An automata characterisation for multiple context-free languages*

Tobias Denkinger

Faculty of Computer Science,
Technische Universität Dresden,
01062 Dresden, Germany
tobias.denkinger@tu-dresden.de

2018-10-23

Abstract

We introduce tree stack automata as a new class of automata with storage and identify a restricted form of tree stack automata that recognises exactly the multiple context-free languages.

Contents

1 Introduction **2**

2 Preliminaries **2**
2.1 Parallel multiple context-free grammars . . . 3
2.2 Automata with storage . . . 3

3 Tree stack automata **4**
3.1 Restricted TSA . . . 6
3.2 Normal forms . . . 7

4 The equivalence of MCFG and restricted TSA **9**
4.1 Every MCFG has an equivalent restricted TSA . . . 9
4.2 Every restricted TSA has an equivalent MCFG . . . 13
4.3 The main theorem . . . 16

5 Conclusion **17**

*This is an extended version of a paper with the same title accepted at DLT 2016 which is available at link.springer.com and via the DOI 10.1007/978-3-662-53132-7_12.

1 Introduction

Prominent classes of languages are often defined with the help of their generating mechanism, e.g. context-free languages are defined via context-free grammars, tree-adjoining languages via tree-adjoining grammars, and indexed languages via indexed grammars. To achieve a better understanding of how languages from a specific language class can be recognised, it is natural to ask for an automaton model. For context-free languages, this question is answered with pushdown automata [Cho62, Sch63], yield languages of tree-adjoining grammars are recognised by embedded pushdown automata [VS88, Sec. 3], and indexed languages are recognised by nested stack automata [Aho69].

Mildly context-sensitive grammars are currently prominent in natural language processing as they are able to express the non-projective constituents and dependencies that occur in natural languages [KS09, Mai10]. Multiple context-free grammars [SMFK91] describe many mildly context-sensitive grammars. Yet, to the author's knowledge, there is no corresponding automaton model. Thread automata [VdlC02b, VdlC02a], introduced by Villemonte de la Clergerie to describe parsing strategies for mildly context-sensitive grammar formalisms, already come close to such an automaton model. A construction of thread automata from ordered simple range concatenation grammars (which are equivalent to multiple context-free languages) was given [VdlC02b, Sec. 4]. A construction for the converse direction as well as proofs of correctness, however, were not provided.

Based on the idea of thread automata, we introduce a new automaton model, tree stack automata, and formalise it using automata with storage [Sco67, Eng14] in the notation of Herrmann and Vogler [HV15], see Section 3. Tree stack automata possess, in addition to the usual finite state control, the ability to manipulate a tree-shaped stack that has the tree's root at its bottom. We find a restriction of tree stack automata that makes them equivalent to multiple context-free grammars and we give a constructive proof for this equivalence, see Section 4.

2 Preliminaries

In this section we fix some notation and briefly recall formalisms used throughout this paper. We denote the set of natural numbers (including 0) by $\mathbb{N}$, $\mathbb{N} \setminus \{0\}$ by $\mathbb{N}_+$, and $\{1, \ldots, n\}$ by $[n]$ for every $n \in \mathbb{N}$. The reflexive, transitive closure of some endorelation r is denoted as r^*. For two sets A and B, we denote the set of partial functions from A to B by $A \to B$. The operator $\to$ shall be right associative. Let $f\colon A \to B$, $a \in A$, and $b \in B$. The *domain of* f, denoted by $\mathrm{dom}(f)$, is the subset of A for which f is defined. If $\mathrm{dom}(f) = A$ we call f *total*. We define $f[a \mapsto b]$ as the partial function from A to B such that $f[a \mapsto b](a) = b$ and $f[a \mapsto b](a') = f(a')$ for every $a' \in \mathrm{dom}(f) \setminus \{a\}$. We sometimes construe partial functions as relations in the usual manner. Let S be a countable set (of *sorts*) and $s \in S$. An *S-sorted set* is a tuple (B, sort) where B is a set and $\mathrm{sort}\colon B \to S$ is total. We denote the preimage of s under sort by B_s and abbreviate (B, sort) by B; sort will always be clear from the context. Let A be a set and $L \subseteq A^*$. We call L *prefix-closed* if for every $w \in A^*$ and $a \in A$ we have that $wa \in L$ implies $w \in L$.

An *alphabet* is a finite set (of *symbols*). Let Γ be an alphabet. The set of *trees over* Γ, denoted by T_Γ, is the set of partial functions from $\mathbb{N}_+^*$ to Γ with finite and prefix-closed domain. The usual definition of trees [Gue83, Sec. 2] additionally requires that for every $\rho \in \mathbb{N}_+^*$ and $n \geq 2$: if ρn is in the domain of a tree then $\rho(n-1)$ is as well; we drop this restriction here.

2.1 Parallel multiple context-free grammars

We fix a set $X = \{x_i^j \mid i, j \in \mathbb{N}_+\}$ of *variables*. Let Σ be an alphabet. The set of *composition representations over* Σ is the $(\mathbb{N}_+^* \times \mathbb{N}_+)$-sorted set RF_Σ where for every $s_1, \ldots, s_\ell, s \in \mathbb{N}_+$ we define $X_{(s_1 \cdots s_\ell, s)} = \{x_i^j \mid i \in [\ell], j \in [s_i]\} \subseteq X$ and $(\mathrm{RF}_\Sigma)_{(s_1 \cdots s_\ell, s)} = \{[u_1, \ldots, u_s]_{(s_1 \cdots s_\ell, s)} \mid u_1, \ldots, u_s \in (\Sigma \cup X_{(s_1 \cdots s_\ell, s)})^*\}$ as a set of strings in which parentheses, brackets, commas, and the elements of $\mathbb{N}_+$, Σ, and $X_{(s_1 \cdots s_\ell, s)}$ are used as symbols. Let $f = [u_1, \ldots, u_s]_{(s_1 \cdots s_\ell, s)} \in \mathrm{RF}_\Sigma$. The *composition function of* f, also denoted by f, is the function from $(\Sigma^*)^{s_1} \times \cdots \times (\Sigma^*)^{s_\ell}$ to $(\Sigma^*)^s$ such that $f((w_1^1, \ldots, w_1^{s_1}), \ldots, (w_\ell^1, \ldots, w_\ell^{s_\ell})) = (u_1', \ldots, u_s')$ where $(u_1', \ldots, u_s')$ is obtained from $(u_1, \ldots, u_s)$ by replacing each occurrence of x_i^j by w_i^j for every $i \in [\ell]$ and $j \in [s_\ell]$. The set of all composition functions for some composition representation over Σ is denoted by F_Σ. From here on we no longer distinguish between composition representations and composition functions. We define the *fan-out of* f as s. We call f *linear* (*non-deleting*) if in $u_1 \cdots u_s$ every element of X occurs at most once (at least once, respectively). The subscript is dropped from f if its sort is clear from the context.

Definition 2.1. A *parallel multiple context-free grammar (short: PMCFG)* is a tuple $G = (N, \Sigma, I, R)$ where N is a finite $\mathbb{N}_+$-sorted set (of *non-terminals*), Σ is an alphabet (of *terminals*), $I \subseteq N_1$ (*initial non-terminals*), and $R \subseteq \bigcup_{k,s,s_1,\ldots,s_k \in \mathbb{N}} N_s \times (\mathrm{F}_\Sigma)_{(s_1 \cdots s_k, s)} \times (N_{s_1} \times \cdots \times N_{s_k})$ is finite (*rules*). □

Let $G = (N, \Sigma, I, R)$ be a PMCFG. A rule $(A, f, A_1 \cdots A_k) \in R$ is usually written as $A \to f(A_1, \ldots, A_k)$; it inherits its sort from f. A PMCFG that only contains rules with linear composition functions is called a *multiple context-free grammar (short: MCFG)*. An MCFG that contains only rules of fan-out at most k is called a k-MCFG.

For every $A \in N$, we recursively define the *set of derivations in* G *from* A as $\mathrm{D}_G(A) = \{r(d_1, \ldots, d_k) \mid r = A \to f(A_1, \ldots, A_k) \in R, \forall i \in [k] \colon d_i \in \mathrm{D}_G(A_i)\}$. The elements of $\mathrm{D}_G(A)$ can be construed as trees over R. Let $d \in \mathrm{D}_G(A)$. By projecting each rule in d on its second component, we obtain a term over F_Σ; the *tuple generated by* d, denoted by $[\![d]\!]$, is obtained by evaluating this term. We identify 1-tuples of strings with strings. The *set of (complete) derivations in* G is $\mathrm{D}_G = \bigcup_{A \in N} \mathrm{D}_G(A)$ ($\mathrm{D}_G^{\mathrm{c}} = \bigcup_{S \in I} \mathrm{D}_G(S)$, respectively). The *language of* G is $L(G) = \{[\![d]\!] \mid d \in \mathrm{D}_G^{\mathrm{c}}\}$.

2.2 Automata with storage

Definition 2.2. A *storage type* is a tuple $S = (C, P, F, C_{\mathrm{i}})$ where C is a set (of *storage configurations*), $P \subseteq \mathcal{P}(C)$ (*predicates*), $F \subseteq C \to C$ (*instructions*), and $C_{\mathrm{i}} \subseteq C$ (*initial configurations*). □

Definition 2.3. An *automaton with storage* is a tuple $\mathcal{M} = (Q, S, \Sigma, q_\mathrm{i}, c_\mathrm{i}, \delta, Q_\mathrm{f})$ where Q is a finite set (of *states*), $S = (C, P, F, C_\mathrm{i})$ is a storage type, Σ is an alphabet (of *terminals*), $q_\mathrm{i} \in Q$ (*initial state*), $c_\mathrm{i} \in C_\mathrm{i}$ (*initial storage configuration*), $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times P \times F \times Q$ is finite (*transitions*), and $Q_\mathrm{f} \subseteq Q$ (*final states*). $\square$

