Journal of Computer and System Sciences 82 (2016) 1329–1359

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

$LR(0)$ conjunctive grammars and deterministic synchronized alternating pushdown automata ☆

Tamar Aizikowitz, Michael Kaminski *

Department of Computer Science, Technion – Israel Institute of Technology, Haifa 32000, Israel

ARTICLE INFO

Article history:
Received 17 May 2014
Received in revised form 26 May 2016
Accepted 27 May 2016
Available online 14 June 2016

Keywords:
Conjunctive grammars
Synchronized alternating pushdown automata
$LR(0)$ conjunctive grammars
Deterministic synchronized alternating pushdown automata

ABSTRACT

The paper introduces a subfamily of synchronized alternating pushdown automata, *deterministic synchronized alternating pushdown automata*, and a subfamily of conjunctive grammars, $LR(0)$ *conjunctive grammars*. It is shown that deterministic synchronized alternating pushdown automata and $LR(0)$ conjunctive grammars have the same recognition/generation power, analogously to the classical equivalence between acceptance by empty stack of deterministic pushdown automata and $LR(0)$ grammars. These models form the theoretical basis for efficient *linear time* parsing of a subfamily of conjunctive languages which *properly* includes the classical $LR(0)$ languages.

© 2016 Elsevier Inc. All rights reserved.

1. Introduction

Context-free languages lay at the very foundations of Computer Science, proving to be one of the most appealing language classes for practical applications. On the one hand, they are quite expressive, covering such syntactic constructs as necessary, e.g., for mathematical expressions. On the other hand, they are polynomially parsable, which makes them practical for real world applications. However, research in certain fields has raised a need for computational models which extend context-free models, without loosing their computational efficiency.

Conjunctive grammars are an example of such a model. Introduced by Okhotin in [10],[1] conjunctive grammars are a generalization of context-free grammars which allow explicit conjunction operations in productions, thereby adding the power of intersection. Conjunctive grammars were shown by Okhotin to accept all finite intersections of context-free languages, as well as some additional languages. Okhotin proved the languages generated by these grammars to be polynomially parsable [10,11,13], making the model practical from a computational standpoint, and therefore, of interest for applications in various fields such as, e.g., programming languages.

Alternating automata models were first introduced by Chandra, Kozen and Stockmeyer in [4]. Alternating pushdown automata were further explored in [9] and shown to accept exactly the exponential time languages. As such, they are too strong to be a counterpart for conjunctive grammars. Synchronized alternating pushdown automata (SAPDA) introduced in [2,3], are a weakened version of alternating pushdown automata. In [2,3], SAPDA were proven to be equivalent[2] to conjunctive grammars. Thus, in particular, SAPDA accept intersections of context-free languages.

☆ The first version of this paper was presented at *the 6th International Computer Science Symposium in Russia – CSR 2011*.

* Corresponding author.
E-mail address: kaminski@cs.technion.ac.il (M. Kaminski).

[1] See [16] for a comprehensive list of references.

[2] We call two models equivalent if they accept/generate the same class of languages.

http://dx.doi.org/10.1016/j.jcss.2016.05.008
0022-0000/© 2016 Elsevier Inc. All rights reserved.

Deterministic context-free languages are a subfamily of context-free languages that can be accepted by a deterministic PDA. In [7], Knuth introduced the notion of $LR(k)$ grammars and proved their equivalence to deterministic PDA. Furthermore, Knuth proved that $LR(0)$ languages (those that can be parsed with no lookahead) are equivalent to deterministic PDA that accept by empty stack.

In [11], Okhotin presented an extension of Tomita's generalized LR parsing algorithm (see [17]) for conjunctive grammars. The algorithm utilizes non-deterministic LR parsing and works for all conjunctive grammars in polynomial time. When applied to deterministic context-free languages, the run-time is linear.

In fact, almost all of the main context-free parsing techniques generalize to conjunctive (and Boolean grammars, see [14]) without any increase in their computational complexity, see [16, Sections 5 and 6]. Namely, the linear-time recursive descent [6] is generalized in [12], the CKY algorithm [20] is generalized in [10], as we have already mentioned above, the generalized LR parsing algorithm [17] is generalized in [11], and parsing by matrix multiplication [18] is generalized in [15].

In this paper we introduce a subfamily of SAPDA, *deterministic* SAPDA (DSAPDA) and a subfamily of conjunctive grammars, $LR(0)$ *conjunctive grammars*. We prove that these subfamilies are equivalent, analogously to the classical case. Furthermore, we present an efficient implementation of DSAPDA which forms the basis of a deterministic linear time parsing algorithm for $LR(0)$ conjunctive languages. This class of languages *properly* contains the classical $LR(0)$ languages, thus expanding upon classical results.

The paper is organized as follows. In Section 2 we recall the definitions of conjunctive grammars and SAPDA. Then, in Section 3 we introduce DSAPDA and prove that their membership problem is decidable in linear time. In Section 4 we define *item* SAPDA and prove that it corresponds to rightmost derivations of conjunctive grammars. As an immediate consequence of that correspondence we obtain a DSAPDA based parser for $LR(0)$ conjunctive grammars which are also defined in that section. In Section 5 we construct an $LR(0)$ conjunctive grammar from a DSAPDA, proving that the two models are equivalent. Finally, Section 6 contains an example of a deterministic conjunctive language that is not a finite intersection of classical context-free languages. We end the paper with a short conclusion and some directions for future research.

2. Preliminaries

In this section we recall the definitions of conjunctive grammars from [10] and SAPDA, from [2,3].

2.1. Conjunctive grammars

Definition 1. A *conjunctive grammar* is a tuple $G = (V, \Sigma, P, S)$, where V and Σ are disjoint finite sets of variables and terminals, respectively, $S \in V$ is the designated start variable, and P is a finite set of productions of the form

$$A \to (\alpha_1 \,\&\, \cdots \,\&\, \alpha_k)$$

such that $A \in V$ and $\alpha_i \in (V \cup \Sigma)^*$, $i = 1, 2, \ldots, k$. We say that k is the *degree* of the production. If $k \geq 2$, the production is called *proper conjunctive* and A is called a *split* variable. Otherwise, i.e., if $k = 1$, the production is called *ordinary* and we just write $A \to \alpha_1$.

Definition 2. *Conjunctive formulas* over $V \cup \Sigma$ are defined as follows.

- All symbols of $V \cup \Sigma$, as well as ϵ, are conjunctive formulas.
- If $\mathcal{A}$ and $\mathcal{B}$ are conjunctive formulas, then $\mathcal{A}\mathcal{B}$ is a conjunctive formula.
- If $\mathcal{A}_1, \ldots, \mathcal{A}_k$ are conjunctive formulas, then $(\mathcal{A}_1 \,\&\, \cdots \,\&\, \mathcal{A}_k)$ is a conjunctive formula.

Throughout this paper we shall use the following notation: σ, τ, etc. denote terminals, u, w, y, etc. denote words of terminals, A, B, etc. denote variables, α, β, etc. denote words over $V \cup \Sigma$, and $\mathcal{A}$, $\mathcal{B}$, etc. denote conjunctive formulas. All the symbols above may also be indexed or primed.

Definition 3. For a conjunctive grammar $G = (V, \Sigma, P, S)$, the relation of *immediate derivability*, denoted $\Rightarrow_G$, on the set of conjunctive formulas is defined as follows.

(1) $s_1\ A\ s_2 \Rightarrow_G s_1(\alpha_1 \,\&\, \cdots \,\&\, \alpha_k)s_2$, for $A \to (\alpha_1 \,\&\, \cdots \,\&\, \alpha_k) \in P$ and
(2) $s_1\ (w \,\&\, \cdots \,\&\, w)\ s_2 \Rightarrow_G s_1\ w\ s_2$, for $w \in \Sigma^*$,

where $s_1, s_2 \in (V \cup \Sigma \cup \{(,), \&\})^*$. We refer to (1) and (2) as *production* and *contraction* steps, respectively. As usual, $\Rightarrow_G^+$ and $\Rightarrow_G^*$ are the transitive and the reflexive transitive closures of $\Rightarrow_G$, respectively.

The language $L(\mathcal{A})$ of a conjunctive formula $\mathcal{A}$ is

$$L(\mathcal{A}) = \{w \in \Sigma^* : \mathcal{A} \Rightarrow_G^* w\}$$

and the language $L(G)$ of a conjunctive grammar G is

$$L(G) = L(S) = \{w \in \Sigma^* : S \Rightarrow_G^* w\}.$$

Remark 4. Like in the classical case, applications of productions to different variables in a conjunctive formula are independent of each other and the order in which productions are applied is irrelevant. Therefore, in a conjunctive grammar derivation, a variable to which a production is applied may be chosen randomly, or may be chosen according to a specified order, e.g., the next variable is the first from the left to the "current" variable, the next variable comes form a different conjunct, if possible, etc.

Example 5. (Cf. [10, Example 1]) The following conjunctive grammar $G = (V, \Sigma, P, S)$ generates the non-context-free language $\{a^n b^n c^n : n = 1, 2, \ldots\}$, called the *multiple agreement* language. In this grammar,

- $V = \{S, S_a, S_{bc}, S_{ac}, S_b\}$,
- $\Sigma = \{a, b, c\}$, and
- P consists of the following productions.
  $S \to (S_a S_{bc} \ \& \ S_{ac})$
  $S_a \to aS_a \mid b$
  $S_{bc} \to bS_{bc}c \mid c$
  $S_{ac} \to aS_{ac}c \mid aS_b$
  $S_b \to bS_b \mid c$

The intuition lying behind the above productions is as follows.

$$L(S_a S_{bc}) = \{a^m b^n c^n : m, n = 1, 2, \ldots\},$$
$$L(S_{ac}) = \{a^m b^n c^m : m, n = 1, 2, \ldots\}.$$

Therefore,

$$L(G) = L(S_a S_{bc}) \cap L(S_{ac}) = \{a^n b^n c^n : n = 1, 2, \ldots\}.$$

For example, the word *aaabbbccc* can be derived as follows.

$$\begin{aligned} S &\Rightarrow (S_a S_{bc} \ \& \ S_{ac}) \Rightarrow (S_a b S_{bc} c \ \& \ S_{ac}) \Rightarrow (S_a bb S_{bc} cc \ \& \ S_{ac}) \Rightarrow (S_a bbccc \ \& \ S_{ac}) \\ &\Rightarrow (aS_a bbccc \ \& \ S_{ac}) \Rightarrow (aaS_a bbccc \ \& \ S_{ac}) \Rightarrow (aaaS_a bbccc \ \& \ S_{ac}) \\ &\Rightarrow (aaabbbccc \ \& \ S_{ac}) \Rightarrow (aaabbbccc \ \& \ aS_{ac}c) \Rightarrow (aaabbbccc \ \& \ aaS_{ac}cc) \\ &\Rightarrow (aaabbbccc \ \& \ aaaS_b cc) \Rightarrow (aaabbbccc \ \& \ aaabS_b cc) \Rightarrow (aaabbbccc \ \& \ aaabbS_b cc) \\ &\Rightarrow (aaabbbccc \ \& \ aaabbbS_b cc) \Rightarrow (aaabbbccc \ \& \ aaabbbccc) \Rightarrow aaabbbccc \end{aligned}$$

Note that S is the only split variable of G.

