string is ever accepted and therefore we must pop out the input when TM reaches an accepting condition, but when we reach an accepting condition, we have now way of knowing if the input was completely read, because components might have non-deterministically synchronized on a string that is not the full input. This would mean that if non-resetting component accepted word w, then input-resetting component would accept any word which has w as prefix.

   The input-resetting component is still more powerful that context-free grammar, because example on $a^n b^n c^n$ works without modifications, but if we allowed a simple change in the input, then a maximal power of Turing machine is attainable.

   Proof. Basic idea. For any language L, create a language $L_\Omega = \{w\Omega \mid w \in L\}$, where $\Omega \notin alph(L)$. L_Ω has non-ambiguous end of input, so we can ensure that the whole input is read before the first query and resetting the input does not matter after that. When a final state is reached, the component will just consume the whole input and accept the word.

   The constructed component accepts a word in L_Ω if and only if the original component accepts the word without trailing Ω. □

6. Input and State is reset. Same maximal power as when input is reset.

7. Input and Stack is reset. Same maximal power as when input is reset.

8. Input, State, and Stack is reset. Same maximal power as when input is reset.

Centralised PCPDAS

Only the first component can query in Centralised PCPDAS.

The construction from Lemma 4 is not possible, because it depends on unbounded amount of communication between 3 automata. Centralised PCPDAS are expected to be weaker than Turing machines and even linear-bounded automata. Their power is an open problem.

Conjecture. Centralised PCPDAS form an infinite hierarchy based on the number of components. *Basic idea.* A language where every letter has the same number of occurrences, $L_k = \{w \mid k = |alph(w)|, x \in alph(w) \Rightarrow occur(x, w) = n\}$, where $k > 1$, requires at least $k - 1$ components, because no component can store more than one kind of symbols on the stack and still is able to compare their number with some other symbol and all components must read the whole input to compare number of occurrences of two symbols.

Centralised PCPDAS demonstrate a difference between L_k and $M_k = \{1^n \dots k^n\}$ because M_k requires only 2 components for any k: both components start by reading all 1s, after that, the second component does nothing and first one compares the number of 1 with 2 by popping the stack while reading the input, then queries the second component and continues to compare count of 1 with 3 to k in this manner. L_k is a superset of M_k, so L_3 is a more complex language than any of M_k.

8 resetting variant of centralised PCPDAS are possible, yet none of them is expected to decrease the power.

Stopping PCPDAS

Components of Stopping PCPDAS that have been queried never resume and remain in the state they had on time of the first query. The power of Stopping PCPDAS is higher than PDA (the automaton from Example 3 is a valid Stopping PCPDAS), but below Turing machines, because the system cannot simulate an unbounded amount of moves. Resetting has no effect. Stopping PCPDAS are not stronger than Centralised PCPDAS, but their equivalence remains an open problem.

Prefix PCPDAS

When a component i queries component j, then only a non-empty prefix of component j's stack is communicated. The shortest possible prefix has length 1 and we can see that it has a power equivalent to Turing machines.

The construction of Turing machine equivalence is simplified with this modification and proving an equivalence with 2-PDA became the best choice.

Proof. Construction. For a 2-PDA $M = (\Sigma, S, \delta, s, F, \Gamma, p1, p2)$, construct Prefix PCPDAS $N = (\Sigma, \Gamma, (S_1, \delta_1, s, F, p1), (S_2, \delta_2, s, \emptyset, p2))$ by performing following steps