Let $\mathcal{M} = (Q, S, \Sigma, q_\mathrm{i}, c_\mathrm{i}, \delta, Q_\mathrm{f})$ be an automaton with storage and $S = (C, P, F, C_\mathrm{i})$. Let $\tau = (q, \omega, p, f, q') \in \delta$ be a transition. We call q the *source state of* τ, p the *predicate of* τ, f the *instruction of* τ, and q' the *target state of* τ. A *configuration of* $\mathcal{M}$ is an tuple (q, c, w) where $q \in Q$, $c \in C$, and $w \in \Sigma^*$. We define the *run relation with respect to* τ as the binary relation $\vdash_\tau$ on the set of configurations of $\mathcal{M}$ such that $(q, c, w) \vdash_\tau (q', c', w')$ iff $(w = \omega w') \wedge (c \in p) \wedge (f(c) = c')$. The *set of runs in* $\mathcal{M}$ is the smallest set $\mathrm{R}_\mathcal{M} \subseteq \delta^*$ where for every $k \in \mathbb{N}$ and $\tau_1, \ldots, \tau_k \in \delta$, the string $\theta = \tau_1 \cdots \tau_k$ is in $\mathrm{R}_\mathcal{M}$ if there are $q_0, \ldots, q_k \in Q$, $c_0, \ldots, c_k \in C$, and $\omega_1, \ldots, \omega_k \in \Sigma \cup \{\varepsilon\}$ such that $(q_0, c_0, \omega_1 \cdots \omega_k) \vdash_{\tau_1} (q_1, c_1, \omega_2 \cdots \omega_k) \vdash_{\tau_2} \ldots \vdash_{\tau_k} (q_k, c_k, \varepsilon)$; we may then write $(q_0, c_0, \omega_1 \cdots \omega_k) \vdash_\theta (q_k, c_k, \varepsilon)$ or $(q_0, c_0) \vdash_\theta (q_k, c_k)$ or $[\![\theta]\!] = \omega_1 \cdots \omega_k$. The *set of valid runs in* $\mathcal{M}$, denoted by $\mathrm{R}^\mathrm{v}_\mathcal{M}$, contains exactly the runs $\theta \in \mathrm{R}_\mathcal{M}$ where $(q_\mathrm{i}, c_\mathrm{i}) \vdash_\theta (q, c)$ for some $q \in Q_\mathrm{f}$ and $c \in C$. For $\theta \in \mathrm{R}^\mathrm{v}_\mathcal{M}$ we say that $\mathcal{M}$ *recognises* $[\![\theta]\!]$. The *language of* $\mathcal{M}$ is $L(\mathcal{M}) = \{[\![\theta]\!] \mid \theta \in \mathrm{R}^\mathrm{v}_\mathcal{M}\}$.

3 Tree stack automata

Informally, a tree stack is a tree with a designated position in it. The root of the tree serves as *bottom-most symbol* and the leaves are *top-most symbols*. We allow the stack pointer to move *downward* (i.e. to the parent) and *upward* (i.e. to any child). We may *write* at any position except for the root. We may also *push* a symbol to any vacant child position of the current node. Formally, for an alphabet Γ, a *tree stack over* Γ is a tuple $(\xi[\varepsilon \mapsto @], \rho)$ where $\xi \in \mathrm{T}_\Gamma$, $@ \notin \Gamma$, and $\rho \in \mathrm{dom}(\xi) \cup \{\varepsilon\}$. The set of all tree stacks over Γ is denoted by $\mathrm{TS}(\Gamma)$. We define the following subsets (or predicates) of and partial functions on $\mathrm{TS}(\Gamma)$:

- $\mathrm{equals}(\gamma) = \{(\xi, \rho) \in \mathrm{TS}(\Gamma) \mid \xi(\rho) = \gamma\}$ for every $\gamma \in \Gamma$ and
- $\mathrm{bottom} = \{(\xi, \rho) \in \mathrm{TS}(\Gamma) \mid \rho = \varepsilon\}$.
- $\mathrm{id}\colon \mathrm{TS}(\Gamma) \to \mathrm{TS}(\Gamma)$ where $\mathrm{id}(\xi, \rho) = (\xi, \rho)$ for every $(\xi, \rho) \in \mathrm{TS}(\Gamma)$,
- $\mathrm{push}\colon \mathbb{N}_+ \to \Gamma \to \mathrm{TS}(\Gamma) \to \mathrm{TS}(\Gamma)$ where $\mathrm{push}_n(\gamma)(\xi, \rho) = (\xi[\rho n \mapsto \gamma], \rho n)$ for every $(\xi, \rho) \in \mathrm{TS}(\Gamma)$, $n \in \mathbb{N}_+$ with $\rho n \notin \mathrm{dom}(\xi)$, and $\gamma \in \Gamma$,
- $\mathrm{up}\colon \mathbb{N}_+ \to \mathrm{TS}(\Gamma) \to \mathrm{TS}(\Gamma)$ where $\mathrm{up}_n(\xi, \rho) = (\xi, \rho n)$ for every $(\xi, \rho) \in \mathrm{TS}(\Gamma)$ and $n \in \mathbb{N}_+$ with $\rho n \in \mathrm{dom}(\xi)$,
- $\mathrm{down}\colon \mathrm{TS}(\Gamma) \to \mathrm{TS}(\Gamma)$ where $\mathrm{down}(\xi, \rho n) = (\xi, \rho)$ for every $(\xi, \rho n) \in \mathrm{TS}(\Gamma)$ with $n \in \mathbb{N}_+$, and
- $\mathrm{set}\colon \Gamma \to \mathrm{TS}(\Gamma) \to \mathrm{TS}(\Gamma)$ where $\mathrm{set}(\gamma)(\xi, \rho) = (\xi[\rho \mapsto \gamma], \rho)$ for every $\gamma \in \Gamma$ and $(\xi, \rho) \in \mathrm{TS}(\Gamma)$ with $\rho \neq \varepsilon$.

$$
\begin{aligned}
\delta\colon \tau_1 &= (1, \mathrm{a}, \mathrm{TS}(\Gamma), \mathrm{push}_1(*), 1)\\
\tau_2 &= (1, \varepsilon, \mathrm{TS}(\Gamma), \mathrm{push}_1(\#), 2)\\
\tau_3 &= (2, \varepsilon, \mathrm{equals}(\#), \mathrm{down}, 2)\\
\tau_4 &= (2, \mathrm{b}, \mathrm{equals}(*), \mathrm{down}, 2)\\
\tau_5 &= (2, \varepsilon, \mathrm{bottom}, \mathrm{up}_1, 3)\\
\tau_6 &= (3, \mathrm{c}, \mathrm{equals}(*), \mathrm{up}_1, 3)\\
\tau_7 &= (3, \varepsilon, \mathrm{equals}(\#), \mathrm{down}, 4)\\
\tau_8 &= (4, \mathrm{d}, \mathrm{equals}(*), \mathrm{down}, 4)\\
\tau_9 &= (4, \varepsilon, \mathrm{bottom}, \mathrm{id}, 5)
\end{aligned}
$$

$$
\begin{aligned}
& (1, \{\underline{(\varepsilon, @)}\}, \mathrm{abcd})\\
\vdash_{\tau_1} & (1, \{(\varepsilon, @), \underline{(1, *)}\}, \mathrm{bcd})\\
\vdash_{\tau_2} & (2, \{(\varepsilon, @), (1, *), \underline{(11, \#)}\}, \mathrm{bcd})\\
\vdash_{\tau_3} & (2, \{(\varepsilon, @), \underline{(1, *)}, (11, \#)\}, \mathrm{bcd})\\
\vdash_{\tau_4} & (2, \{\underline{(\varepsilon, @)}, (1, *), (11, \#)\}, \mathrm{cd})\\
\vdash_{\tau_5} & (3, \{(\varepsilon, @), \underline{(1, *)}, (11, \#)\}, \mathrm{cd})\\
\vdash_{\tau_6} & (3, \{(\varepsilon, @), (1, *), \underline{(11, \#)}\}, \mathrm{d})\\
\vdash_{\tau_7} & (4, \{(\varepsilon, @), \underline{(1, *)}, (11, \#)\}, \mathrm{d})\\
\vdash_{\tau_8} & (4, \{\underline{(\varepsilon, @)}, (1, *), (11, \#)\}, \varepsilon)\\
\vdash_{\tau_9} & (5, \{\underline{(\varepsilon, @)}, (1, *), (11, \#)\}, \varepsilon)
\end{aligned}
$$

Figure 1: Set of transitions and a valid run in $\mathcal{M}$ (cf. Example 3.2).

We may denote a tree stack $(\xi, \rho) \in \mathrm{TS}(\Gamma)$ by writing ξ as a set and underlining the unique tuple of the form (ρ, γ) in this set. Consider for example a tree $\xi \in \mathrm{T}_{\{@,*,\#\}}$ with domain $\{\varepsilon, 2, 23\}$ such that $\xi\colon \varepsilon \mapsto @,\ 2 \mapsto *,\ 23 \mapsto \#$. We would then denote the tree stack $(\xi, 2) \in \mathrm{TS}(\{*, \#\})$ by $\{(\varepsilon, @), \underline{(2, *)}, (23, \#)\}$.

Definition 3.1. Let Γ be an alphabet. The *tree stack storage with respect to* Γ is the storage type $(\mathrm{TS}(\Gamma), P, F, \{\{\underline{(\varepsilon, @)}\}\})$, abbreviated by $\mathrm{TS}(\Gamma)$, where

$$
\begin{aligned}
P &= \{\mathrm{bottom}, \mathrm{equals}(\gamma), \mathrm{TS}(\Gamma) \mid \gamma \in \Gamma\} \text{ and}\\
F &= \{\mathrm{id}, \mathrm{push}_n(\gamma), \mathrm{up}_n, \mathrm{down}, \mathrm{set}(\gamma) \mid \gamma \in \Gamma, n \in \mathbb{N}\}.
\end{aligned}
$$
□

We call automata with tree stack storage *tree stack automata (short: TSA).* In a storage configuration (ξ, ρ) of a TSA $\mathcal{M}$ we call ξ the *stack (of* $\mathcal{M}$*)* and ρ the *stack pointer (of* $\mathcal{M}$*).*

Example 3.2. Let $\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}\}$ and $\Gamma = \{*, \#\}$. Consider the TSA

$$\mathcal{M} = ([5], \mathrm{TS}(\Gamma), \Sigma, 1, \{\underline{(\varepsilon, @)}\}, \delta, \{5\})$$

where δ is shown in Fig. 1. Figure 1 also shows the valid run $\tau_1\tau_2\tau_3\tau_4\tau_5\tau_6\tau_7\tau_8\tau_9$ in $\mathcal{M}$ recognising *abcd*. The language of $\mathcal{M}$ is $L(\mathcal{M}) = \{\mathrm{a}^n\mathrm{b}^n\mathrm{c}^n\mathrm{d}^n \mid n \in \mathbb{N}\}$ and thus not context-free. □

While $\mathcal{M}$ from the above example only uses a monadic stack, a TSA may also utilise branching as shown in the next example.