2.2. Synchronized alternating pushdown automata

Introduced in [2,3], SAPDA are a natural counterpart of conjunctive grammars. In an SAPDA, transitions are made to a conjunction of actions that are pairs of the form (`change state`, `replace the top stack symbol`). The model is non-deterministic, because several conjunctions may be possible from a given configuration. If all conjunctions are of one action, the automaton is an ordinary PDA.[3]

The stack memory of an SAPDA is a tree. Each leaf has a processing head which reads the input and writes to its branch independently. When a conjunctive transition is applied, the branch splits into multiple branches, one for each conjunct. When all sibling branches (i.e., those "growing" from the same node) empty, they must empty synchronously, i.e., after reading the same portion of the input and in the same state, after which the computation continues from the parent branch.

Definition 6. A *synchronized alternating pushdown automaton* is a tuple $\boldsymbol{A} = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$, where

- Q is a finite set of states,

[3] The above "conjunctive" description of alternating automata is standard in the field of formal verification, e.g., see [8]. Namely, conjunctions of more than one actions correspond to *universal* states of alternating automata introduced in [4] and conjunctions consisting of one action only correspond to *existential* states.

- Σ is the input alphabet,
- Γ is the stack alphabet,
- δ is a function that assigns to each element of $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ a finite subset of

$$\{(q_1, \alpha_1) \wedge \cdots \wedge (q_k, \alpha_k) : k = 1, 2, \ldots, (q_i, \alpha_i) \in Q \times \Gamma^*, i = 1, \ldots, k\},$$

- $q_0 \in Q$ is the initial state, and
- $\perp \in \Gamma$ is the initial stack symbol.

We describe the current stage of the automaton computation as a labeled tree that encodes the stack content, the current states of the branches, and the remaining input to be read for each branch. States and remaining inputs are saved in leaves only, as these encode the branches currently processed.

Definition 7. A *configuration* of an SAPDA is a labeled tree, not containing nodes of degree one. Each internal node is labeled with $\alpha \in \Gamma^*$ denoting the branch stack content, and each leaf is labeled with (q, w, α), where

- $q \in Q$ is the current state of the branch,
- $w \in \Sigma^*$ is the remaining input to be read by the branch, and
- $\alpha \in \Gamma^*$ is the branch stack content.

For a node v in a configuration C, we denote the label of v in C by $C(v)$. If a configuration is of one node only,[4] it is denoted by the label of that node. That is, if a configuration has a single node labeled with (q, w, α), then it is denoted by (q, w, α).

At each computation step, a transition is applied to one branch. If a branch empties, it cannot be chosen for the next transition (because it has no top stack symbol). If all sibling branches are empty, and each of them emptied with the *same* remaining input (i.e., after processing the same portion of the input) and in the same state, the branches are collapsed back to the parent branch.

Definition 8. Let $\boldsymbol{A}$ be an SAPDA and let C and C' be configurations of $\boldsymbol{A}$. We say that C *yields* C' *in one step*, denoted $C \vdash_{\boldsymbol{A}} C'$ ($\boldsymbol{A}$ is omitted if understood from the context), if one of the conditions below is satisfied.

- There exists a leaf v in C, $C(v) = (q, \sigma w, X\gamma)$ and a transition

$$(q_1, \alpha_1) \wedge \cdots \wedge (q_k, \alpha_k) \in \delta(q, \sigma, X)$$

  for which the following holds.
  - If $k = 1$, then C' is obtained from C by relabeling v with $(q_1, w, \alpha_1\gamma)$.
  - If $k > 1$, then C' is obtained from C by relabeling v with γ, and joining to it k *new* child nodes $v_1, \ldots, v_k$ such that $C'(v_i) = (q_i, w, \alpha_i)$, $i = 1, \ldots, k$.

  We refer to the first case as an *ordinary* transition and we refer to the second case as a *proper conjunctive* (or *split*) transition. In both cases we say that the computation step is by $(q_1, \alpha_1) \wedge \cdots \wedge (q_k, \alpha_k)$ applied to v.
- There exists a node v in C that has $k \geq 2$ child nodes $v_1, \ldots, v_k$, all of which are leaves labeled with the same (q, w, ϵ), and C' is obtained from C by removing all leaves v_i, $i = 1, \ldots, k$ and relabeling v with $(q, w, C(v))$.
  In this case we say that the computation step is by *collapsing* (of the child nodes of v).

That is, an ordinary transition by $(q_1, \alpha_1) \in \delta(q, \sigma, X)$ results in changing the leaf label $(q, \sigma w, X\gamma)$ to $(q_1, w, \alpha_1\gamma)$, a proper conjunctive transition by $(q_1, \alpha_1) \wedge \cdots \wedge , (q_k, \alpha_k) \in \delta(q, \sigma, X)$ results in changing the leaf label $(q, \sigma w, X\gamma)$ to γ and joining to the leaf k (new) child nodes labeled with $(q_1, w, \alpha_1), \ldots, (q_k, w, \alpha_k)$, respectively, and the collapsing of sibling branches results in deleting the corresponding leaf nodes and changing the parent node label γ to (q, w, γ).

As usual, $\vdash^+_{\boldsymbol{A}}$ and $\vdash^*_{\boldsymbol{A}}$ are the transitive and the reflexive transitive closures of $\vdash_{\boldsymbol{A}}$, respectively.

Definition 9. Let $\boldsymbol{A} = (Q, \Sigma, \Gamma, \delta, q_0, \perp)$ be an SAPDA and let $w \in \Sigma^*$.

- The *initial* configuration of $\boldsymbol{A}$ on w is the configuration $(q_0, w, \perp)$.
- An *accepting* configuration of $\boldsymbol{A}$ is a configuration (q, ϵ, ϵ) for all $q \in Q$.
- A *computation* of $\boldsymbol{A}$ on w is a sequence of configurations $C_0, \ldots, C_n$ such that
  - C_0 is the initial configuration, and
  - $C_{i-1} \vdash_{\boldsymbol{A}} C_i$ for $i = 1, \ldots, n$.

[4] That is, the configuration tree consists of the root only, which is also the only leaf of the tree.

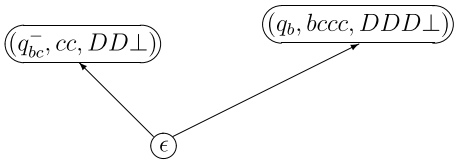

Fig. 1. A configuration of $\boldsymbol{A}$ on input *aaabbbccc* after reading *aaabbbc* by the left ("q_{bc}"-) branch and reading *aaabb* by the right ("q_{ac}"-) branch.

- An *accepting* computation of $\boldsymbol{A}$ on w is a computation whose last configuration is accepting.

The language $L(\boldsymbol{A})$ of $\boldsymbol{A}$ is the set of all $w \in \Sigma^*$ such that $\boldsymbol{A}$ has an accepting computation on w.[5]

Remark 10. (Cf. Remark 4.) As all active, i.e., currently processed, branches in an automaton computation are independent, the order in which they are processed is irrelevant. Therefore, an automaton computation may process the branches in a random order, or process them in a specified order, e.g., left-to-right or choosing each time a new branch, if possible. However, any computation satisfies the following conditions.

- A transition can be applied only to an active branch.
- If a branch empties, it cannot be chosen for the next transition (because it has no top symbol).
- If all sibling branches, i.e., branches originating at the same (parent) node, are empty and each of them emptied with the *same* remaining input and in the same state,[6] they are collapsed back to the parent branch.

Note that the (active) branches do not necessarily need to read the same input symbol, but before collapsing they must synchronize on the current state and the current position in the input.

Example 11. SAPDA $\boldsymbol{A} = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$ defined below accepts the *multiple agreement* language from Example 5

- $Q = \{q_0, q_{bc}^+, q_{bc}^-, q_{ac}^+, q_b, q_{ac}^-\}$,
- $\Sigma = \{a, b, c\}$,
- $\Gamma = \{\bot, D\}$, and
- δ is defined as follows.
  - $\delta(q_0, \epsilon, \bot) = (q_{bc}^+, \bot) \wedge (q_{ac}^+, \bot)$,

  ---

  - $\delta(q_{bc}^+, a, \bot) = \{(q_{bc}^+, \bot)\}$,
  - $\delta(q_{bc}^+, b, \bot) = \{(q_{bc}^+, D\bot)\}$,
  - $\delta(q_{bc}^+, b, D) = \{(q_{bc}^+, DD)\}$,
  - $\delta(q_{bc}^+, c, D) = \{(q_{bc}^-, \epsilon)\}$,
  - $\delta(q_{bc}^-, c, D) = \{(q_{bc}^-, \epsilon)\}$,
  - $\delta(q_{bc}^-, \epsilon, \bot) = \{(q_0, \epsilon)\}$,

  ---

  - $\delta(q_{ac}^+, a, \bot) = \{(q_{ac}^+, D\bot)\}$,
  - $\delta(q_{ac}^+, a, D) = \{(q_{ac}^+, DD)\}$,
  - $\delta(q_{ac}^+, b, D) = \{(q_b, D)\}$,
  - $\delta(q_b, b, D) = \{(q_b, D)\}$,
  - $\delta(q_b, c, D) = \{(q_{ac}^-, \epsilon)\}$,
  - $\delta(q_{ac}^-, c, D) = \{(q_{ac}^-, \epsilon)\}$,
  - $\delta(q_{ac}^-, \epsilon, \bot) = \{(q_0^-, \epsilon)\}$,

  and the value of δ on all other triples from $Q \times \Sigma \times \Gamma$ is empty.

The intuition for the above transitions is as follows. The "q_{bc}-branch" of $\boldsymbol{A}$ verifies that the input is of the form $a^m b^n c^n$, $m, n = 1, 2, \ldots$, and the "q_{ac}-branch" of $\boldsymbol{A}$ verifies that the input is of the form $a^m b^n c^m$, $m, n = 1, 2, \ldots$, see Fig. 1.