1. $S_1 = S \cup \{\}$, $S_1 = S \cup \{\}$

2. $\forall q i d_1 d_2 \to q' d_1' d_2' \in \delta$,
   (a) add $q\varepsilon d_1 \to \langle q, d_1, 0\rangle_1 Q_2$ to δ_1 and $q\varepsilon\varepsilon \to \langle q, 0\rangle_2 \varepsilon$ to δ_2
   (b) add $\langle q, d_1, 0\rangle_1 i d_2 \to \langle q, i, d_1, d_2, q', d_1', d_2', 1\rangle_1 \langle q, i, d_1, d_2, q', d_1', d_2'\rangle$ to δ_1 and $\langle q, 0\rangle \varepsilon\varepsilon \to \langle q, 1\rangle_2 Q_1$ to δ_2
   (c) add $\langle q, i, d_1, d_2, q', d_1', d_2', 1\rangle_1 \varepsilon \langle q, i, d_1, d_2, q', d_1', d_2'\rangle \to \langle q', 2\rangle_1 d_1'$ to δ_1 and $\langle q, 1\rangle_2 \varepsilon \langle q, i, d_1, d_2, q', d_1', d_2'\rangle \to \langle q, i, d_1, d_2, q', d_2', 2\rangle_2 \varepsilon$ to δ_2
   (d) add $\langle q', 2\rangle_1 \varepsilon\varepsilon \to q'\varepsilon$ to δ_1 and $\langle q, i, d_1, d_2, q', d_2', 2\rangle_2 i d_2 \to q' d_2'$ to δ_2

Basic idea. Component 1 takes care of the first stack and Component 2 of the seconds stack. Every step of 2-PDA uses both topmost symbols, so the component 1 queries the topmost symbol of the component 2, decided a rule that will be simulated and pushed it on stack. Component 2 reads the rule and they both perform it. □

Centralised Prefix PCPDAS

Centralised Prefix PCPDAS are equivalent to 2-PDA.

Basic idea. Because there is only one-way communication, the decision which rule to take is performed by the second component. The second component does not know which symbol is on top of the first stack, so it non-deterministically decides on a symbol and puts the selected rule on top of the second stack. The first component queries the symbol that encodes selected rule and then both components perform the one step that is possible with the rule. If the second component did not select a symbol that is on top of the first stack, then the system gets stuck. Non-determinism guarantees that the selected symbol is always correct if the input word belongs to the language.

The second component always had the option of non-deterministically determining the symbol on top of the first stack, but without prefix modification, the second component could not determine the correct number of steps to wait until the next transition, because the first component had to pop an unbounded amount of extraneous symbols that were a suffix of the second stack.

Chapter 4

Applications

The purpose of this thesis was basic research and the application of basic research is broadening of mankind's knowledge, so models from previous chapters were designed without any particular application in mind. The correct direction is to first understand a practical problem and then design a model that best fits into our physical constraints, which is incompatible with the goal of answering a theoretical question, but this chapter will start with an outline of sensible applications and then give one practical application to dendroclimatology.

CDPDAS do not yield additional power, so its best is for simplification. The obvious case are PDAs that explicitly simulated the operation of derivation rules, but if PDA does logically independent operations, then separating them into multiple automata can also bring clarity to the design. The terminating derivation mode is the most sophisticated and therefore useful in this regard.

PCPDAS are equivalent to Turing Machines and as the name would suggest, PCPDAS are better suited for problems requiring parallel cooperation.

When parallelizing, we are interested in minimizing the time to get a result, because the total amount of work cannot be smaller that the amount of work that a single computation unit would perform.

The only method to achieve a speedup with parallelization is splitting the work. *Embarrassingly parallel* problems need very little communication between workers to achieve the result. PCPDAS models those applications well, especially if each part can be recognized with a PDA.

Increasing amount of communication brings forth the main drawback of unmodified PCPDAS: overhead from copying the whole stack. Prefix PCPDAS alleviate the overhead and seem as the most useful variation.

4.1 Dendroclimatology

Protected by bark are three main layers of tree, phloem, cambium, and xylem [13]. Phloem is the outermost layer composed of living cells that transport sap with nutrients between any parts that need it. Cambium is a layer produces phloem and xylem cells, resulting in a growth in diameter. Xylem is a layer that transports water with minerals upwards. Dead xylem cells do not transport water and create a structure that supports the tree instead.

Tree growth rings are found in xylem and environmental variables (such as temperature, precipitation, soil, neighbours, etc.) determine the structure of tree rings [3]. Xylem takes most of tree trunk and most of xylem is composed of dead cells and therefore works as a log of environmental variables during a lifetime of a tree, although trees only grow for about a third to half of a year, so information about the rest of the year is lost, unless it had a long lasting effect.