Example 3.3. Let again $\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}\}$ and $\Gamma = \{*, \#\}$. Consider the TSA

$$\mathcal{M}' = ([9], \mathrm{TS}(\Gamma), \Sigma, 1, \{\underline{(\varepsilon, @)}\}, \delta', \{9\})$$

$$
\begin{array}{ll}
 & \big(1, \{\underline{(\varepsilon, @)}\}, \mathrm{aabccd}\big) \\
\vdash_{\tau'_1} & \big(2, \{(\varepsilon, @), \underline{(1, *)}\}, \mathrm{abccd}\big) \\
\vdash_{\tau'_2} & \big(2, \{(\varepsilon, @), (1, *), \underline{(11, *)}\}, \mathrm{bccd}\big) \\
\vdash_{\tau'_3} & \big(3, \{(\varepsilon, @), (1, *), (11, *), \underline{(111, \#)}\}, \mathrm{bccd}\big) \\
\vdash_{\tau'_4\tau'_4\tau'_4} & \big(3, \{\underline{(\varepsilon, @)}, (1, *), (11, *), (111, \#)\}, \mathrm{bccd}\big) \\
\vdash_{\tau'_5} & \big(4, \{(\varepsilon, @), (1, *), (11, *), (111, \#), \underline{(2, *)}\}, \mathrm{ccd}\big) \\
\vdash_{\tau'_7} & \big(5, \{(\varepsilon, @), (1, *), (11, *), (111, \#), (2, *), \underline{(21, \#)}\}, \mathrm{ccd}\big) \\
\vdash_{\tau'_8\tau'_8} & \big(5, \{\underline{(\varepsilon, @)}, (1, *), (11, *), (111, \#), (2, *), (21, \#)\}, \mathrm{ccd}\big) \\
\vdash_{\tau'_9} & \big(6, \{(\varepsilon, @), \underline{(1, *)}, (11, *), (111, \#), (2, *), (21, \#)\}, \mathrm{ccd}\big) \\
\vdash_{\tau'_{10}\tau'_{10}} & \big(6, \{(\varepsilon, @), (1, *), (11, *), \underline{(111, \#)}, (2, *), (21, \#)\}, \mathrm{d}\big) \\
\vdash_{\tau'_{11}} & \big(7, \{(\varepsilon, @), (1, *), \underline{(11, *)}, (111, \#), (2, *), (21, \#)\}, \mathrm{d}\big) \\
\vdash_{\tau'_{12}\tau'_{12}} & \big(7, \{\underline{(\varepsilon, @)}, (1, *), (11, *), (111, \#), (2, *), (21, \#)\}, \mathrm{d}\big) \\
\vdash_{\tau'_{13}} & \big(8, \{(\varepsilon, @), (1, *), (11, *), (111, \#), \underline{(2, *)}, (21, \#)\}, \mathrm{d}\big) \\
\vdash_{\tau'_{14}} & \big(8, \{(\varepsilon, @), (1, *), (11, *), (111, \#), (2, *), \underline{(21, \#)}\}, \varepsilon\big) \\
\vdash_{\tau'_{15}} & \big(9, \{(\varepsilon, @), (1, *), (11, *), (111, \#), (2, *), \underline{(21, \#)}\}, \varepsilon\big)
\end{array}
$$

Figure 2: A valid run in $\mathcal{M}'$ (cf. Example 3.3).

with $\delta' = \{\tau'_1, \ldots, \tau'_{15}\}$ where

$$
\begin{aligned}
\tau'_1 &= \big(1, \mathrm{a}, \mathrm{bottom}, \mathrm{push}_1(*), 2\big), & \tau'_9 &= \big(5, \varepsilon, \mathrm{bottom}, \mathrm{up}_1, 6\big),\\
\tau'_2 &= \big(2, \mathrm{a}, \mathrm{TS}(\mathit{\Gamma}), \mathrm{push}_1(*), 2\big), & \tau'_{10} &= \big(6, \mathrm{c}, \mathrm{equals}(*), \mathrm{up}_1, 6\big),\\
\tau'_3 &= \big(2, \varepsilon, \mathrm{TS}(\mathit{\Gamma}), \mathrm{push}_1(\#), 3\big), & \tau'_{11} &= \big(6, \varepsilon, \mathrm{equals}(\#), \mathrm{down}, 7\big),\\
\tau'_4 &= \big(3, \varepsilon, \mathrm{TS}(\mathit{\Gamma}), \mathrm{down}, 3\big), & \tau'_{12} &= \big(7, \varepsilon, \mathrm{equals}(*), \mathrm{down}, 7\big),\\
\tau'_5 &= \big(3, \mathrm{b}, \mathrm{bottom}, \mathrm{push}_2(*), 4\big), & \tau'_{13} &= \big(7, \varepsilon, \mathrm{bottom}, \mathrm{up}_2, 8\big),\\
\tau'_6 &= \big(4, \mathrm{b}, \mathrm{TS}(\mathit{\Gamma}), \mathrm{push}_1(*), 4\big), & \tau'_{14} &= \big(8, \mathrm{d}, \mathrm{equals}(*), \mathrm{up}_1, 8\big), \text{ and}\\
\tau'_7 &= \big(4, \varepsilon, \mathrm{TS}(\mathit{\Gamma}), \mathrm{push}_1(\#), 5\big), & \tau'_{15} &= \big(8, \varepsilon, \mathrm{equals}(\#), \mathrm{id}, 9\big).\\
\tau'_8 &= \big(5, \varepsilon, \mathrm{TS}(\mathit{\Gamma}), \mathrm{down}, 5\big), & &
\end{aligned}
$$

Then $\mathcal{M}'$ recognises the languages $L' = \{\mathrm{a}^i \mathrm{b}^j \mathrm{c}^i \mathrm{d}^j \mid i, j \in \mathbb{N} \setminus \{0\}\}$. A valid run of $\mathcal{M}'$ on the word aabccd is shown in Fig. 2. □

3.1 Restricted TSA

Similar to Villemonte de la Clergerie [VdlC02b], we are interested in how often any specific position in the stack is reached from below. For every TSA $\mathcal{M}$ we define $(c_{\mathcal{M}}(\theta) \colon \mathbb{N}_+^* \to \mathbb{N}_+ \mid \theta \in R_{\mathcal{M}}^{\mathrm{v}})$ as the family of total functions where $c_{\mathcal{M}}(\varepsilon)(\rho) = 0$

for every $\rho \in \mathbb{N}_+^*$, and for every $\theta\tau \in R_{\mathcal{M}}^{\mathrm{v}}$ with $\tau \in \delta$ we have $c_{\mathcal{M}}(\theta\tau) = c_{\mathcal{M}}(\theta)$ if τ has neither a push- nor up-instruction, and we have $c_{\mathcal{M}}(\theta\tau) = c_{\mathcal{M}}(\theta)[\rho \mapsto c_{\mathcal{M}}(\theta)(\rho)+1]$ if τ has a push- or up-instruction and $\{\underline{(\varepsilon, @)}\} \vdash_{\theta\tau} (\xi, \rho)$ for some tree ξ. We call $\mathcal{M}$ *k-restricted* if $c_{\mathcal{M}}(\theta)(\rho) \leq k$ holds for every $\theta \in R_{\mathcal{M}}^{\mathrm{v}}$ and $\rho \in \mathbb{N}_+^*$. Note that $\mathcal{M}$ from Example 3.2 and $\mathcal{M}'$ from Example 3.3 are both 2-restricted.

Since (unrestricted) TSA can write at any position (except for ε) arbitrarily often, they can simulate Turing machines. It is apparent that 1-restricted TSA are exactly as powerful as pushdown automata. The power of k-restricted TSA for $k \geq 2$ is thus between the context-free and recursively enumerable languages.

3.2 Normal forms

We will see that loops that do not move the stack pointer as well as acceptance with non-ε stack pointers can be removed.

Let $\mathcal{M} = (Q, \mathrm{TS}(\Gamma), \Sigma, q_{\mathrm{i}}, \{\underline{(\varepsilon, @)}\}, \delta, Q_{\mathrm{f}})$ be a TSA. For each $q, q' \in Q$ and $\gamma, \gamma' \in \Gamma \cup \{@\}$ we define $R_{\mathcal{M}}(q, q')|_{\mathrm{stay}}^{\gamma \to \gamma'}$ as the set of runs θ in $\mathcal{M}$ such that θ only uses set- or id-instructions and there are tree stacks $(\xi, \rho), (\zeta, \rho) \in \mathrm{TS}(\Gamma)$ with $\xi(\rho) = \gamma$, $\zeta(\rho) = \gamma'$, and $(q, (\xi, \rho)) \vdash_\theta (q', (\zeta, \rho))$.

Definition 3.4. We call a TSA $\mathcal{M} = (Q, \mathrm{TS}(\Gamma), \Sigma, q_{\mathrm{i}}, \{\underline{(\varepsilon, @)}\}, \delta, Q_{\mathrm{f}})$ *cycle-free* if for every $q \in Q$ and $\gamma \in \Gamma \cup \{@\}$ we have $R_{\mathcal{M}}(q, q)|_{\mathrm{stay}}^{\gamma \to \gamma} = \{\varepsilon\}$. □

Lemma 3.5. For every (k-restricted) TSA $\mathcal{M}$, there is a (k-restricted) cycle-free TSA $\mathcal{M}'$ such that $L(\mathcal{M}) = L(\mathcal{M}')$.

Proof idea. Instead of performing all iterations of some loop $\theta \in R_{\mathcal{M}}(q, q)|_{\mathrm{stay}}^{\gamma \to \gamma} \setminus \{\varepsilon\}$ at the same position ρ in the stack, we insert additional push-instructions before each iteration of the loop. In order to find position ρ again after the desired number of iterations, we write symbols $*$ or $\#$ before every push, where a $*$ signifies that we have to perform at least two further down-instructions to reach ρ and $\#$ signifies that we will be at ρ after one more down-instruction. After returning to ρ, we enter a state $\tilde{q}$ that is equivalent to q except that it prevents us from entering the loop again.

Proof. Let $\mathcal{M} = (Q, \mathrm{TS}(\Gamma), \Sigma, q_{\mathrm{i}}, \{\underline{(\varepsilon, @)}\}, \delta, Q_{\mathrm{f}})$ be a TSA and

$$\tau_1 \cdots \tau_n = (q_0, \omega_1, p_1, f_1, q_1) \cdots (q_{n-1}, \omega_n, p_n, f_n, q_n)$$

be a shortest element of $R_{\mathcal{M}}(q, q)|_{\mathrm{stay}}^{\gamma \to \gamma} \setminus \{\varepsilon\}$ with $q_0 = q = q_n$.