3. Deterministic SAPDA

In this section we define the notion of a deterministic SAPDA (DSAPDA) and show that the membership problem for DSAPDA is decidable in linear time. The definition of DSAPDA is analogous to the definition of a classical deterministic PDA.

[5] Alternatively, one can extend the definition of $\boldsymbol{A}$ with a set of *final* states $F \subseteq Q$, and define collapsing and acceptance by final states, similarly to the classical definition. It can be readily seen that such an extension results in an equivalent model of computation.

[6] That is, the branches are synchronized.

Definition 12. An SAPDA $\boldsymbol{A} = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$ is *deterministic* if

- for all $q \in Q$, $\sigma \in \Sigma \cup \{\epsilon\}$, and $X \in \Gamma$, $|\delta(q, \sigma, X)| \leq 1$[7] and
- whenever $\delta(q, \sigma, X) \neq \emptyset$ for some $\sigma \in \Sigma$, then $\delta(q, \epsilon, X) = \emptyset$.

Example 13. The automaton from Example 11 is deterministic.

Remark 14. Similarly to the non-deterministic case, one can extend the definition of $\boldsymbol{A}$ with a set of *final* states $F \subseteq Q$ and define the language accepted by final state as the language consisting of all words w such that $(q_0, w, \bot) \vdash^* C$, where the labels of the leaves of C are in $F \times \{\epsilon\} \times \Gamma^*$.

Note that, in contrast with (non-deterministic) SAPDA accepting by final state, branch collapsing is by *empty stack*, and the final state condition applies only to branches which have read the whole input, cf. footnote 5. This is because a deterministic automaton "cannot guess" the end of the input along a branch.

Theorem 15. *The membership problem for DSAPDA is decidable in linear time.*

For the proof of Theorem 15, like in the classical case, we must eliminate the possibility of existence of infinite sequences of ϵ-transitions in the automaton computation. We also must circumvent the potentially exponential number of branches that the automaton can open.

Infinite sequences of ϵ-transitions are treated as follows.

Definition 16. We say that computation

$$(q, \epsilon, X) = C_1 \vdash C_2 \vdash \cdots \vdash C_{i-1} \vdash C_i \vdash C_{i+1} \vdash \cdots \vdash C_m \tag{1}$$

of an SAPDA $\boldsymbol{A} = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$ is an *ϵ-chain* (from (q, ϵ, X)) if for all $i = 2, 3, \ldots, m-1$ the following holds.

- If C_i results from C_{i-1} by an ordinary transition, then C_{i+1} results from C_i by a transition applied to the same branch;
- if C_i results from C_{i-1} by a proper conjunctive transition, then C_{i+1} results from C_i by a transition applied to a new leaf open at the branch; and
- if C_i results from C_{i-1} by collapsing of sibling branches, then C_{i+1} results from C_i by a transition applied to the parent branch.

Note that, since $C_1 = (q.\epsilon, X)$, all computation steps in (1) are by ϵ-transitions.

Definition 17. Let B be a positive integer. We say that an SAPDA $\boldsymbol{A} = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$ is B-bounded if for all states $q \in Q$ and all stack symbols $X \in \Gamma$ the maximum length of an ϵ-chain from (q, ϵ, X) does not exceed B.

Lemma 18. *For each DSAPDA one can construct an equivalent B-bounded DSAPDA (for some computable positive integer B).*

The proof of the lemma is based on its classical counterpart. To apply it, we need the following definition.

Definition 19. Let $\boldsymbol{A} = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$ be an SAPDA. The *trace automaton* of $\boldsymbol{A}$ is the ordinary pushdown automaton $\boldsymbol{A}_T = (Q, \Sigma, \Gamma, \delta_T, q_0, \bot)$, where δ_T is defined as follows.

$$\delta_T(q, a, X) = \{(q_i, \alpha_i) : (q_1, \alpha_1) \wedge \cdots \wedge (q_k, \alpha_k) \in \delta(q, a, X), \;\; i = 1, 2, \ldots, k\}.$$

Proof of Lemma 18. Let $\boldsymbol{A}$ and $\boldsymbol{A}_T$ be as above. It follows from Definition 19 that there is an ϵ-chain from (q, ϵ, X) of length m if and only if there is a computation of A_T from (q, ϵ, X) of length m.

Using [1, Algorithm 2.16, pp. 187–188], one can compute a positive integer B such that if there is a computation of $\boldsymbol{A}_T$ from (q, ϵ, X) of length B, then there is an infinite computation of $\boldsymbol{A}_T$ from (q, ϵ, X),[8] implying that there is an "infinite" ϵ-chain from (q, ϵ, X) in $\boldsymbol{A}$.

Thus, for all pairs $(q, X) \in Q \times \Gamma$ for which there is a computation of $\boldsymbol{A}_T$ from (q, ϵ, X) of length B, we can redefine $\delta(q, \epsilon, X)$ to be $\emptyset$ without affecting the accepted language. □

So, by Lemma 18, we may assume that DSAPDA $\boldsymbol{A} = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$ under consideration is B-bounded and comes together with the bound B. We further modify $\boldsymbol{A}$ as follows.

[7] In what follows we write $\delta(q, \sigma, X) = (q', X')$ instead of $\delta(q, \sigma, X) = \{(q', X')\}$.

[8] In [1] the automaton is deterministic, but the algorithm works for non-deterministic automata as well.

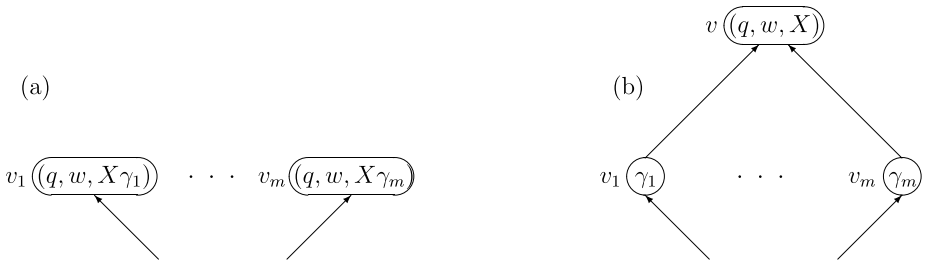

Fig. 2. Nodes $v_1, v_2, \ldots, v_m$ in a configuration: (a) before and (b) after merging to node v.

Definition 20. Transitions of the form $\delta(q, \epsilon, X) = (q', \epsilon)$ are called *decreasing*.

For all $q, q' \in Q$ and $X \in \Gamma$ such that

$$(q, \epsilon, X) \vdash_{\boldsymbol{A}}^{*} (q', \epsilon, \epsilon)\,, \tag{2}$$

we (equivalently) replace the ϵ-transition $\delta(q, \epsilon, X)$ with the decreasing ϵ-transition $\delta(q, \epsilon, X) = (q', \epsilon)$.[9] Such replacing of ϵ-transitions will be referred to as *ϵ-modification*.

To circumvent the potentially exponential number of branches that the automaton can open, we observe the following. If leaf labels of a configuration have the same state/top stack symbol combinations, then they behave identically on the same input, as long as each stack height is greater than or equal to the initial one. This happens because the automaton is deterministic.

By this observation, we do not need to deal with the potentially exponential number of branches to decide membership for DSAPDA. From time to time the branch heads having the same state/remaining input/top stack symbol combinations are *merged* to one head. The computation then proceeds on the merged branch, as long as its stack is not empty. Once the merged branch empties, the computation *separates* to the original branches, see Fig. 2.

Namely, assuming that the DSAPDA under consideration is ϵ-modified and B-bounded for some positive integer B, we shall process an input w as follows, see Remark 10.

The automaton computation (that also involves branch merging and separating) advances in $|w|+1$ rounds, where round i, $i = 1, 2, \ldots, |w|$, consists of four stages.

1. In the first stage, the automaton makes all possible moves by a decreasing ϵ-transition, by collapsing, or by separating merged branches whose stack is empty.
2. In the second stage, branch-heads are grouped by the same state/top stack symbol combination and the elements of each group are merged into a new branch.
3. In the third stage, the automaton makes all possible moves by an ϵ-transition or collapsing.
4. Finally, in the fourth stage, every active branch (one after another) reads the ith symbol of w.

Round $|w| + 1$ consists of stages 1, 2, and 3 only.

In the rest of this section we show that the complexity of the above $|w|$-round computation is $O(|w|)$, thus proving Theorem 15.

Note that the extension of the automaton computation with merging and separating branches of the stack memory tree, requires a directed acyclic graph structure rather than a tree. Therefore, we shall deal with *graph* configurations which are defined as follows.

Definition 21. (Cf. Definition 7.) A *graph configuration* of a DSAPDA $\boldsymbol{A} = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$ is a labeled directed acyclic graph such that

- each edge exiting a node of out-degree greater than 1 enters a node of in-degree 1,[10]
- each internal node is labeled with $\alpha \in \Gamma^*$ (being the stack content associated with the node), and
- each sink node is labeled with $(q, w, \alpha) \in Q \times \Sigma^* \times \Gamma^*$ (being the components of the corresponding active branch of the computation).

For a graph configuration C, we denote by $N(C)$ the set of all nodes of C.

[9] Note that (2) is decidable, because $\boldsymbol{A}$ is B-bounded, and also implies that $\delta(q, \epsilon, X)$ is nonempty.

[10] Such edges correspond to a proper conjunctive transition, whereas edges exiting a node of out-degree 1 correspond to merging branches with the same state/remaining input/top stack symbol combination. The latter also includes the degenerate merging in which there is only one branch-head with some state/remaining input/top stack symbol combination.

Definition 22. (Cf. Definition 8.) Let $\boldsymbol{A} = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$ be a DSAPDA and let C and C' be graph configurations of $\boldsymbol{A}$. We say that C *yields* C' *in one step*, denoted $C \Vdash_{\boldsymbol{A}} C'$ ($\boldsymbol{A}$ is omitted if understood from the context), if one of the conditions below is satisfied.

- There exists a sink node v in C, $C(v) = (q, \sigma w, X\gamma)$ and a transition
$$\delta(q, \sigma, X) = (q_1, \alpha_1) \wedge \cdots \wedge (q_k, \alpha_k)$$
for which the following holds.
  - If $k = 1$, then C' is obtained from C by relabeling v with $(q_1, w, \alpha_1\gamma)$.
  - If $k > 1$, then C' is obtained from C by relabeling v with γ, and joining to it k *new* child nodes $v_1, \ldots, v_k$ such that $C'(v_i) = (q_i, w, \alpha_i)$, $i = 1, \ldots, k$.