Xylem grows fast during spring and is characterized with thin cellular walls and large intercellular space, while growth in summer has thicker walls and little space in between.

Evolution is very slow, so a child of a tree can be assumed to have undistinguishable growth response to environmental variables. By monitoring environmental conditions during growth of a sufficient sample of children, we can map the structure of tree rings into environmental variables. The mapping allows us to estimate environmental variables by looking at tree rings of close family members, which is beneficial in places where we had no measuring equipment, be it a remote part of the country or the same place thousand years ago.

Knowledge about effects of variables on growth rings is based on experiments and observations. The most important part is translating xylem into a sequence of symbols that can be an input to automata. There are two main options for input

1. Divide xylem into uniform chunks (e.g. 0.01 mm width) and measure the average density of cells in each chunk. A finite set of symbols is enough to express density, so the input will contain n symbols, where n is the total width of xylem divided by sample width.

2. Find places with sudden drops of density and measure the distance between them. Average width between drops (width of tree rings) in Central European oaks is 1.87 mm [4], hence just a million symbols in the input set would provide full coverage and good precision. The input would then contain $n - 1$ symbols, where n is the number of density drops (tree rings).

The latter method has a long tradition and is easier to find in databases[1], but the former method preserves more information about the wood and laboratories are investing in it. A ring width can be inferred in the former method and Maximum Latewood Density alone has good correlation to climate [5].

Prepared samples of xylems then need to be classified by using information about environmental variables that were collected while the tree was growing. For simplicity of demonstration, let us assume that only average weekly precipitation has been collected.

[1] A good source of raw datasets is https://www.ncdc.noaa.gov/data-access/paleoclimatology-data/datasets/tree-ring

Using expert input, possibly with machine learning techniques, we obtain a mapping between xylem density, age of the tree, estimated calendar week of the year, average weekly temperature and precipitation. Another possible starting point is to encode common existing models that use an idealized equation [2] with temperature and precipitations variables and quantize them into PDA rules. The advantage of automata is that special cases that were improperly modeled in the equation are easy to add without regressing cases that already had good correlation with reality.

The obtained PDA parses an input of density samples and sequence of temperature and precipitation estimates for chosen time granularity are determined from the parse tree. PDA rules are going to be ambiguous, because there are many other environmental variables that we have ignored, which is where parallel communication comes into play.

The recommended minimal number of samples to estimate the historical precipitation and temperature is 20 trees per site [1], so k samples are used as an input to a PCPDAS of degree $> k$. Component 1 to k will parse on sample and all of them will work in parallel, having expected precipitation and temperature pairs on the stack and component $k+1$ will collect results and accept the input only if all parses yield the same result. Samples and their parsing might not always be in agreement, so component $k+2$ would compute an average and accept only if component $k+1$ fails.

PCPDAS can also exploit embarrassingly parallel property of the problem – the input string for each tree can be divided into years and each component will take only few years, with year or two overlaps to provide context for possible disasters with long-lasting effect. Dividing the input into smaller chunks is also beneficial for performance, because nondeterminism is a property that has to be simulated with exhaustive search with above-linear time complexity and smaller search space is an effective way to keep a feasible computation time.

Chapter 5

Summary and Conclusion

Chapter 2 introduces cooperating distributed pushdown automata systems, a new class of automata that adapts the main concept from cooperating distributed context-free grammar systems to pushdown automata. CDPDA contains an arbitrary amount of PDA and the system employs stepping rules to control the amount of steps that each component (a PDA automaton) does before a next component is selected. Cooperation in grammar system means that grammars work on the same sentential form. CDPDAS work with the same stack and input, so each component only has its own state. There are several options of handling the state when the next component is selected. The most general variation defined the state of next component as a function of current state and last state of the next component.

CDPDAS have the same power as standard PDA. PDA have perfect control over applicable rules and the state is able to express any finite amount of information. CDPDAS regulated rules and used a counter that is potentially infinite, but the regulation is reshaped into a finite one, because it was always evaluated with $\leq$, $=$, or $\geq$ function with a constant, and stored in state.