Construct the automaton $\mathcal{M}' = (Q', \mathrm{TS}(\Gamma'), \Sigma, q_{\mathrm{i}}, \{\underline{(\varepsilon, @)}\}, \delta', Q_{\mathrm{f}}')$ where $Q' = Q \cup \{q_0', \ldots, q_{n-1}', q^\uparrow, q^\downarrow, \tilde{q}_0\}$, $q_0', \ldots, q_{n-1}', q^\uparrow, q^\downarrow, \tilde{q}_0$ are pairwise different and not in Q, $\Gamma' = \Gamma \cup \{*, \#\}$, $*$ and $\#$ are different and not in Γ, $Q_{\mathrm{f}}' = Q_{\mathrm{f}}$ if $q_0 \notin Q_{\mathrm{f}}$, $Q_{\mathrm{f}}' = Q_{\mathrm{f}} \cup \{\tilde{q}_0\}$ if $q_0 \in Q_{\mathrm{f}}$, δ' contains the transition $(\bar{q}, \omega, p', f, \hat{q})$ for every $(\bar{q}, \omega, p, f, \hat{q}) \in \delta \setminus \{\tau_n\}$ where $p' = \mathrm{TS}(\Gamma')$ if $p = \mathrm{TS}(\Gamma)$, and $p' = p$ otherwise, δ' contains the transition $(\tilde{q}_0, \omega, p', f, \hat{p})$ for every $(q_0, \omega, p, f, \hat{q}) \in \delta \setminus \{\tau_1\}$ where $p' = \mathrm{TS}(\Gamma')$ if $p = \mathrm{TS}(\Gamma)$, and $p' = p$ otherwise, and also δ' contains transitions

$$\begin{aligned}
\tilde{\tau}_n &= (q_{n-1}, \omega_n, p'_n, f_n, \tilde{q}_0) \\
\tau^{\uparrow} &= (q, \varepsilon, \mathrm{TS}(\Gamma'), \mathrm{push}_j(\#), q^{\uparrow}) , \\
\tau'_0 &= (q^{\uparrow}, \varepsilon, \mathrm{TS}(\Gamma'), \mathrm{push}_j(*), q'_0) , \\
\tau'_\kappa &= (q'_{\kappa-1}, \omega_\kappa, \mathrm{TS}(\Gamma'), \mathrm{id}, q'_\kappa) && \text{for every } \kappa \in [n-1], \\
\tau'_n &= (q'_{n-1}, \omega_n, \mathrm{TS}(\Gamma'), \mathrm{id}, q^{\uparrow}) , \\
\tau' &= (q^{\uparrow}, \varepsilon, \mathrm{TS}(\Gamma'), \mathrm{id}, q^{\downarrow}) , \\
\tau^{\downarrow}_{\mathrm{a}} &= (q^{\downarrow}, \varepsilon, \mathrm{equals}(*), \mathrm{down}, q^{\downarrow}) \text{ , and} \\
\tau^{\downarrow}_{\mathrm{b}} &= (q^{\downarrow}, \varepsilon, \mathrm{equals}(\#), \mathrm{down}, q)
\end{aligned}$$

where $p'_n = \mathrm{TS}(\Gamma')$ if $p_n = \mathrm{TS}(\Gamma)$ and $p'_n = p_n$ otherwise, and $j \in \mathbb{N}$ such that no push_j-instruction occurs in δ. By definition of the above transitions, we have $[\![(\tau_1 \cdots \tau_n)^\ell]\!] = [\![\tau^{\uparrow}(\tau'_0 \cdots \tau'_n)^\ell(\tau^{\downarrow}_{\mathrm{a}})^\ell \tau^{\downarrow}_{\mathrm{b}}]\!]$ for every $\ell \in \mathbb{N}$ and hence for every valid run θ in $\mathcal{M}$, there is a valid run θ' in $\mathcal{M}'$ with $[\![\theta]\!] = [\![\theta']\!]$. We iterate the above construction until the automaton is cycle-free. ■

Definition 3.6. We say that a TSA $\mathcal{M}$ is in *stack normal form* if the stack pointer of $\mathcal{M}$ is ε whenever we reach a final state. □

Lemma 3.7. For every (k-restricted) TSA $\mathcal{M}$, there is a (k-restricted) TSA $\mathcal{M}'$ in stack normal form such that $L(\mathcal{M}) = L(\mathcal{M}')$.

Proof idea. We introduce a new state q_{f} as the only final state and add transitions such that, beginning from any original final state, we may perform down-instructions until the predicate bottom is satisfied and then enter state q_{f}.

Proof. Let $\mathcal{M} = (Q, \mathrm{TS}(\Gamma), \Sigma, q_{\mathrm{i}}, \{\underline{(\varepsilon, @)}\}, \delta, Q_{\mathrm{f}})$ and $q_{\mathrm{down}}, q_{\mathrm{f}} \notin Q$. We construct an automaton $\mathcal{M}' = (Q \cup \{q_{\mathrm{down}}, q_{\mathrm{f}}\}, \mathrm{TS}(\Gamma), \Sigma, q_{\mathrm{i}}, \{\underline{(\varepsilon, @)}\}, \delta', \{q_{\mathrm{f}}\})$ where

$$\begin{aligned}
\delta' = \delta \cup \{(q, \varepsilon, \Gamma, \mathrm{id}, q_{\mathrm{down}}) \mid q \in Q_{\mathrm{f}}\} &\cup \{(q_{\mathrm{down}}, \varepsilon, \Gamma, \mathrm{down}, q_{\mathrm{down}})\} \\
&\cup \{(q_{\mathrm{down}}, \varepsilon, \mathrm{bottom}, \mathrm{id}, q_{\mathrm{f}})\} \ .
\end{aligned}$$

Since q_{f} is reachable from every element of Q_{f} and every storage configuration without reading additional symbols, we have that $L(\mathcal{M}) = L(\mathcal{M}')$. Also $\mathcal{M}'$ is in stack normal form since q_{f} can only be reached when the configuration satisfies the predicate bottom. This construction preserves k-restrictedness since $\delta' \setminus \delta$ can not reach states from Q and contains no additional push or up-instructions. ■

Note that $\mathcal{M}$ from Example 3.2 is cycle-free and in stack normal form whereas $\mathcal{M}'$ from Example 3.3 is cycle-free but not in stack normal form.

4 The equivalence of MCFG and restricted TSA

4.1 Every MCFG has an equivalent restricted TSA

The following construction applies the idea of Villemonte de la Clergerie [VdlC02b, Sec. 4] to the case of parallel multiple context-free grammars where, additionally, we have to deal with copying, deletion, and permutation of argument components. The overall idea is to incrementally guess for an input word w a derivation d of G (that accepts w) on the stack while traversing the relevant components of the composition functions on the right-hand sides of already guessed rules (in d) left-to-right. This specific traversal of the derivation tree is ensured using states and stack symbols that encode positions in the rules of G.[1]

Construction 4.1. Let $G = (N, \Sigma, I, R)$ be a PMCFG, $\Gamma = \{\square\} \cup R \cup \bar{R}$, and $\bar{R} = \big\{\langle r, i, j\rangle \mid r = A \to [u_1, \ldots, u_s](A_1, \ldots, A_\ell) \in R, i \in [s], j \in \{0, \ldots, |u_i|\}\big\}$. Intuitively, an element $\langle r, i, j\rangle \in \bar{R}$ stands for the position in r right after the j-th symbol of the i-th component. The *automaton with respect to G* is $\mathcal{M}(G) = (Q, \mathrm{TS}(\Gamma), \Sigma, \square, \{\underline{(\varepsilon, @)}\}, \{\square\}, \delta)$ where $Q = \{q, q_+, q_- \mid q \in \bar{R} \cup \{\square\}\}$ and δ is the smallest set such that for every $r = S \to [u](A_1, \ldots, A_\ell) \in R$ with $S \in I$, we have the transitions

$$\begin{aligned} \mathrm{init}(r) &= \big(\square, \varepsilon, \mathrm{TS}(\Gamma), \mathrm{push}_1(\square), \langle r, 1, 0\rangle\big)\ , \\ \mathrm{suspend}_1(r, 1, \square) &= \big(\langle r, 1, |u|\rangle, \varepsilon, \mathrm{equals}(\square), \mathrm{set}(r), \square_-\big)\ \text{, and} \\ \mathrm{suspend}_2(\square) &= \big(\square_-, \varepsilon, \mathrm{TS}(\Gamma), \mathrm{down}, \square\big)\ \text{ in } \delta; \end{aligned}$$

for every $r = A \to [u_1, \ldots, u_s](A_1, \ldots, A_\ell) \in R$, $i \in [s]$, $j \in [|u_i|]$ where $\sigma \in \Sigma$ is the j-th symbol in u_i, we have the transition

$$\mathrm{read}(r, i, j) = \big(\langle r, i, j-1\rangle, \sigma, \mathrm{TS}(\Gamma), \mathrm{id}, \langle r, i, j\rangle\big)\ \text{ in } \delta,$$

and for every $r = A \to [u_1, \ldots, u_s](A_1, \ldots, A_\ell) \in R$, $i \in [s]$, $j \in [|u_i|]$, $\kappa \in [\ell]$, $r' = A_\kappa \to [v_1, \ldots, v_{s'}](B_1, \ldots, B_{\ell'}) \in R$, $m \in [s']$ where $x_\kappa^m \in X$ is the j-th symbol in u_i, we have the transitions (abbreviating $\langle r, i, j\rangle$ by q)

$$\begin{aligned} \mathrm{call}(r, i, j, r') &= \big(\langle r, i, j-1\rangle, \varepsilon, \mathrm{TS}(\Gamma), \mathrm{push}_\kappa(q), \langle r', m, 0\rangle\big)\ , \\ \mathrm{resume}_1(r, i, j) &= \big(\langle r, i, j-1\rangle, \varepsilon, \mathrm{TS}(\Gamma), \mathrm{up}_\kappa, q_+\big)\ , \\ \mathrm{resume}_2(r, i, j, r') &= \big(q_+, \varepsilon, \mathrm{equals}(r'), \mathrm{set}(q), \langle r', m, 0\rangle\big)\ , \\ \mathrm{suspend}_1(r', m, q) &= \big(\langle r', m, |v_m|\rangle, \varepsilon, \mathrm{equals}(q), \mathrm{set}(r'), q_-\big)\ \text{, and} \\ \mathrm{suspend}_2(q) &= \big(q_-, \varepsilon, \mathrm{TS}(\Gamma), \mathrm{down}, q\big)\ \text{ in } \delta \end{aligned}$$

□

Let us abbreviate a run $\mathrm{suspend}_1(r', m, q)\,\mathrm{suspend}_2(q)$ by $\mathrm{suspend}(r', m, q)$ and a run $\mathrm{resume}_1(r, i, j)\,\mathrm{resume}_2(r, i, j, r')$ by $\mathrm{resume}(r, i, j, r')$.