  Like in Definition 8, we refer to the first case as an ordinary transition and we refer to the second case as a proper conjunctive or split transition. In both cases we say that the computation step is by $(q_1, \alpha_1) \wedge \cdots \wedge (q_k, \alpha_k)$ (applied to v).
- There exists a node v in C that has $k \geq 2$ child nodes $v_1, \ldots, v_k$, all of which are sink nodes labeled with the same (q, w, ϵ), and C' is obtained from C by removing all sink nodes v_i, $i = 1, \ldots, k$ and relabeling v with $(q, w, C(v))$.
  Like in Definition 8, in this case we say that the computation step is by *collapsing* (of child nodes of v).
- There exist sink nodes $v_1, v_2, \ldots, v_m$, $m \geq 2$, of C such that $C(v_i) = (q, w, X\gamma_i)$, $i = 1, 2, \ldots, m$, and C' is obtained from C by joining to it a new sink node v labeled with (q, w, X) that becomes the only child node of each of the v_is, $i = 1, 2, \ldots, m$, and relabeling v_i with γ_i, $i = 1, 2, \ldots, m$.
  We refer to this case as *merging* (of $v_1, v_2, \ldots, v_m$), see Fig. 2.
- There exists a sink node v of C, $C(v) = (q, w, \epsilon)$, that has $m \geq 2$ parent nodes $v_1, v_2, \ldots, v_m$ and C' is obtained from C by deleting v and relabeling v_i with $(q, w, C(v_i))$, $i = 1, 2, \ldots, m$.
  In this case we say that the computation step is by *separating* (applied to v).

The notions of the initial graph configuration, an accepting graph configuration, a "$\Vdash$-computation" (i.e., a sequence of graph configurations starting at the initial one and related by $\Vdash$), and an accepting $\Vdash$-computation are similar to those introduced in Definition 9.

In what follows we restrict ourselves to $\Vdash$-computations consisting of four stage rounds described above (plus the three-stage last round, of course).

Such computations possess the following property.

Lemma 23. *For an ϵ-modified DSAPDA, no separating transition is possible at stage* 3 *of a $\Vdash$-computation round.*

Proof. Assume to the contrary, that a separating transition can be applied to a node v that was added by a merging transition at stage 2 of the current or an earlier round of the computation. Let v result in merging nodes $v_1, v_2, \ldots, v_m$ labeled $(q, w', X\gamma_1), (q, w', X\gamma_2), \ldots, (q, w', X\gamma_m)$, respectively. Then the label of v after merging is (q, w', X).

When a separating transition is applied to v, its stack component must be empty. That is, the label of v must be of the form (q', w', ϵ), implying $(q, w', X) \Vdash^+ (q', w', \epsilon)$. Thus, $(q, \epsilon, X) \Vdash^+ (q', \epsilon, \epsilon)$ and, since the automaton is ϵ-modified,

$$\delta(q, \epsilon, X) = (q', \epsilon, \epsilon)\,. \tag{3}$$

By definition, transition (3) should have been applied to each node v_i, $i = 1, 2, \ldots, m$ at stage 1: $(q, w', X\gamma_i) \Vdash (q', w', \gamma_i)$. Therefore, the $(q, w', X\gamma_i)$s cannot be the labels of the nodes v_is just before the merging, in contradiction to our assumption. □

In a straightforward manner, each $\Vdash$-computation can be transformed into a $\vdash$-computation by separating nodes and vice versa, each $\vdash$-computation can be transformed into a $\Vdash$-computation by merging nodes.[11] Thus, we have Lemma 24 below.

Lemma 24. *Let $\boldsymbol{A} = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$ be a DSAPDA. Then for all $w \in \Sigma^*$ and all $q \in Q$, $(q_0, w, \bot) \vdash_{\boldsymbol{A}} (q, \epsilon, \epsilon)$ if and only if $(q_0, w, \bot) \Vdash_{\boldsymbol{A}} (q, \epsilon, \epsilon)$.*

Next, we are going to estimate the length of an accepting $\Vdash$-computation of a DSAPDA on an input word.

Definition 25. Let $\boldsymbol{C} = C_0 \Vdash C_1 \Vdash \cdots \Vdash C_n$ be a $\Vdash$-computation. For $i = 0, 1, \ldots, n$ we define the *number of symbols pushed into the stack in the first i* transitions, denoted $P_{\boldsymbol{C}}(i)$, recursively, as follows.

[11] These transformations are unique up to the order of the computation steps.

- $P_{\boldsymbol{C}}(0) = 1$, and
- if the ith computation step is by an ordinary or by a proper conjunctive transition

$$\delta(q, \sigma, X) = (q_1, \alpha_1) \wedge \cdots \wedge (q_k, \alpha_k), \tag{4}$$

then

$$P_{\boldsymbol{C}}(i+1) = P_{\boldsymbol{C}}(i) + \sum_{j=1}^{k} |\alpha_j|.$$

Otherwise, $P_{\boldsymbol{C}}(i+1) = P_{\boldsymbol{C}}(i)$.

Finally, we define the *number of symbols pushed into the stack by computation* $\boldsymbol{C}$, denoted $P(\boldsymbol{C})$, as $P(\boldsymbol{C}) = P_{\boldsymbol{C}}(n)$.

Below we employ the following notation. Let $\boldsymbol{A} = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$ be a DSAPDA and let $\boldsymbol{C}$ be a $\Vdash$-computation of $\boldsymbol{A}$.

- The maximal number of branches open in a single transition of $\boldsymbol{A}$ is denoted by $b_{\boldsymbol{A}}$.
- The maximal number of symbols pushed into the stack in a single transition of $\boldsymbol{A}$ is denoted by $p_{\boldsymbol{A}}$. That is,

$$p_{\boldsymbol{A}} = \max\left\{ \sum_{i=1}^{k} |\alpha_i| : \delta(q, \sigma, X) = (q_1, \alpha_1) \wedge \cdots \wedge (q_k, \alpha_k),\ \ (q, \sigma, X) \in Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \right\}.$$

- The length of $\boldsymbol{C}$ is denoted by $|\boldsymbol{C}|$. For example, in Definition 25, $|\boldsymbol{C}| = n$.
- The number of ordinary transitions of $\boldsymbol{C}$ is denoted $o_{\boldsymbol{C}}$.
- The number of non-decreasing ordinary transitions of $\boldsymbol{C}$ is denoted $o'_{\boldsymbol{C}}$.
- The number of proper conjunctive transitions of $\boldsymbol{C}$ is denoted $p_{\boldsymbol{C}}$.
- The number of collapsing transitions of $\boldsymbol{C}$ is denoted $c_{\boldsymbol{C}}$.
- The number of merging transitions of $\boldsymbol{C}$ is denoted $m_{\boldsymbol{C}}$.
- The number of separating transitions of $\boldsymbol{C}$ is denoted $s_{\boldsymbol{C}}$.

Propositions 26 and 27 are easy consequences of this notation.

Proposition 26. *Let $\boldsymbol{C}$ be a computation of a DSAPDA $\boldsymbol{A}$. Then*

(*i*) $o_{\boldsymbol{C}} + p_{\boldsymbol{C}} + c_{\boldsymbol{C}} + m_{\boldsymbol{C}} + s_{\boldsymbol{C}} = |\boldsymbol{C}|$,
(*ii*) $P(\boldsymbol{C}) \leq (o'_{\boldsymbol{C}} + p_{\boldsymbol{C}})p_{\boldsymbol{A}} + 1$.

Proof. By the definition of a $\Vdash$-computation (Definition 22), each transition of a DSAPDA on a graph configuration is ordinary, proper conjunctive, collapsing, merging, or separating. This proves (*i*).

The proof of (*ii*) is an easy induction on $|\boldsymbol{C}|$. For the basis $|\boldsymbol{C}| = 1$, by definition, $P(\boldsymbol{C}) = 1$, and for the inductive step, by the same definition, $P(\boldsymbol{C})$ can be incremented only by either non-decreasing ordinary transition or by a proper conjunctive transition, and, in any case, by at most $p_{\boldsymbol{A}}$. □

Proposition 27. *Let $\boldsymbol{C}$ be an accepting computation. Then*

(*i*) $c_{\boldsymbol{C}} = p_{\boldsymbol{C}}$, $s_{\boldsymbol{C}} = m_{\boldsymbol{C}}$,
(*ii*) $o_{\boldsymbol{C}} + p_{\boldsymbol{C}} = P(\boldsymbol{C})$,
(*iii*) $s_{\boldsymbol{C}} \leq P(\boldsymbol{C})$, *and*
(*iv*) $|\boldsymbol{C}| \leq 4P(\boldsymbol{C})$.

Proof. Since an accepting computation starts and ends at the same node (the source), the number of collapsing transitions is equal to the number of proper conjunctive transitions and the number of separating transitions and is equal to the number of merging transitions, which is (*i*).

For the proof of (*ii*), we observe that each transition (4) replaces the topmost symbol in the respective branch and, eventually, each symbol pushed into the stack becomes the topmost symbol of some branch, because the computation is accepting.

The proof of (*iii*) uses a similar argument. If a separating transition applies to a node v, that node must be previously added by a merging transition applied to some nodes with the same state/remaining input/top stack symbol combination, (q, w, X), say, that becomes the label of v.

When a separating transition is applied to v, its stack must be empty. In particular, X must be replaced by application of some ordinary or a proper conjunctive transition. Thus, the number of separating transitions is at most as the number of ordinary or proper conjunctive transitions, that, by (*ii*), is $P(\boldsymbol{C})$.

Finally, (iv) follows from the (in)equalities below.

$$\begin{aligned}|\boldsymbol{C}| &= o_{\boldsymbol{C}} + p_{\boldsymbol{C}} + c_{\boldsymbol{C}} + m_{\boldsymbol{C}} + s_{\boldsymbol{C}} \\ &= (o_{\boldsymbol{C}} + p_{\boldsymbol{C}}) + p_{\boldsymbol{C}} + 2s_{\boldsymbol{C}} \\ &\leq 2(o_{\boldsymbol{C}} + p_{\boldsymbol{C}}) + 2s_{\boldsymbol{C}} \\ &\leq 4P(\boldsymbol{C})\,,\end{aligned}$$

where

- the first equality is by Proposition 26(i),
- the second equality is by (i),
- the first inequality is trivial, and
- the last inequality is by (ii) and (iii). □

Even though, Proposition 27(iv) provides an upper bound on the computation length in terms of the number of symbols pushed into the stack in the course of the computation, it is not sufficient, because the computation time also depends on the number of configurations' nodes involved in each transition. Namely, whereas the number of nodes involved in a proper conjunctive or a collapsing transition is bounded by $b_{\boldsymbol{A}} + 1$ – the maximal number of branches open in a single transition plus the parent branch, the number of nodes involved in a merging or a separating transition is not bounded. Thus, we shall estimate the number of nodes (counting repetitions, of course) involved in all transitions of the computation. For this we need to refine the notion of the number of symbols pushed into the stack. Specifically, we need the distribution of $P_{\boldsymbol{C}}(i)$ among the nodes of the ith graph configuration C_i, that is defined as follows.