Chapter 3 introduces parallel communicating pushdown automata systems, a new class of automata that adapts the main concept from parallel communicating context-free grammar systems to pushdown automata. PCPDAS contains an arbitrary amount of PDA and the system synchronously steps all of them in parallel. In each step, a component can query other components by placing a special symbol on the stack. Between steps, these special symbols are replaced by whole stack of the queried component.

Returning, Non-returning, and Prefix PCPDAS have the power of Turing machines, while centralised PCPDAS are weaker and conjectured to define an infinite hierarchy of languages.

This thesis demonstrates that adapting concepts from context-free grammar systems to automata has very different effect on power. Adapting CDGS, which is stronger than CF, did not surpass the power of PDA. Variants of PCGS that had different powers became equivalent when adapted to PCPDAS as adapting PCGS became turing complete in most of its variations, while only the prefix variation of PCGS covers RE and other variants have powers similar to CDGS.

CDPDAS did not increase power, because it added only a constant number of states and one state already provides perfectly controllable constant storage. CDPDAS+ are introduced as

a novel modification similar to CDPDAS, but not adapting CDGS, to test this hypothesis by adding an indirect unbounded storage that is accessible when changing components. The modification did not falsify the hypothesis as CDPDAS+ are equivalent to Turing machines.

The only studied variation of Grammar Systems that increased the power and did not end up Turing-complete is Centralised PCPDAS and their power is left as an open problem.

Bibliography

[1] National Climatic Data Center. Tree ring data description. http://www.ncdc.noaa.gov/paleo/treeinfo.html. Retrieved 2016-05-25.

[2] Harold C. Fritts, Eugene A. Vaganov, Irina V. Sviderskaya, and Alexander V. Shashin. Climatic variation and tree-ring structure in conifers: empirical and mechanistic models of tree-ring width, number of cells, cell size, cell-wall thickness and wood density. *Climate research*, 1:192–197, 1991.

[3] A.M. García-Suárez, C.J. Butler, and M.G.L. Baillie. Climate signal in tree-ring chronologies in a temperate climate: A multi-species approach. *Dendrochronologia*, 27(3):183–198, 2009.

[4] Polona Hafner, Jožica Gričar, Mitja Skudnik, and Tom Levanič. Variations in environmental signals in tree-ring indices in trees with different growth potential. *PLoS ONE*, 10(11), 2015.

[5] Yu jiang Yuan, Tong wen Zhang, Wen shou Wei, Daniel Nievergelt, Anne Verstege, Shu long Yu, Rui bo Zhang, and Jan Esper. Development of tree-ring maximum latewood density chronologies for the western tien shan mountains, china: Influence of detrending method and climate response. *Dendrochronologia*, 31(3):192–197, 2003.

[6] Alexander Meduna. *Automata and Languages: Theory and Applications [Springer, 2000]*. Springer Verlag, 2005.

[7] Alexander Meduna and Petr Zemek. *Regulated Grammars and Their Transformations*. Brno University of Technology, 2010.

[8] Alexander Meduna and Petr Zemek. *Regulated Grammars and Automata*. Springer US, 2014.

[9] Gheorghe Păun. Parallel communication grammar systems: Recent results, open problems. *Acta Cybern.*, 12(4):381–395, July 1996.

[10] Ondřej Ryšavý. Cooperating distributed grammar systems: A comparison of generative power. www.fit.vutbr.cz/~meduna/mti/2001_2002/rysavy.pdf, 2002. Retrieved 2016-05-25.

[11] Jiří Techet, Tomáš Masopust, and Alexander Meduna. Cooperating distributed grammar systems. http://www.fit.vutbr.cz/~meduna/work/lib/exe/fetch.php?media=lectures:phd:tid:frvs:09-cdgspres.pdf. Retrieved 2016-05-25.

[12] Jiří Techet, Tomáš Masopust, and Alexander Meduna. Parallel communicating grammar systems. http://www.fit.vutbr.cz/~meduna/work/lib/exe/fetch.php?media=lectures:phd:tid:frvs:10-pcgspres.pdf. Retrieved 2016-05-25.

[13] Wikipedia. Xylem. https://en.wikipedia.org/wiki/Xylem. Retrieved 2016-05-25.