[1] The control flow of our constructed automaton is similar to that of the treewalk evaluator for attribute grammars [KW76, Sec. 3]. The two major differences are that the treewalk evaluator also treats inherited attributes (which are not present in PMCFGs) and that our constructed automaton generates the tree on the fly (while the treewalk evaluator is already provided with the tree).

$$
\begin{array}{ll}
 & \big(\Box, \{\underline{(\varepsilon, @)}\}\big) \\
\vdash_{\mathrm{init}(r_1)} & \big(\langle r_1,1,0\rangle, \{(\varepsilon,@), \underline{(1,\Box)}\}\big) \\
\vdash_{\mathrm{call}(r_1,1,1,r_3)} & \big(\langle r_3,1,0\rangle, \{(\varepsilon,@), (1,\Box), \underline{(11,\langle r_1,1,1\rangle)}\}\big) \\
\vdash_{\mathrm{suspend}(r_3,1,\langle r_1,1,1\rangle)} & \big(\langle r_1,1,1\rangle, \{(\varepsilon,@), \underline{(1,\Box)}, (11,r_3)\}\big) \\
\vdash_{\substack{\mathrm{call}(r_1,1,2,r_4)\\ \mathrm{read}(r_4,1,1)}} & \big(\langle r_4,1,1\rangle, \{(\varepsilon,@), (1,\Box), (11,r_3), \underline{(12,\langle r_1,1,2\rangle)}\}\big) \\
\vdash_{\mathrm{call}(r_4,1,2,r_5)} & \big(\langle r_5,1,0\rangle, \{(\varepsilon,@), (1,\Box), (11,r_3), (12,\langle r_1,1,2\rangle), \underline{(121,\langle r_4,1,2\rangle)}\}\big) \\
\vdash_{\mathrm{suspend}(r_5,1,\langle r_4,1,2\rangle)} & \big(\langle r_4,1,2\rangle, \{(\varepsilon,@), (1,\Box), (11,r_3), \underline{(12,\langle r_1,1,2\rangle)}, (121,r_5)\}\big) \\
\vdash_{\mathrm{suspend}(r_4,1,\langle r_1,1,2\rangle)} & \big(\langle r_1,1,2\rangle, \{(\varepsilon,@), \underline{(1,\Box)}, (11,r_3), (12,r_4), (121,r_5)\}\big) \\
\vdash_{\mathrm{resume}(r_1,1,3,r_3)} & \big(\langle r_3,2,0\rangle, \{(\varepsilon,@), (1,\Box), \underline{(11,\langle r_1,1,3\rangle)}, (12,r_4), (121,r_5)\}\big) \\
\vdash_{\mathrm{suspend}(r_3,2,\langle r_1,1,3\rangle)} & \big(\langle r_1,1,3\rangle, \{(\varepsilon,@), \underline{(1,\Box)}, (11,r_3), (12,r_4), (121,r_5)\}\big) \\
\vdash_{\substack{\mathrm{resume}(r_1,1,4,r_4)\\ \mathrm{read}(r_4,2,1)}} & \big(\langle r_4,2,1\rangle, \{(\varepsilon,@), (1,\Box), (11,r_3), \underline{(12,\langle r_1,1,4\rangle)}, (121,r_5)\}\big) \\
\vdash_{\mathrm{resume}(r_4,2,2,r_5)} & \big(\langle r_5,2,0\rangle, \{(\varepsilon,@), (1,\Box), (11,r_3), (12,\langle r_1,1,4\rangle), \underline{(121,\langle r_4,2,2\rangle)}\}\big) \\
\vdash_{\mathrm{suspend}(r_5,2,\langle r_4,2,2\rangle)} & \big(\langle r_4,2,2\rangle, \{(\varepsilon,@), (1,\Box), (11,r_3), \underline{(12,\langle r_1,1,4\rangle)}, (121,r_5)\}\big) \\
\vdash_{\mathrm{suspend}(r_4,2,\langle r_1,1,4\rangle)} & \big(\langle r_1,1,4\rangle, \{(\varepsilon,@), \underline{(1,\Box)}, (11,r_3), (12,r_4), (121,r_5)\}\big) \\
\vdash_{\mathrm{suspend}(r_1,1,\Box)} & \big(\Box, \{\underline{(\varepsilon,@)}, (1,r_1), (11,r_3), (12,r_4), (121,r_5)\}\big)
\end{array}
$$

Figure 3: Run of $\mathcal{M}(G)$ that recognises bd (cf. Example 4.2). The symbols b and d are read by $\mathrm{read}(r_4,1,1)$ and $\mathrm{read}(r_4,2,1)$, respectively, all other transitions in this run read ε.

Example 4.2. Consider the MCFG $G = (\{S,A,B\},\{a,b,c,d\},\{S\},R)$ where

$$
\begin{array}{llll}
R\colon & r_1 = S \to [x_1^1 x_2^1 x_1^2 x_2^2](A,B) & r_2 = A \to [\mathrm{a}x_1^1, \mathrm{c}x_1^2](A) & r_3 = A \to [\varepsilon,\varepsilon]() \\
 & & r_4 = B \to [\mathrm{b}x_1^1, \mathrm{d}x_1^2](B) & r_5 = B \to [\varepsilon,\varepsilon]()\,.
\end{array}
$$

Then $L(G) = \{\mathrm{a}^i\mathrm{b}^j\mathrm{c}^i\mathrm{d}^j \mid i,j \in \mathbb{N}\}$. Figure 3 shows that $\mathcal{M}(G)$ recognises bd. $\Box$

For the rest of Section 4.1, let $G = (N,\Sigma,I,R)$ and $\bar{R}$ be defined as in Construction 4.1.

Lemma 4.3. The TSA $\mathcal{M}(G)$ is k-restricted if G is a k-MCFG.

Proof. Let $G = (N,\Sigma,I,R)$. Consider some arbitrary position $\rho \in \mathbb{N}^*$ and number $\kappa \in \mathbb{N}$. Position $\rho\kappa$ can only be reached from below if the current stack pointer is at position ρ and if we either execute the transition $\mathrm{call}(r,i,j,r')$ or the transition $\mathrm{resume}_1(r,i,j)$ for some $r = A \to [u_1,\ldots,u_s](A_1,\ldots,A_\ell) \in R$, $r' = A_\kappa \to [v_1,\ldots,v_{s'}](B_1,\ldots,B_{\ell'}) \in R$,

$i \in [s]$, $j \in [|u_i|]$, and $m \in [s']$ with $(u_i)_j = x_\kappa^m$. For those transitions to be applicable, the automaton has to be in state $\langle r, i, j\rangle$. Therefore, there are exactly as many states from which we can reach position ρi as there are occurrences of elements of $\{x_\kappa^1, \dots, x_\kappa^{s'}\}$ in the string $u_1 \cdots u_s$. Since $[u_1, \dots, u_s]$ is linear, the number of such occurrences is smaller or equal to s' and (since G is a k-MCFG) also smaller or equal to k. It is easy to see that in the part of the run where the stack pointer is never below ρ, the states $\langle r, i, 1\rangle, \dots, \langle r, i, |u_i|\rangle$ occur in that order whenever the stack pointer is at ρ and, in particular, none of those states occur twice. Therefore, we have that $c_{\mathcal{M}(G)}(\theta)(\rho\kappa) \leq k$ for every run θ and, since ρ and κ were chosen arbitrarily, we have that for any non-empty position $\rho' \neq \varepsilon$ and every run θ holds $c_{\mathcal{M}(G)}(\theta)(\rho') \leq k$. Since the position ε can never be entered from below, we have that $\mathcal{M}(G)$ is k-restricted. ■

Lemma 4.4. $L(G) \subseteq L(\mathcal{M}(G))$.

Proof. For every $A \in N$ and every derivation $d = r(d_1, \dots, d_m) \in \mathrm{D}_G(A)$ where $\mathrm{sort}(r) = (s_1 \cdots s_m, s)$ and $r = A \to [u_1, \dots, u_s](B_1, \dots, B_m)$, we recursively construct a tuple $(\theta^1, \dots, \theta^s)$ of runs in $\mathcal{M}(G)$. For the derivations $d_1, \dots, d_m$ we already have the tuples $(\theta_1^1, \dots, \theta_1^{s_1}), \dots, (\theta_m^1, \dots, \theta_m^{s_m})$, respectively. For every $\kappa \in [s]$, let $u_\kappa = \omega_1 \cdots \omega_\ell$ where $\omega_1, \dots, \omega_\ell \in \Sigma \cup X$. We define $\theta_\kappa = \omega_1' \cdots \omega_\ell'$ as the run in $\mathcal{M}(G)$ such that for every $\kappa' \in [\ell]$, we have that $\omega_{\kappa'}' = \mathrm{read}(r, \kappa, \kappa')$ if $\omega_{\kappa'} \in \Sigma$, $\omega_{\kappa'}' = \mathrm{call}(r, \kappa, \kappa', r')\, \theta_i^1\, \mathrm{suspend}(r', 1, \langle r, \kappa, \kappa'\rangle)$ if $\omega_{\kappa'} = x_i^1$ for some $i \geq 1$, and $\omega_{\kappa'}' = \mathrm{resume}(r, \kappa, \kappa', r')\, \theta_i^j\, \mathrm{suspend}(r', j, \langle r, \kappa, \kappa'\rangle)$ if $\omega_{\kappa'} = x_i^j$ for some $i \geq 1$ and $j \geq 2$, where $r' = d_i(\varepsilon)$. We can prove by structural induction on d that $[\![d]\!] = ([\![\theta^1]\!], \dots, [\![\theta^s]\!])$. If $d \in \mathrm{D}_G^{\mathrm{c}}$, then s is 1 and hence the valid run $\mathrm{init}(r)\, \theta^1\, \mathrm{suspend}(r, 1, \square)$ recognises exactly $[\![d]\!]$. ■

Lemma 4.5. Let $\tau_1, \dots, \tau_n \in \delta$ with $\theta = \tau_1 \cdots \tau_n \in R_{\mathcal{M}(G)}$ and let $\rho \in \mathbb{N}_+^* \setminus \{\varepsilon\}$. There is a rule $\varphi_\theta(\rho)$ in G such that, during the run θ, the automaton $\mathcal{M}(G)$ is in some state $\langle \varphi_\theta(\rho), i, j\rangle \in \bar{R}$ whenever the stack pointer is at ρ.

Proof. The rule $\varphi_\theta(\rho)$ is selected when ρ is first reached (with call). Then whenever we enter ρ with resume, a previous $\mathrm{suspend}_1$ has stored $\varphi_\theta(\rho)$ at position ρ and resume_2 enforces the claimed property. The claimed property is preserved by read. And whenever we enter ρ with suspend, a previous call or resume_2 has stored an appropriate state in the stack and suspend merely jumps back to that state, observing the claimed property. ■

Examining the form of runs in $\mathcal{M}(G)$ (Construction 4.1) and using Lemma 4.5 we observe:

Lemma 4.6. Let $\tau, \tau' \in \delta$, $q, q', q'' \in Q$, $\xi, \xi', \xi'' \in \mathrm{TS}(\Gamma)$, $\rho \in \mathbb{N}_+^*$, $i \in \mathbb{N}_+$, and $\varphi_\theta(\rho i)$ be of the form $A \to [u_1, \dots, u_s](A_1, \dots, A_\ell)$. Then:

1. If $(q', (\xi', \rho)) \vdash_\tau (q, (\xi, \rho i))$ with $q \in \bar{R}$, then $q = \langle \varphi_\theta(\rho i), j, 0\rangle$ for some $j \in [s]$ and τ must be either an init- or call-transition.
2. If $(q'', (\xi'', \rho)) \vdash_\tau (q', (\xi', \rho i)) \vdash_{\tau'} (q, (\xi, \rho i))$ with $q' \in \{q_+ \mid q \in \bar{R}\}$, then $q = \langle \varphi_\theta(\rho i), j, 0\rangle$ for some $j \in [s]$, τ is a resume_1-transition, and τ' is a resume_2-transition.