Definition 28. Let $\boldsymbol{C}$ be a $\Vdash$-computation $C_0 \Vdash C_1 \Vdash \cdots \Vdash C_n$ such that a node deleted at some collapsing or separating transition in $\boldsymbol{C}$ is not introduced later by a splitting or merging transition.[12]

For a node $v \in \bigcup_{i=0}^{n} N(C_i)$ and $i = 0, 1, \ldots, n$, we define the *number of symbols pushed into the stack of v in the first i transitions*, denoted $P^v_{\boldsymbol{C}}(i)$, recursively, as follows.

- If $N(C_0) = \{v\}$, then $P^v_{\boldsymbol{C}}(0) = 1$. Otherwise, $P^v_{\boldsymbol{C}}(0) = 0$.
- If the ith computation step is by an ordinary transition $\delta(q, \sigma, X) = (q, \alpha)$ applied to v, then $P^v_{\boldsymbol{C}}(i+1) = P^v_{\boldsymbol{C}}(i) + |\alpha|$.
- If v is added to C_i by a proper conjunctive transition (4) and the stack component of the label of v in C_{i+1} is α_j, then $P^v_{\boldsymbol{C}}(i+1) = |\alpha_j|$.
- If v is added to C_i by a merging transition, then $P^v_{\boldsymbol{C}}(i+1) = 1$.
- Otherwise, $P^v_{\boldsymbol{C}}(i+1) = P^v_{\boldsymbol{C}}(i)$.

Finally, we define the *number of symbols pushed into the stack of v by computation $\boldsymbol{C}$*, denoted $P^v_{\boldsymbol{C}}$, as $P^v_{\boldsymbol{C}} = P^v_{\boldsymbol{C}}(j)$, where

$$j = \max\{i : v \in N(C_i)\}\,.$$

Proposition 29. *For a $\Vdash$-computation $\boldsymbol{C}$,*

$$\sum_{v \in \bigcup_{i=0}^{n} N(C_i)} P^v_{\boldsymbol{C}} = P(\boldsymbol{C}) + m_{\boldsymbol{C}}\,. \tag{5}$$

Proof. By Definitions 25 and 28, each symbol pushed into the stack is pushed into the stack of some node by a non-merging transition and vice versa. Therefore, the left side of (5) differs from $P(\boldsymbol{C})$ by the number $m_{\boldsymbol{C}}$ of merging transitions. □

Proposition 30. *Let $\boldsymbol{C}$ be an accepting $\Vdash$-computation. Then the number of nodes involved of all merging transitions in $\boldsymbol{C}$ does not exceed $P(\boldsymbol{C}) + 2m_{\boldsymbol{C}}$.*

Proof. This is because each merging transition deletes the topmost symbol from each node it merges, and eventually, each symbol pushed into the stack becomes the topmost symbol of some branch. Thus, the number of merging transitions in

[12] That is, for all $i = 0, 1, \ldots, n-1$, if $v \in N(C_i)$, but $v \notin N(C_{i+1})$, then for all $j > i$, $v \notin N(C_j)$ Recall that $N(C)$ denotes the set of all nodes of a graph configuration C. This requirement is not restrictive, because we may rename the nodes in $\bigcup_{i=1}^{n} N(C_i)$, if necessary.

which node v is involved is at most $P^v_{\boldsymbol{C}}$. By Proposition 29, the number of nodes being merged does not exceed $P(\boldsymbol{C}) + m_{\boldsymbol{C}}$. In addition, each merging transition adds a new sink node.[13] □

Corollary 31. *Let $\boldsymbol{C}$ be an accepting $\Vdash$-computation of a DSAPDA $\boldsymbol{A}$. Then the number of nodes (counting repetitions) involved in all transitions of $\boldsymbol{C}$ does not exceed* $(4b_{\boldsymbol{A}} + 6)((o'_{\boldsymbol{C}} + p_{\boldsymbol{C}})p_{\boldsymbol{A}} + 1)$.

Proof. The calculation is straightforward.

- The number of nodes involved in the ordinary transitions of $\boldsymbol{C}$ is $o_{\boldsymbol{C}}$.
- The number of nodes involved in the proper conjunctive transitions of $\boldsymbol{C}$ is at most $(b_{\boldsymbol{A}} + 1)p_{\boldsymbol{C}}$, because a proper conjunctive transition of $\boldsymbol{A}$ involves a parent node and at most $b_{\boldsymbol{A}}$ child nodes it opens.
- Similarly, the number of nodes involved in the collapsing transitions of $\boldsymbol{C}$ is also at most $(b_{\boldsymbol{A}} + 1)c_{\boldsymbol{C}}$.
- By Proposition 30, the number of nodes involved in all merging transitions in $\boldsymbol{C}$ is at most $P(\boldsymbol{C}) + 2m_{\boldsymbol{C}}$.
- By Proposition 27(i), the number of nodes involved in all separating transitions in $\boldsymbol{C}$ is also at most $P(\boldsymbol{C}) + 2s_{\boldsymbol{C}}$.

Summing up, we obtain that the number of nodes involved in all transitions of $\boldsymbol{C}$ is at most

$$\begin{aligned} o_{\boldsymbol{C}} + (b_{\boldsymbol{A}} + 1)p_{\boldsymbol{C}} + (b_{\boldsymbol{A}} + 1)c_{\boldsymbol{C}} + P(\boldsymbol{C}) + 2m_{\boldsymbol{C}} + P(\boldsymbol{C}) + 2s_{\boldsymbol{C}} & \\ \leq (b_{\boldsymbol{A}} + 1)(o_{\boldsymbol{C}} + p_{\boldsymbol{C}} + c_{\boldsymbol{C}} + m_{\boldsymbol{C}} + s_{\boldsymbol{C}}) + 2P(\boldsymbol{C}) & \\ = (b_{\boldsymbol{A}} + 1)|\boldsymbol{C}| + 2P(\boldsymbol{C}) & \\ \leq (4b_{\boldsymbol{A}} + 6)P(\boldsymbol{C}) & \\ \leq (4b_{\boldsymbol{A}} + 6)((o'_{\boldsymbol{C}} + p_{\boldsymbol{C}})p_{\boldsymbol{A}} + 1), & \end{aligned}$$

where

- the first inequality is trivial,
- the equality is by Proposition 26(i),
- the second inequality is by Proposition 27(iv), and
- the last inequality is by Proposition 26(ii). □

Now we are ready for the proof of Theorem 15.

Proof of Theorem 15. Recall that DSAPDA $\boldsymbol{A} = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$ is ϵ-modified and B-bounded for some positive integer B, and an input w is processed in $|w| + 1$ rounds, as described before Definition 21.

Let $\boldsymbol{C}$ be an accepting $\Vdash$-computation of $\boldsymbol{A}$ on an input w. For the proof of Theorem 15 it suffices to show that

$$o'_{\boldsymbol{C}} + p_{\boldsymbol{C}} \leq |w||Q||\Gamma|\frac{b_{\boldsymbol{A}}^{B+2} - 1}{b_{\boldsymbol{A}} - 1}. \tag{6}$$

Indeed, (6), together with Corollary 31, implies that the number of nodes (counting repetitions) involved in all transitions of $\boldsymbol{C}$ is $O(|w|)$, and, since the number of symbols processed at each node by a single transition is bounded by $p_{\boldsymbol{A}}$, the theorem would follow.

For the proof of (6), we observe that non-decreasing ordinary transitions and proper conjunctive transitions can appear only in stages 3 and 4 of the $\Vdash$-computation rounds. Therefore, the number of such transitions can be calculated as follows.

For each of the $|w| + 1$ computation rounds, stage 3 starts from at most $|Q||\Gamma|$ merged nodes created at stage 2.

By Lemma 23, for each node added by a merging transition, "pruning" the graph configurations obtained in stages 3 and 4 of the corresponding round at this node results in a tree configuration whose underlying tree is of degree at most $b_{\boldsymbol{A}}$ and of height at most $B + 1$. This is because there are at most B consecutive ϵ-transitions along each path in stage 3 and one transition on the corresponding input symbol in stage 4. Therefore, the maximal number of such transitions is at most

$$\sum_{j=1}^{B+1} b_{\boldsymbol{A}}^j = \frac{b_{\boldsymbol{A}}^{B+2} - 1}{b_{\boldsymbol{A}} - 1},$$

which, together with the maximal number $(|w| + 1)|Q||\Gamma|$ of all merged nodes occurring in all $|w| + 1$ rounds of $\boldsymbol{C}$, proves (6). □

[13] Note that a node may be merged several times, which is also addressed in the proof.

4. Item SAPDA and *LR*(0) conjunctive grammars

In this section we extend the classical notion of item pushdown automata and $LR(0)$ grammars to conjunctive grammars.[14] We start with an informal description of the $LR(0)$ parser, that is, actually an item SAPDA. Then we define the canonical collection of item sets of a conjunctive grammar, define item SAPDA (needed for the proof of correctness of our parsing algorithm), and show that the latter is a natural counterpart of rightmost derivations (see Definition 33 below) of the corresponding conjunctive grammar. Finally, we define $LR(0)$ conjunctive grammars for which the corresponding item SAPDA is deterministic.

Similarly to the case of ordinary $LR(0)$ grammars, a conjunctive grammar is $LR(0)$ if its set of $LR(0)$-items is conflict-free. However, in the case of conjunctive grammars, in addition to shift–reduce and reduce–reduce conflicts, there are also split–split, split–reduce, and split–shift conflicts, where the split operation of a parser corresponds to a proper conjunctive (split) transition.

Namely, a rightmost derivation of a conjunctive grammar is divided into ordinary derivations, each consisting of a sequence of applications of ordinary productions. These derivations are "synchronized" by proper conjunctive productions (which may cause the above "split" conflicts). In each ordinary derivation, the parser operates exactly as the ordinary $LR(0)$ parser and is based on the same analysis of productions (with handles, viable prefixes, items, and sets of valid items). For every proper conjunctive production $B \to \alpha_1 \& \cdots \& \alpha_k$, the parser anticipates a potential reduction by such a production.[15] Accordingly, it splits the corresponding branch of its stack into k sibling branches, one for each conjunct in the right side of the production. Then branch i processes the same portion of the input starting, to some extent, from the closure of

$$\{B_i \to \cdot\alpha_i : B \to \alpha_1 \& \cdots \& \alpha_k \in P\},$$

where B_i is a copy of B associated with α_i. If, after reading the same portion of the input, branch i ends with $B_i \to \alpha_i\cdot$, in its stack, $i = 1, 2, \ldots, k$ (the same B for all branches), the parser reduces by $B \to \alpha_1 \& \cdots \& \alpha_k$, collapsing the k branches to the parent branch, and shifts B to the stack of the parent branch.

Unlike in the classical case, when the"conjunctive" parser cannot reduce, it not necessarily shifts the coming input symbol to the stack. This is because that symbol might be the first symbol of a subword resulting in contraction of the (same) words derived from the α_is. If so, B is shifted instead (after collapsing, of course). For this reason, in such a case, shift comes *after* reduction to B.

The correctness proof is a natural extension of the proof for the ordinary $LR(0)$ parsing. In particular, the latter apply word by word to the non-conjunctive segments of the computation.

4.1. Preliminary definitions

To introduce item SAPDA and $LR(0)$ *conjunctive* grammars we need some preliminary definitions. These definitions are almost the same as their non-conjunctive counterparts (see [5, Section 10.6], say), because they are needed for ordinary derivations along a single branch. The "only" difference is existence of split variables.

We start with the notion of a *rightmost* derivation.

Definition 32. Let $\mathcal{A}$ be a conjunctive formula. The set of *rightmost* variables of $\mathcal{A}$, denoted $\boldsymbol{R}(\mathcal{A})$, is defined by the following recursion.

- If $\mathcal{A} \in (V \cup \Sigma)^*$, then
$$\boldsymbol{R}(\mathcal{A}) = \begin{cases} \emptyset, & \text{if } \mathcal{A} \in \Sigma^* \\ \text{the rightmost variable of } \mathcal{A}, & \text{otherwise} \end{cases}.$$
- If $\mathcal{A} = \mathcal{B}\mathcal{C}$, then
$$\boldsymbol{R}(\mathcal{A}) = \begin{cases} \boldsymbol{R}(\mathcal{C}), & \text{if } \boldsymbol{R}(C) \neq \emptyset \\ \boldsymbol{R}(\mathcal{B}), & \text{otherwise} \end{cases}.$$
- If $\mathcal{A} = (\mathcal{A}_1 \ \& \ \cdots \ \& \ \mathcal{A}_n)$, $n \geq 2$, then $\boldsymbol{R}(\mathcal{A}) = \bigcup_{i=1}^{n} \boldsymbol{R}(\mathcal{A}_i)$.

Definition 33. (Cf. [3, Definition 20]) A derivation step of a conjunctive grammar is *rightmost*, if it is either by or, if no contraction is possible, by a production applied to a rightmost variable of the conjunctive formula. A derivation is *rightmost*, if each its derivation step is rightmost.

[14] We assume that the reader is familiar with the $LR(0)$ parsing algorithm, see [5, Section 10.7], say.

[15] Of course, this depends on the set of valid items.

The one-step rightmost derivation relation is denoted $\Rightarrow_R$, and, as usual, $\Rightarrow_R^+$ and $\Rightarrow_R^*$ are the transitive and the reflexive transitive closures of $\Rightarrow_R$, respectively.

Similarly to context-free grammars, in conjunctive grammars, the *order* in which independent derivation productions are applied does not affect the derived word, see Remark 4. Therefore, we have the following lemma.

Lemma 34. *Let $\gamma \in (V \cup \Sigma)^*$ and $w \in \Sigma^*$ be such that $\gamma \Rightarrow^* w$. Then $\gamma \Rightarrow_R^* w$.*

A conjunctive formula derived by a rightmost derivations is called a *right-sentential form*.

The preliminary definitions we need are as follows. Let C be a variable.

- An *ordinary C-right-sentential form* is a word $\gamma \in (V \cup \Sigma)^*$ such that $C \Rightarrow_R^* \gamma$. In what follows, C is either the start variable S at the beginning of the derivation or the new branch start variable $\widehat{S}$ in derivations along newly open branches.
- A *handle* of an ordinary C-right-sentential form γ is a subword β of γ such that

  $$C \Rightarrow_R^* \delta A w \Rightarrow_R \delta\beta w = \gamma\,.$$

  That is, a handle is a subword that could be introduced in the last *non-contraction* step in a rightmost derivation of γ from C. Note that, since the derivation is rightmost, no production or contraction is applied to the left of A.
- A *viable prefix* of an ordinary C-right-sentential form γ is any prefix of γ ending no farther right than the right end of a handle of γ.
- An *item* is an expression of the form $A \to \alpha \cdot \beta$, where $A \to \alpha\beta \in P$.
- Items of the form $A \to \alpha\cdot$ are called *complete*.
- An item $A \to \alpha \cdot \beta$ is *valid* for an C-viable prefix γ if there is the rightmost derivation

  $$C \Rightarrow_R^* \delta A w \Rightarrow \delta\alpha\beta w$$

  and $\delta\alpha = \gamma$.
- Items of the form $A \to \alpha \cdot B\beta$, where B is a split variable, are called *split*.
- A set of items is *split* if it contains a split item. Otherwise, it is *ordinary*.

Example 35. Since in grammar G from Example 5 the only split variable S does not appear in the right side of any production, G has no split items.

Next we extend [1, Theorem 5.10, p. 387] and [5, Theorem 10.9, p. 251] to computation of sets of items valid for viable prefixes.

Definition 36. Let I be a set of $LR(0)$ items. We define the item-closure of I, denoted $[I]$, as the minimal (with respect to inclusion) set of items such that

- $I \subseteq [I]$, and
- if $A \to \alpha \cdot B\beta \in [I]$ and $B \to \gamma \in P$, then $B \to \cdot\gamma \in [I]$.

A set of items I is *closed*, if $I = [I]$. The set of all closed sets of items is denoted by $\mathcal{I}$.

We proceed with the definition of the (transition) function $g : \mathcal{I} \times (V \cup \Sigma) \to \mathcal{I}$. Let $I \in \mathcal{I}$ and let $X \in V \cup \Sigma$. Then

$$g(I, X) = [\{A \to \alpha X \cdot \beta : A \to \alpha \cdot X\beta \in I\}]\,.$$

We extend g to the function $g : \mathcal{I} \times (V \cup \Sigma)^* \to \mathcal{I}$ by

- $g(I, \epsilon) = I$, and
- $g(I, \gamma X) = g(g(I, \gamma), X)$, where $X \in V \cup \Sigma$.

Theorem 38 below employs the following notation. For a variable $C \in V$ we define the set of items I_C by

$$I_C = [\{C \to \cdot\alpha : C \to \alpha \in P\}]\,.$$

Remark 37. If there is no ordinary production with variable C in the left side, then $I_C = \emptyset$.

Theorem 38. *Let γ be a C-viable prefix of G. Then $g(I_C, \gamma)$ is the set of all items valid for γ.*

The proof of Theorem 38 is exactly like that of [1, Theorem 5.10, p. 387] or [5, Theorem 10.9, p. 251]. This is because all the above definitions deal only with ordinary productions and, therefore, the theorem actually is about ordinary context-free grammars. We omit the proof.

To define an item SAPDA that acts as a *non-deterministic* LR parser for conjunctive languages, we must first simplify the grammar as follows.

Definition 39. A conjunctive grammar is *simple*, if the three following conditions are satisfied.

1. The start variable is not split and does not appear in the right side of any production.
2. For each split variable (see Definition 1) there is a unique production with it in the left side.
3. There is a "new branch start variable" $\widehat{S}$ and for each proper conjunctive production

$$B \to (\alpha_1 \& \alpha_2 \& \cdots \& \alpha_k) \tag{7}$$

there are k (non-split) copies B_i of B, $i = 1, 2, \ldots, k$. Also, there are ordinary productions of the form

$$\widehat{S} \to B_i \tag{8}$$

and ordinary productions of the form

$$B_i \to \alpha_i\,, \tag{9}$$

$i = 1, 2, \ldots, k$. In addition, $\widehat{S}$ appears only in productions of the form (8) and B_is appear only in productions of the form (8) or (9).

Obviously, variables $\widehat{S}$ and B_is, $i = 1, 2, \ldots, k$, are useless. They, actually, belong to the parser and are needed for monitoring the branches open by B. Namely, if the ith branch reduces by $\widehat{S} \to B_i$, then the corresponding stack becomes empty and the automaton enters *state* B. If this happens for all $i = 1, 2, \ldots, k$, the branches collapse and B is shifted to the stack of the parent branch.

Roughly speaking, the intuition lying behind clause 2 of the definition is as follows.

Were there two proper conjunctive productions $B \to \alpha_1 \& \alpha_2$ and $B \to \alpha_1' \& \alpha_2'$, say, it might be possible that first branch reduces to $[\widehat{S} \to B_1 \cdot]$ because of $B_1 \to \alpha_1 \cdot$ and the second branch reduces to $[\widehat{S} \to B_2 \cdot]$ because of $B_2 \to \alpha_2' \cdot$. In such a case, no valid reduction is possible. Nevertheless, the parser would reduce to B.

Of course, one might think of marking the B_is by the corresponding productions. However, for this we need a bit different (but still equivalent) model of computation: "partially" or "input" synchronized alternating PDA in which collapsing is possible from different states of the sibling leaf branches with the same remaining input. In such a case, the state of the parent branch after collapsing depends on the states of the child branches. In this equivalent model, the automaton can remember the conjuncts participating in the branch reductions and decide whether they come from the same production.

Anyway, every conjunctive grammar can be equivalently converted to a simple form, say, in the following manner, cf. [1, p. 372].

Clause 1 can be achieved, by adding a new start variable $\overline{S}$ and production $\overline{S} \to S$.

Clause 2 can be achieved, by adding, for each proper conjunctive production π of the form (7), a new split variable B_π, productions $B \to B_\pi$ and

$$B_\pi \to (\alpha_1 \& \alpha_2 \& \cdots \& \alpha_k)\,, \tag{10}$$

and deleting production π.[16]

Finally, clause 3 can be achieved just by adding the required variables and productions of the forms (8) and (9).