3. If $(q, (\xi, \rho i)) \vdash_\tau (q', (\xi', \rho i)) \vdash_{\tau'} (q'', (\xi'', \rho))$, then $q = \langle \varphi_\theta(\rho i), j, |u_j| \rangle$ for some $j \in [s]$, τ is a suspend$_1$-transition, and τ' is a suspend$_2$-transition.

Proof. (for 1, 2, and 3) The first projection of q is $\varphi_\theta(\rho i)$ due to Lemma 4.5.

(for 1) We only move the stack pointer to a child position and simultaneously go to a state from the set $\bar{R}$ when making a init or a call transition. From the definition of init and call transitions we know that the third projection of q is 0.

(for 2) We only move the stack pointer to a child position and simultaneously go to a state from the set $\{q_+ \mid q \in \bar{R}\}$ when making a resume$_1$ transition. Every resume$_1$ transition is followed by a resume$_2$ transition. From the definition of resume$_2$ transitions we know that the third projection of q is 0.

(for 3) We only move the stack pointer to a parent position when making a suspend$_2$ transition. Every suspend$_2$ transition is preceded by a suspend$_1$ transition. From the definition of suspend$_1$ transitions we know that the third projection of q is $|u_j|$. ∎

Lemma 4.7. $L(G) \supseteq L(\mathcal{M}(G))$ if G only has productive non-terminals.

Proof. For every run $\theta \in R_{\mathcal{M}(G)}$ we define $\varphi'_\theta \colon \mathbb{N}^* \to R$ by $\varphi'_\theta(\rho) = \varphi_\theta(1\rho)$ for every $\rho \in \mathbb{N}_+^*$ with $1\rho \in \mathrm{dom}(\varphi_\theta)$ (cf. Lemma 4.5). Then φ'_θ is a tree. One could show for every $d \in \mathrm{D}_G$ with $d \supseteq \varphi'_\theta$ by structural induction on φ'_θ that for every $\rho \in \mathrm{dom}(\varphi'_\theta)$ and every maximal interval $[a, b]$ where $\rho_a, \dots, \rho_b$ have prefix ρ, we have $[\![\tau_a \cdots \tau_b]\!] = [\![d|_\rho]\!]_m$ with $q_a = \langle \varphi'_\theta(\rho), m, 0 \rangle$ for some $m \in \mathbb{N}_+$. Let us call this property (†). Let $\tau_1, \dots, \tau_n \in \delta$ with $\theta = \tau_1 \cdots \tau_n \in R^{\mathrm{v}}_{\mathcal{M}(G)}$. Consider the run $(\Box, (@, \varepsilon)) \vdash_{\tau_1} (q_1, (\xi_1, 1\rho_1)) \vdash_{\tau_2} \dots \vdash_{\tau_{n-1}} (q_{n-1}, (\xi_{n-1}, 1\rho_{n-1})) \vdash_{\tau_n} (\Box, (\xi_n, \varepsilon))$. By (†) we obtain that $[\![\tau_2 \cdots \tau_{n-1}]\!] = [\![d]\!]$. By Lemma 4.6 and the fact that only an init-transition may start from $\Box$ we obtain that τ_1 is an init-transition and τ_n is a suspend$_2$-transition. Thus $[\![\tau_1]\!] = \varepsilon = [\![\tau_n]\!]$ and therefore $[\![\theta]\!] = [\![d]\!]$.

It remains to proof (†). For this we will denote the i-th component of the tuple generated by a derivation d as $[\![d]\!]_i$. Let $\varphi'_\theta(\rho) = A \to [u_1, \dots, u_s](A_1, \dots, A_\ell)$. Consider the run $(q_{a-1}, (\xi_{a-1}, 1\rho_{a-1})) \vdash_{\tau_a} (q_a, (\xi_a, 1\rho_a)) \vdash_{\tau_{a+1}} \dots \vdash_{\tau_b} (q_b, (\xi_b, 1\rho_b))$. Since $[a, b]$ is maximal and a transition can add at most one symbol to the stack pointer, we know that $\rho_a = \rho = \rho_b$. By Lemma 4.6 we also know that $q_a = \langle \varphi'_\theta(\rho), m, 0 \rangle$ and $q_b = \langle \varphi'_\theta(\rho), m, |u_m| \rangle$ for some $m \in [s]$. We now define the strings $w_1, \dots, w_{|u_m|}$ for every $i \in [|u_m|]$

1. as $w_i = \sigma$ if $(u_m)_i = \sigma$ for some $\sigma \in \Sigma$ and
2. as $w_i = [\![d|_{\rho\kappa}]\!]_j$ if $(u_m)_i = x^j_\kappa$ for some $x^j_\kappa \in X$.

From Construction 4.1 we know that if we are in Case 1, then the only possibility to continue from state $\langle \varphi'_\theta(\rho), m, i-1 \rangle$ is to use the transition $\mathrm{read}(\varphi'_\theta(\rho), m, i)$. We then end up in state $\langle \varphi'_\theta(\rho), m, i \rangle$. If we are in Case 2 and in state $\langle \varphi'_\theta(\rho), m, i-1 \rangle$, then, due to Construction 4.1 and Lemma 4.5, we can only continue with either (depending on the current stack) $\mathrm{call}(\varphi'_\theta(\rho), m, i, \varphi'_\theta(\rho\kappa))$ or $\mathrm{resume}(\varphi'_\theta(\rho), m, i, \varphi'_\theta(\rho\kappa))$. By induction hypothesis we know that after executing either of the above runs, the automaton will recognise $[\![d|_{\rho\kappa}]\!]_j = w_i$ and set the stack pointer to ρ. Then (by Construction 4.1) the

$$
\begin{array}{rl}
\delta\colon \tau_1 = & \big(1, \mathrm{a}, \mathrm{TS}(\Gamma), \mathrm{push}_1(*), 1\big) \\
\tau_2 = & \big(1, \varepsilon, \mathrm{TS}(\Gamma), \mathrm{push}_1(\#), 2\big) \\
\tau_3 = & \big(2, \varepsilon, \mathrm{equals}(\#), \mathrm{down}, 2\big) \\
\tau_4 = & \big(2, \mathrm{b}, \mathrm{equals}(*), \mathrm{down}, 2\big) \\
\tau_5 = & \big(2, \varepsilon, \mathrm{bottom}, \mathrm{up}_1, 3\big) \\
\tau_6 = & \big(3, \mathrm{c}, \mathrm{equals}(*), \mathrm{up}_1, 3\big) \\
\tau_7 = & \big(3, \varepsilon, \mathrm{equals}(\#), \mathrm{down}, 4\big) \\
\tau_8 = & \big(4, \mathrm{d}, \mathrm{equals}(*), \mathrm{down}, 4\big) \\
\tau_9 = & \big(4, \varepsilon, \mathrm{bottom}, \mathrm{id}, 5\big)
\end{array}
\qquad
\begin{array}{rl}
 & \big(1, \{\underline{(\varepsilon, @)}\}, \mathrm{abcd}\big) \\
\vdash_{\tau_1} & \big(1, \{(\varepsilon, @), \underline{(1, *)}\}, \mathrm{bcd}\big) \\
\vdash_{\tau_2} & \big(2, \{(\varepsilon, @), (1, *), \underline{(11, \#)}\}, \mathrm{bcd}\big) \\
\vdash_{\tau_3} & \big(2, \{(\varepsilon, @), \underline{(1, *)}, (11, \#)\}, \mathrm{bcd}\big) \\
\vdash_{\tau_4} & \big(2, \{\underline{(\varepsilon, @)}, (1, *), (11, \#)\}, \mathrm{cd}\big) \\
\vdash_{\tau_5} & \big(3, \{(\varepsilon, @), \underline{(1, *)}, (11, \#)\}, \mathrm{cd}\big) \\
\vdash_{\tau_6} & \big(3, \{(\varepsilon, @), (1, *), \underline{(11, \#)}\}, \mathrm{d}\big) \\
\vdash_{\tau_7} & \big(4, \{(\varepsilon, @), \underline{(1, *)}, (11, \#)\}, \mathrm{d}\big) \\
\vdash_{\tau_8} & \big(4, \{\underline{(\varepsilon, @)}, (1, *), (11, \#)\}, \varepsilon\big) \\
\vdash_{\tau_9} & \big(5, \{\underline{(\varepsilon, @)}, (1, *), (11, \#)\}, \varepsilon\big)
\end{array}
$$

Figure 1: Set of transitions and a valid run in $\mathcal{M}$, cf. Example 3.2, repeated from page 5.

automaton is in state $\langle \varphi'_\theta(\rho), m, i+1\rangle$. Repeating the step above eventually brings $\mathcal{M}(G)$ to the state $\langle \varphi'_\theta(\rho), m, |u_m|\rangle$ where only some suspend-transition is applicable. Thus $[\![\tau_a \cdots \tau_b]\!] = w_1 \cdots w_{|u_m|} = [\![d|_\rho]\!]_m$. ■

Proposition 4.8. $L(G) = L(\mathcal{M}(G))$ if G only has productive non-terminals.

Proof. The claim follows directly from Lemmas 4.4 and 4.7. ■

$\mathcal{M}(G)$ **is almost a parser for** G. Let (ξ, ε) be a storage configuration of $\mathcal{M}(G)$ after recognising some word w and let $\xi|_1$ be the first subtree of ξ, defined by the equation $\xi|_1(\rho) = \xi(1\rho)$. Then every complete derivation d in G with $\xi|_1 \subseteq d$ generates w. If G only contains rules with non-deleting composition functions, we even have that $\xi|_1$ *is* a derivation in G generating w. In Fig. 3, for example, we see that $r_1(r_3, r_4(r_5))$ is a derivation of bd in G (cf. Example 4.2).