As we shall see in the next section, $LR(0)$ grammars constructed from DSAPDA are simple and it will be clear in the sequel why uniqueness imposed by clause 2 is essential.[17]

Definition 40. Let $G = (V, \Sigma, P, S)$ be a simple conjunctive grammar and let $B \in V$ be a split variable. The *degree* of B, denoted $\deg B$ is the degree of the unique proper conjunctive production in P with B in the left side.

In the rest of this section we assume that the grammars under consideration are simple.

Example 41. Grammar G' resulting in adding to grammar G from Example 5 new variables $\overline{S}$, $\widehat{S}$, S_1, and S_2 and productions $\overline{S} \to S$, $\widehat{S} \to S_1$, $\widehat{S} \to S_2$, $S_1 \to S_a S_{bc}$, and $S_2 \to S_{ac}$ is simple.

[16] That is, B becomes a non-split variable, production π disappears, but (10) comes instead.

[17] In ordinary $LR(0)$ grammars, the reason for the requirement for the start variable not to appear in the right side of any production also becomes clear only after acquaintance with the parsing algorithm.

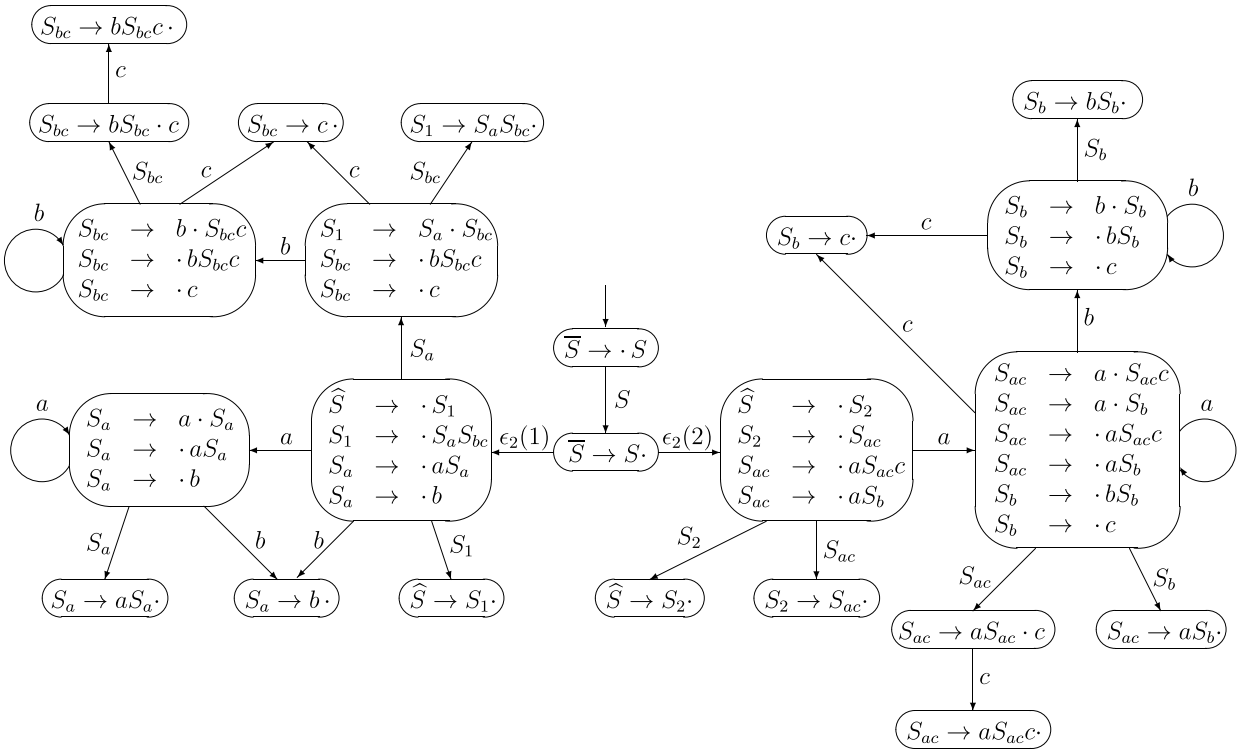

Fig. 3. The canonical collection of item sets $\mathcal{C}_{G'}$ and the *goto* $\boldsymbol{g}$ function for G'. The subscript "2" of ϵ indicates that the degree of the corresponding proper conjunctive production is 2 and the arguments 1 and 2 of ϵ_2 refer to the corresponding conjunct.

At this point we need one more bit of notation.

For a set of items I and an integer $k \geq 2$, we denote by $s_k(I)$ the set of variables B such that I contains an item of the form $A \to \alpha \cdot B\beta$ and $\deg B = k$.

We can now move to the definition of the *goto* function $\boldsymbol{g}$ that is the following extension of the transition function g to $\mathcal{I} \times (V \cup \Sigma \cup \{\epsilon\})$, cf. the classical *goto* function that is defined on $\mathcal{I} \times (V \cup \Sigma)$.

For a set of items $I \in \mathcal{I}$, $\boldsymbol{g}(I, \epsilon)$ consists of all tuples

$$([I_k(1)], [I_k(2)], \ldots, [I_k(k)]), \tag{11}$$

$k = 2, 3, \ldots$, where

$$I_k(i) = \{\widehat{S} \to \cdot B_i : B \in s_k(I)\}$$

$i = 1, 2, \ldots, k$.[18]

Next, we define the canonical collection of item sets of a conjunctive grammar.

Definition 42. Let $G = (V, \Sigma, P, S)$ be a (simple) conjunctive grammar and let $\boldsymbol{g}$ be the *goto* function. We define the *canonical collection of item sets* of G to be the minimal set $\mathcal{C}_G$ containing I_S such that for each $I \in \mathcal{C}_G$ the following holds.

- For all $X \in V \cup \Sigma$, $\boldsymbol{g}(I, X) \in \mathcal{C}_G$, and
- if $(I_1, I_2, \ldots, I_k) \in \boldsymbol{g}(I, \epsilon)$, then $I_i \in \mathcal{C}_G$, $i = 1, 2, \ldots, k$.

Example 43. The canonical collection of item sets $\mathcal{C}_{G'}$ and the *goto* $\boldsymbol{g}$ function for grammar G' from Example 41 is presented in Fig. 3.

4.1.1. Item SAPDA

In this section, we define the item SAPDA of a conjunctive grammar G. For the definition we need the notion of *extended* SAPDA. All components, but the transition function δ, of an extended SAPDA $\boldsymbol{A} = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$ are as in the ordinary SAPDA, but the domain of δ is a finite subset of $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma^+$, rather than $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$. The meaning of the

[18] Note that $\boldsymbol{g}(I, \epsilon)$ depends on the order of conjuncts of (7), see also Remark 57 in the sequel.

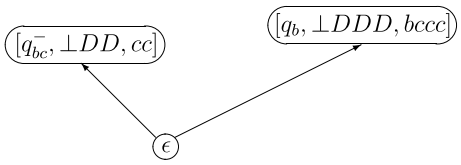

Fig. 4. The configuration from Fig. 1 in the new notation.

extended δ is that in a given state with the given input symbol, the corresponding topmost portion of the stack may be replaced with the second component of a pair from the value of δ. It can be readily seen that such an extension does not increase the computation power, cf. [1, Section 2.5.2].

We shall also represent leaf labels of tree configurations with the top of the stack to the right end, rather than the left. To distinguish the new notation from the old we shall use brackets rather than parentheses: $[q, \gamma, w]$ will be the synonym of (q, w, γ^R).

Example 44. In the new notation, the configuration in Fig. 1 is as in Fig. 4.

Finally we shall use the formula-like representation of automaton configurations, see Definition 45 below.

Definition 45. (Cf. [3, Definition 13].) Let C be a configuration of an SAPDA. The *formula-like representation* $r(C)$ of C is defined by the following recursion.

- If C consists of a single node labeled $[q, \gamma, w]$, then $r(C) = [q, \gamma, w]$.
- If C consists of a root node labeled γ with k sub-trees $C_1, \ldots, C_k$ rooted at the child nodes of the root, then $r(C) = \gamma(r(C_1) \wedge \cdots \wedge r(C_n))$.

Example 46. The formula-like representation of the configuration in Fig. 4 is, after omitting the external parentheses,

$$[q_{bc}^{-}, \bot DD, cc] \wedge [q_b, \bot DDD, bccc]\,.$$

Definition 47. Let $G = (V, \Sigma, P, S)$ be a simple conjunctive grammar. The *item* SAPDA for G is the (extended) SAPDA $\boldsymbol{A}_G = (Q, \Sigma, \Gamma, \delta, q, \bot)$, where

- $Q = \{q\} \cup s(V)$, q is a new symbol and $s(V)$ is the set of all split variables of G,
- $\Gamma = V \cup \Sigma \cup \mathcal{C}_G$,
- $\bot = I_S$, and
- δ is defined as follows.

`shift` For $\sigma \in \Sigma$, $\delta(q, \sigma, I) = \{(q, I\sigma\, g(I, \sigma))\}$, if for some $A \in V$ and some $\alpha, \beta \in (V \cup \Sigma)^*$, $A \to \alpha \cdot \sigma\beta \in I$, and is $\emptyset$, otherwise. That is, if I contains an item $A \to \alpha \cdot \sigma\beta$, $\sigma \in \Sigma$ and the next input symbol is σ, then σ is shifted to the stack, and $\boldsymbol{g}(I, \sigma)$ is written above σ.

`reduce` $\delta(q, \epsilon, I_0 X_1 I_1 X_2 \cdots I_{n-1} X_n I_n) = \{(q, I_0, A, g(I_0, A)) : A \to X_1 X_2 \cdots X_n \cdot \in I_n\}$.

That is, if I contains a complete item $A \to X_1 X_2 \cdots X_n \cdot$, then the X_is and the padding item sets are removed from the stack, revealing some item set I_0 at the top of the stack, A is written above I_0, and $\boldsymbol{g}(I_0, A)$ is written above A.

`split` $\delta(q, \epsilon, I)$ consists of transitions of the form

$$(q, I_k(1)) \wedge (q, I_k(2)) \wedge \cdots \wedge (q, I_k(k))\,,$$

where I_n contains a split item of degree k. That is, k new branches are open, and the ith component of the k-tuple from $\boldsymbol{g}(I, \epsilon)$ is put into the ith branch, $i = 1, 2, \ldots, k$.

`preparation for collapse` $\delta(q, \epsilon, I_k(i) B_i \emptyset) = \{(B, \epsilon)\}$, $i = 1, 2, \ldots, k$. Since B_i does not appear in the right side of any production, the stack is emptied.

`completion of collapse` $\delta(B, \epsilon, I) = (q, IBg(I, B))$. This is the `reduce` action corresponding to the unique production with B in the left side.