4.2 Every restricted TSA has an equivalent MCFG

We construct an MCFG $G'(\mathcal{M})$ that recognises the valid runs of a given automaton $\mathcal{M}$, and then use the closure of MCFGs under homomorphisms. A tuple of runs $(\theta_1, \ldots, \theta_m)$ can be derived from non-terminal $\langle q_1, q'_1, \ldots, q_m, q'_m; \gamma_0, \ldots, \gamma_m\rangle$ iff the runs $\theta_1, \ldots, \theta_m$ all return to the stack position they started from and never go below it, and θ_i starts from state q_i and stack symbol γ_{i-1} and ends with q'_i and γ_i for every $i \in [m]$. We start with an example.

Example 4.9. Recall the TSA $\mathcal{M}$ from Example 3.2 (also cf. Fig. 1). Note that $\mathcal{M}$ is cycle-free and in stack normal form. Let us consider position ε of the stack. The only transitions applicable there are τ_1, τ_2, τ_5, and τ_9. Clearly, every valid run in $\mathcal{M}$ starts with τ_1 or τ_2 and ends with τ_9, every τ_5 must be preceded by τ_4 or τ_3, and every τ_9 must be preceded by τ_8 or τ_7. Thus each valid run in $\mathcal{M}$ is either of the form $\theta = \tau_1\theta_1\tau_4\tau_5\theta_2\tau_8\tau_9$ or $\theta' = \tau_2\theta'_1\tau_3\tau_5\theta'_2\tau_7\tau_9$ for some runs θ_1, θ_2, θ'_1, and θ'_2. The target state of τ_1 is 1 and the source state of τ_4 is 2. Also τ_1 pushes a $*$ to position 1 and the predicate of τ_4 accepts only $*$. Thus θ_1 must go from state 1 to 2 and from stack symbol $*$ to $*$ at position 1. Similarly, we obtain that θ_2, θ'_1, and θ'_2 go from state 3 to 4, 2 to 2, and 3 to 3, respectively, and from stack symbol $*$ to $*$, $\#$ to $\#$, and $\#$ to $\#$, respectively, at position 1. The runs θ_1 and θ_2 are linked since they are both executed while the stack pointer is in the first subtree of the stack; the same holds for θ'_1 and θ'_2.

Clearly, linked runs need to be produced by the same non-terminal. For the pair (θ_1, θ_2) of linked runs, we have the non-terminal $\langle 1, 2, 3, 4; *, *, * \rangle$ and for (θ'_1, θ'_2) we have $\langle 2, 2, 3, 3; \#, \#, \# \rangle$. Since θ and θ' go from state 1 to 5 and from storage symbol @ to @, we have the rules

$$\begin{aligned}&\langle 1, 5; @, @\rangle \to \big[\tau_1 x_1^1 \tau_4 \tau_5 x_1^2 \tau_8 \tau_9\big]\big(\langle 1, 2, 3, 4; *, *, *\rangle\big) \text{ and}\\ &\langle 1, 5; @, @\rangle \to \big[\tau_2 x_1^1 \tau_3 \tau_5 x_1^2 \tau_7 \tau_9\big]\big(\langle 2, 2, 3, 3; \#, \#, \#\rangle\big) \text{ in } G'(\mathcal{M}).\end{aligned}$$

Next, we explore the non-terminal $\langle 1, 2, 3, 4; *, *, * \rangle$, i.e. we need a run that goes from state 1 to 2 and from storage symbol $*$ to $*$ and another run that goes from state 3 to 4 and from storage symbol $*$ to $*$. There are only two kinds of suitable pairs of runs: $\big(\tau_1\theta_1\tau_4, \tau_6\theta_2\tau_8\big)$ and $\big(\tau_2\theta'_1\tau_3, \tau_6\theta'_2\tau_7\big)$ for some runs θ_1, θ_2, θ'_1, and θ'_2. The runs θ_1, θ_2, θ'_1, and θ'_2 of this paragraph then have the same state and storage behaviour as in the previous paragraph and we have rules

$$\begin{aligned}&\langle 1, 2, 3, 4; *, *, *\rangle \to \big[\tau_1 x_1^1 \tau_4, \tau_6 x_1^2 \tau_8\big]\big(\langle 1, 2, 3, 4; *, *, *\rangle\big) \text{ and}\\ &\langle 1, 2, 3, 4; *, *, *\rangle \to \big[\tau_2 x_1^1 \tau_3, \tau_6 x_1^2 \tau_7\big]\big(\langle 2, 2, 3, 3; \#, \#, \#\rangle\big) \text{ in } G'(\mathcal{M}).\end{aligned}$$

For non-terminal $\langle 2, 2, 3, 3; \#, \#, \# \rangle$, we may only take the pair of empty runs and thus have the rule $\langle 2, 2, 3, 3; \#, \#, \# \rangle \to \big[\varepsilon, \varepsilon\big]\big(\big)$ in $G'(\mathcal{M})$. □

For all $q, q' \in Q$, $\gamma, \gamma' \in \Gamma$, and $j \in \mathbb{N}_+$ we define the following sets:

$$\begin{aligned}\delta(q, q')|_{\mathrm{up}_j}^{\gamma\nearrow\bullet} &= \{(q, \omega, p, \mathrm{up}_j, q') \in \delta \mid \gamma \in p\}\,,\\ \delta(q, q')|_{\mathrm{push}_j}^{\gamma\nearrow\gamma'} &= \{(q, \omega, p, \mathrm{push}_j(\gamma'), q') \in \delta \mid \gamma \in p\}\,, \text{ and}\\ \delta(q, q')|_{\mathrm{down}}^{\gamma\searrow\bullet} &= \{(q, \omega, p, \mathrm{down}, q') \in \delta \mid \gamma \in p\}\,.\end{aligned}$$

For every $q, q' \in Q$, $\gamma, \gamma' \in \Gamma \cup \{@\}$, $\beta, \beta' \in \Gamma$, and $j \in \mathbb{N}_+$ we distinguish the following groups of runs (to help the intuition, they are visualised in Fig. 5):

1. A sequence of id- or set-instructions followed by an up- or push-instruction:

$$\Omega^{\uparrow}_{\mathcal{M}}(q, q'; \gamma, \gamma'; j, \beta) = \bigcup\nolimits_{\bar{q}\in Q} R_{\mathcal{M}}(q, \bar{q})|_{\mathrm{stay}}^{\gamma\to\gamma'} \cdot \big(\delta(\bar{q}, q')|_{\mathrm{push}_j}^{\gamma'\nearrow\beta} \cup \delta(\bar{q}, q')|_{\mathrm{up}_j}^{\gamma'\nearrow\bullet}\big)$$

2. A down-instruction followed by id- or set-instructions:

$$\Omega^{\downarrow}_{\mathcal{M}}(q, q'; \gamma, \gamma'; \beta') = \bigcup\nolimits_{\bar{q}\in Q} \delta(q, \bar{q})|_{\mathrm{down}}^{\beta'\searrow\bullet} \cdot R_{\mathcal{M}}(\bar{q}, q')|_{\mathrm{stay}}^{\gamma\to\gamma'}$$

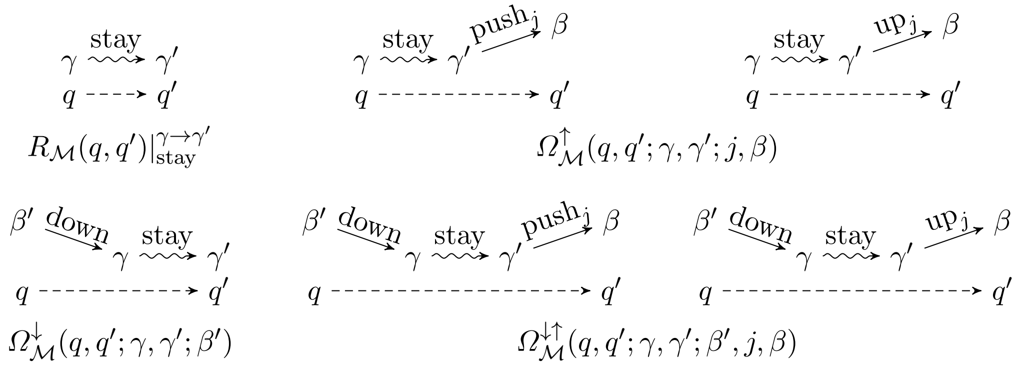

Figure 5: Groups of runs in $\mathcal{M}$ where dashed arrows signify the change of states and continuous arrows signify the change in the storage.

3. A down-instruction, then a sequence of id- or set-instructions and finally an up- or push-instruction:

$$\Omega^{\downarrow\uparrow}_{\mathcal{M}}(q,q';\gamma,\gamma';\beta',j,\beta) = \bigcup_{\bar{q}\in Q} \delta(q,\bar{q})|^{\beta'\searrow\bullet}_{\text{down}} \cdot \Omega^{\uparrow}_{\mathcal{M}}(\bar{q},q';\gamma,\gamma';j,\beta)$$

The arguments of $\Omega^{\uparrow}_{\mathcal{M}}$, $\Omega^{\downarrow}_{\mathcal{M}}$, and $\Omega^{\downarrow\uparrow}_{\mathcal{M}}$ are grouped using semicolons. The first group describes the state behaviour of the run; the second group describes the storage behaviour at the parent position (i.e. the position of the set- and id-instructions), and the third group describes the storage behaviour at the child positions (i.e. the positions immediately above the parent position).

We build tuples of runs from the three groups above by matching the storage behaviour of neighbouring runs at the parent position. A tuple $t = (\theta_0, \ldots, \theta_\ell)$ of runs is *admissible* if $\ell = 0$ and θ_0 only uses id- and set-instructions; or if $\ell \geq 1$, θ_0 is in group 1, θ_ℓ is in group 2, and for every $i \in [\ell]$, we have

$$\theta_{i-1} \in \Omega^{\uparrow}_{\mathcal{M}}(q,q';\gamma,\bar{\gamma};j,\beta) \cup \Omega^{\downarrow\uparrow}_{\mathcal{M}}(q,q';\gamma,\bar{\gamma};\beta',j,\beta) \qquad \text{and}$$
$$\theta_i \in \Omega^{\downarrow}_{\mathcal{M}}(q'',q''';\bar{\gamma},\gamma';\beta''') \cup \Omega^{\downarrow\uparrow}_{\mathcal{M}}(q'',q''';\bar{\gamma},\gamma';\beta''',j',\beta'')$$

for some $\gamma, \bar{\gamma}, \gamma' \in \Gamma \cup \{@\}$, $\beta, \beta', \beta'', \beta''' \in \Gamma$, $q, q', q'', q''' \in Q$, and $j, j' \in \mathbb{N}_+$. Note that only the $\bar{\gamma}$ has to match. Then $\theta_{i-1}\theta_i$ may *not* be a run in $\mathcal{M}$ since it is not guaranteed that $q' = q''$ and $\beta = \beta'$. We therefore say that there is a $(q', q''; j, \beta, \beta')$*-gap between* θ_{i-1} *and* θ_i. Let $q_1, q_2 \in Q$ and $\gamma_1, \gamma_2 \in \Gamma \cup \{@\}$. We say that t *has type* $\langle q_1, q_2; \gamma_1, \gamma_2 \rangle$ if $\ell = 0$ and $\theta_0 \in R_{\mathcal{M}}(q_1, q_2)|^{\gamma_1\to\gamma_2}_{\text{stay}}$; or if $\ell \geq 1$, the first transition in θ_0 has source state q_1 and its predicate contains γ_1, the last transition of θ_ℓ has target state q_2, the last set-instruction occurring in t, if there is one, is $\text{set}(\gamma_2)$, and $\gamma_1 = \gamma_2$ if no set-instruction occurs in t. The set of admissible tuples in $\Omega^{*}_{\mathcal{M}}$ is denoted by $\Omega^{\star}_{\mathcal{M}}$. We define $t[y_1, \ldots, y_\ell] = \theta_0 y_1 \theta_1 \cdots y_\ell \theta_\ell$ for every $y_1, \ldots, y_\ell \in X$ to later fill the gaps with variables.

Let $T = (t_1, \ldots, t_s) \in (\Omega^{\star}_{\mathcal{M}})^*$ and $\ell_1, \ldots, \ell_s$ be the counts of gaps in $t_1, \ldots, t_s$, respectively. For every $i \in [s]$ and $\kappa \in [\ell_i]$ we set $q_{(i,\kappa)}, q'_{(i,\kappa)} \in Q$, $\beta_{(i,\kappa)}, \beta'_{(i,\kappa)} \in \Gamma$, and

$j_{(i,\kappa)} \in \mathbb{N}_+$ such that the κ-th gap in t_i is a $(q_{(i,\kappa)}, q'_{(i,\kappa)}; j_{(i,\kappa)}, \beta_{(i,\kappa)}, \beta'_{(i,\kappa)})$-gap. Let $\varphi_T, \psi_T \colon \mathbb{N}_+ \times \mathbb{N}_+ \to \mathbb{N}_+$ and $\pi_T \colon \mathbb{N}_+ \times \mathbb{N}_+ \to \mathbb{N}_+ \times \mathbb{N}_+$ be partial functions such that for every $i \in [s]$ and $\kappa \in [\ell_i]$, the number $j_{(i,\kappa)}$ is the $\varphi_T(i,\kappa)$-th distinct number occurring in $J = j_{(1,1)} \cdots j_{(1,|t_1|)} \cdots j_{(s,1)} \cdots j_{(s,|t_s|)}$ when read left-to-right, $j_{(i,\kappa)}$ occurs for the $\psi_T(i,\kappa)$-th time at the element with index (i,κ) in J, and $\pi_T(i,\kappa) = (\varphi_T(i,\kappa), \psi_T(i,\kappa))$. Moreover let m be the count of distinct numbers in J. We call T *admissible* if

- the κ-th run in t_i ends with a push-instruction whenever $\varphi_T(i,\kappa) = 1$,
- $\beta'_{\pi_T^{-1}(\kappa',\kappa)} = \beta_{\pi_T^{-1}(\kappa',\kappa+1)}$ for every $\kappa' \in [m]$ and $\kappa \in [\ell_{\kappa'} - 1]$, and
- there are $q_1, \bar{q}_1, \ldots, q_s, \bar{q}_s \in Q$ and $\gamma_0, \ldots, \gamma_s \in \Gamma \cup \{@\}$ such that for every $\kappa \in [s]$, we have that t_κ is of type $\langle q_\kappa, \bar{q}_\kappa; \gamma_{\kappa-1}, \gamma_\kappa \rangle$.

We then say that T *has* $(A; B_1, \ldots, B_m)$, denoted by $\mathrm{type}(T) = (A; B_1, \ldots, B_m)$, where $A = \langle q_1, \bar{q}_1, \ldots, q_s, \bar{q}_s; \gamma_0, \ldots, \gamma_s \rangle$, and for every $\kappa' \in [m]$:

$$B_{\kappa'} = \langle q_{\pi_T^{-1}(\kappa',1)}, q'_{\pi_T^{-1}(\kappa',1)}, \ldots, q_{\pi_T^{-1}(\kappa',\ell_{\kappa'})}, q'_{\pi_T^{-1}(\kappa',\ell_{\kappa'})}; \\ \beta_{\pi_T^{-1}(\kappa',1)}, \beta'_{\pi_T^{-1}(\kappa',1)}, \ldots, \beta'_{\pi_T^{-1}(\kappa',\ell_{\kappa'})} \rangle \,.$$

The set of admissible elements of $(\Omega^{\star}_{\mathcal{M}})^*$ is denoted by $\Omega^{\star\star}_{\mathcal{M}}$.

Construction 4.10. Let $\mathcal{M} = (Q, \mathrm{TS}(\Gamma), \Sigma, q_\mathrm{i}, \{(\varepsilon, @)\}, \delta, Q_\mathrm{f})$ be a cycle-free k-restricted TSA in stack normal form. Define the k-MCFG $G'(\mathcal{M}) = (N, \Sigma, I, R')$ where $N = \{A, B_1, \ldots, B_m \mid \langle A; B_1, \ldots, B_m \rangle \in \mathrm{type}(\Omega^{\star\star}_{\mathcal{M}})\}$, $I = \{\langle q_\mathrm{i}, q; @, @ \rangle \mid q \in Q_\mathrm{f}\}$, and R' contains for every $T = (t_1, \ldots, t_s) \in \Omega^{\star\star}_{\mathcal{M}}$ the rule $A \to \big[u_1, \ldots, u_s\big](B_1, \ldots, B_m)$ where $(A; B_1, \ldots, B_m)$ is the type of T and $u_\kappa = t_\kappa[x^{\psi_T(\kappa,1)}_{\varphi_T(\kappa,1)}, \ldots, x^{\psi_T(\kappa,\ell_\kappa)}_{\varphi_T(\kappa,\ell_\kappa)}]$ for every $\kappa \in [s]$. Let $G(\mathcal{M})$ be a k-MCFG recognising $\{[\![\theta]\!] \mid \theta \in L(G'(\mathcal{M}))\}$.[2] □

Proposition 4.11. $L(\mathcal{M}) = L(G(\mathcal{M}))$ for every cycle-free k-restricted TSA $\mathcal{M}$ in stack normal form.

Proof. We can show by induction that $G'(\mathcal{M})$ generates exactly the valid runs of $\mathcal{M}$. Our claim then follows from the definition of language of $G(\mathcal{M})$. ■

4.3 The main theorem

Theorem 4.12. Let $L \subseteq \Sigma^*$ and $k \in \mathbb{N}_+$. The following are equivalent:

1. There is a k-MCFG G with $L = L(G)$.
2. There is a k-restricted tree stack automaton $\mathcal{M}$ with $L = L(\mathcal{M})$.

Proof. We get the implication $(1 \implies 2)$ from Lemma 4.3 and Proposition 4.8 and the implication $(2 \implies 1)$ from Lemmas 3.5 and 3.7 and Proposition 4.11. ■

[2]The k-MCFG $G(\mathcal{M})$ exists since $[\![\cdot]\!]$ is a homomorphism and k-MCFLs are closed under homomorphisms [SMFK91, Thm. 3.9].

5 Conclusion

The automata characterisation of multiple context-free languages presented in this paper is achieved through tree stack automata that possess, in addition to the usual finite state control, the ability to manipulate a tree-shaped stack; tree stack automata are then restricted by bounding the number of times that the stack pointer enters any position of the stack from below (cf. Section 3). The proofs for the inclusions of multiple context-free languages in restricted tree stack languages and vice versa are both constructive; the former even works for parallel multiple context-free grammars, although the resulting automaton may then no longer be restricted (cf. Section 4). Theorem 4.12 closes a gap in formal language theory open since the introduction of MCFGs [SMFK91]. The proof allows for the easy implementation of a parser for parallel multiple context-free grammars.

References

[Aho69] A. V. Aho. Nested stack automata. *Journal of the ACM*, 16(3):383–406, 1969. DOI: `10.1145/321526.321529`.

[Cho62] N. Chomsky. Formal properties of grammars. In R. D. Luce, R. R. Bush, and E. Galanter, editors, *Handbook of Mathmatical Psychology*, volume 2. 1962.

[Eng14] J. Engelfriet. Context-free grammars with storage. *CoRR*, 2014. arXiv: `1408.0683`.

[Gue83] I. Guessarian. Pushdown tree automata. *Mathematical systems theory*, 16(1):237–263, 1983. DOI: `10.1007/BF01744582`.

[HV15] L. Herrmann and H. Vogler. A Chomsky-Schützenberger theorem for weighted automata with storage. In A. Maletti, editor, *Proceedings of the 6th International Conference on Algebraic Informatics (CAI 2015)*, volume 9270, pages 90–102. Springer International Publishing, 2015. DOI: `10.1007/978-3-319-23021-4_11`.

[KS09] M. Kuhlmann and G. Satta. Treebank grammar techniques for non-projective dependency parsing. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '09, pages 478–486, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics. URL: `http://dl.acm.org/citation.cfm?id=1609067.1609120`.

[KW76] K. Kennedy and S. K. Warren. Automatic generation of efficient evaluators for attribute grammars. *Proceedings of the 3rd ACM SIGACT-SIGPLAN symposium on Principles on programming languages (POPL'76)*, 1976. DOI: `10.1145/800168.811538`.

[Mai10] W. Maier. Direct Parsing of Discontinuous Constituents in German. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, SPMRL '10, pages 58–66, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL: `http://dl.acm.org/citation.cfm?id=1868771.1868778`.

[Sch63] M. P. Schützenberger. On context-free languages and push-down automata. *Information and control*, 6(3):246–264, 1963. DOI: `10.1016/S0019-9958(63)90306-1`.

[Sco67] D. Scott. Some definitional suggestions for automata theory. *Journal of Computer and System Sciences*, 1(2):187–212, 1967. DOI: `10.1016/s0022-0000(67)80014-x`.

[SMFK91] H. Seki, T. Matsumura, M. Fujii, and T. Kasami. On multiple context-free grammars. *Theoretical Computer Science*, 88(2):191–229, 1991. DOI: `10.1016/0304-3975(91)90374-B`.

[VdlC02a] È. Villemonte de la Clergerie. Parsing mcs languages with thread automata. In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+6)*, pages 101–108, 2002.

[VdlC02b] È. Villemonte de la Clergerie. Parsing mildly context-sensitive languages with thread automata. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING '02)*, volume 1, pages 1–7, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. DOI: `10.3115/1072228.1072256`.

[VS88] K. Vijay-Shanker. *A study of tree adjoining grammars*. PhD thesis, 1988.