`accept` $\delta(q, \epsilon, I_S S \emptyset) = \{(q, \epsilon)\}$. Since S does not appear in the right side of any production, after the `accept` transition, the stack is emptied.

Remark 48. Note that `shift`, `reduce`, `split`, and `accept` are actually the values of the *parsing action function* on $\mathcal{C}_G$ that, together with the *goto* function, forms the parsing table for G, see [1, p. 392].

Let $G(I,i,k) = (V, \Sigma, P_{I,i,k}, \widehat{S})$ be the conjunctive grammar in which $P_{I,i,k}$ results from P in deleting all productions $\widehat{S} \to B_j$ with $B \notin s_k(I)$ or $j \neq i$.[19] The grammar $G(I,i,k)$ is the part of the grammar G used in processing of the ith branch open by the k branch split transition. Thus, the branch stack content of a configuration of $\boldsymbol{A}_G$ is always of the form

$$I_0 X_1 I_1 X_2 I_2 \cdots I_{n-1} X_n I_n\,, \quad n = 0, 1, \ldots\,,$$

where either

- $I_0 = I_S$ and $X_1 X_2 \ldots X_n$ is an S-viable prefix of G and $I_m = g(I_S, X_1 X_2 \cdots X_m)$, $m = 0, 1, \ldots, n$, or
- $I_0 = I_k(i)$ and $X_1 X_2 \ldots X_n$ is an $\widehat{S}$-viable prefix of $G(I,i,k)$ and $I_m = g(I_k(i), X_1 X_2 \cdots X_m)$, $m = 0, 1, \ldots, n$.

We shall prove that $L(\boldsymbol{A}_G) = L(G)$. The proof is an extension of the classical one and is based on Propositions 50 and 51 below. These propositions employ the following notation.

For an S'-viable prefix, $S' = S, \widehat{S}$, $\gamma = X_1 X_2 \cdots X_n \in (V \cup \Sigma)^*$, the stack content $I_0 X_1 I_1 X_2 I_2 \cdots I_{n-1} X_n I_n \in \Gamma^*$ is denoted by γ^I.[20]

Example 49. In the above notation, in `preparation for collapse`, $I_k(i) B_i \emptyset$ is B_i^I, where $I_0 = I_k(i)$, and, in `accept`, $I_S S \emptyset$ is just S^I.

Proposition 50. *Let $S' = S, \widehat{S}$ and let γ be an S'-viable prefix. If*

$$S' \Rightarrow_R^* \gamma B v \Rightarrow_R^+ u v\,,$$

then

$$[q, I_{S'}, u] \vdash^+ [q, (\gamma B)^I, \epsilon]\,, \tag{12}$$

where

- *the derivation is in G, if $S' = S$, and*
- *the derivation is in some $G(I,i,k)$, if $S' = \widehat{S}$, in which case $I_{S'} = I_k(i)$.*

Proposition 51. *If*

$$[q, \gamma'^I, u] \vdash^* [q, \gamma''^I, \epsilon]\,, \tag{13}$$

then

$$\gamma'' \Rightarrow^* \gamma' u\,. \tag{14}$$

Before proving the propositions we state their immediate corollary.

Corollary 52. $L(\boldsymbol{A}_G) = L(G)$.

Proof. Assume $S \Rightarrow^* w$. That is

$$S \Rightarrow^* \epsilon S \epsilon \Rightarrow^* \epsilon w\,,$$

implying, by Lemma 34,

$$S \Rightarrow^* \epsilon S \epsilon \Rightarrow_R^* \epsilon w\,.$$

By Proposition 50, $[q, I_S, w] \vdash^* [q, S^I, \epsilon]$ and, continuing the computation by `accept`, we obtain $w \in L(\boldsymbol{A}_G)$.

Conversely, assume $[q, I_S, w] \vdash^* [q, \epsilon, \epsilon]$. Then the last transition in the computation is by `accept`. That is,

$$[q, I_S, w] \vdash^* [q, S^I, \epsilon] \vdash [q, \epsilon, \epsilon]$$

and, by Proposition 51, $S \Rightarrow^* w$, because $I_S = \epsilon^I$. □

[19] Equivalently, $\widehat{S} \to B_j \in P_{I,i,k}$ if and only if $B \in s_k(I)$ and $j = i$.

[20] Note that the stack content is always of the form γ^I for some S'-viable prefix γ.

Proof of Proposition 50. The proof is by double induction: the outer induction is on the number m of proper conjunctive productions in derivation $\gamma Bv \Rightarrow_R^* uv$ and the inner induction is on the length n of that derivation.

Outer induction basis: $m = 0$. We shall prove by induction on n that

$$S' \Rightarrow_R^* \gamma Bv \Rightarrow_R^n uv \tag{15}$$

implies (12). The proof is similar to the proof of the first part of [1, Lemma 2.25].

Inner induction basis: $n = 1$. Then

$$S' \Rightarrow_R^* \gamma Bv \Rightarrow \gamma\beta v = uv\,.$$

That is, $\gamma\beta \in \Sigma^*$. Let $\beta = \sigma_1\sigma_2\cdots\sigma_\ell \in \Sigma^*$. Then $u = \gamma\sigma_1\sigma_2\cdots\sigma_\ell$ and we have

$$[q, I_S, u] \vdash^* [q, \gamma^I\sigma_1 I_1\sigma_2 I_2\cdots I_{\ell-1}\sigma_\ell I_\ell, \epsilon] \vdash [q, \gamma^I BI, \epsilon] = [q, (\gamma B)^I, \epsilon]\,,$$

where all transitions, but the last one, are by `shift` and the last transition is by `reduce` with $B \to \beta$. Since u is an S'-viable prefix, there are such `shift` transitions.

Inner induction step: Let $n \geq 2$ and let

$$S' \Rightarrow_R^* \gamma Bv \Rightarrow_R \gamma\beta v \Rightarrow_R^{n-1} uv\,,$$

where $\beta = \beta' B' w$ and $w = \sigma_1\sigma_2\cdots\sigma_\ell \in \Sigma^*$. Then $\gamma\beta = \gamma\beta' B' w \notin \Sigma^*$ and

$$S' \Rightarrow_R^* \gamma Bv \Rightarrow \gamma\beta' B' wv \Rightarrow_R^{n-1} u'wv = u'\sigma_1\sigma_2\cdots\sigma_\ell v = uv\,. \tag{16}$$

That is, $u = u'w$, implying

$$\begin{array}{lcl}
[q, I_S, u] & = & [q, I_S, u'w] \\
& \vdash^* & [q, (\gamma\beta' B')^I, w] \\
& = & [q, (\gamma\beta' B')^I, \sigma_1\sigma_2\cdots\sigma_\ell] \\
& \vdash^\ell & [q, (\gamma\beta' B')^I\sigma_1 I_1\sigma_2 I_2\cdots I_{\ell-1}\sigma_\ell I_\ell, \epsilon]\,, \\
& = & [q, (\gamma\beta' B' w)^I, \epsilon] \\
& \vdash & [q, (\gamma B)^I, \epsilon]
\end{array}$$

where the first sequence of transitions is by the induction hypothesis for $n-1$, the second is by ℓ `shift`s, and the last transition is by `reduce` with $B \to \beta = \beta' B' w$. Of course, the `shift` transitions come from the derivation step $\gamma Bv \Rightarrow \gamma' B' wv$ of (16).

Outer induction step: Assume that the implication is true for all m' less than m. We shall prove by induction on $n \geq 1$ that (15) implies (12).

If the first step in derivation $\gamma Bv \Rightarrow_R^n uv$ is by an ordinary production, the proof is exactly as in the outer induction basis $m = 0$.

Assume that the first step in derivation $\gamma Bv \Rightarrow_R^n uv$ is by a proper conjunctive production. Then (15) is of the form

$$S' \Rightarrow_R^* \gamma Bv \Rightarrow_R \gamma(\alpha_1 \,\&\, \cdots \,\&\, \alpha_k)v \Rightarrow_R^* \gamma(w \,\&\, \cdots \,\&\, w)v \Rightarrow_R \gamma wv \Rightarrow_R^* u'wv\,,$$

where $k \geq 2$, $\alpha_i \Rightarrow_R^* w$ by at most $m-1$ proper conjunctive productions, and $u = u'w$.

We shall distinguish between the cases $\gamma \in \Sigma^*$ and $\gamma \notin \Sigma^*$.

Assume $\gamma \in \Sigma^*$. Then $u = \gamma w$, implying

$$\begin{array}{lcl}
[q, I_{S'}, u] & = & [q, I_{S'}, \gamma w] \\
& \vdash^* & [q, \gamma^I, w] \\
& \vdash^* & \gamma^I([q, I_k(1), w] \wedge [q, I_k(2), w] \wedge \cdots \wedge [q, I_k(k), w]) \\
& \vdash^* & \gamma^I([q, I_k(1)B_1\emptyset, \epsilon] \wedge [q, I_k(2)B_2\emptyset, \epsilon] \wedge \cdots \wedge [q, I_k(k)B_k\emptyset, \epsilon])\,, \\
& \vdash^k & \gamma^I([B, \epsilon, \epsilon] \wedge [B, \epsilon, \epsilon] \wedge \cdots \wedge [B, \epsilon, \epsilon]) \\
& \vdash & [B, \gamma^I, \epsilon] \\
& \vdash & [q, (\gamma B)^I, \epsilon]
\end{array}$$

where the first sequence of transitions is by `shift`; the second is by `split`, because $B \in s_k(g(I_{S'}, \gamma))$; the third is by the outer induction hypothesis; the forth is by `preparation for collapse`; the fifth is by `collapse`; and the last is by

`completion of collapse`. Similarly to the outer induction basis, since γ is an S'-viable prefix, there are such `shift` transitions.

Assume $\gamma = \gamma' B' w' \notin \Sigma^*$. Then $u = u'w'w$, where $\gamma' B' \Rightarrow_R^* u'$, implying

$$\begin{aligned}
[q, I_{S'}, u] &= [q, I_{S'}, u'w'w] \\
&\vdash^* [q, (\gamma' B')^I, w'w] \\
&\vdash^* [q, (\gamma' B' w')^I, w] \\
&\vdash^* (\gamma' B' w')^I ([q, I_k(1), w] \wedge [q, I_k(2), w] \wedge \cdots \wedge [q, I_k(k), w]) \\
&\vdash^* (\gamma' B' w')^I ([q, I_k(1) B_1 \emptyset, \epsilon] \wedge [q, I_k(2) B_2 \emptyset, \epsilon] \wedge \cdots \wedge [q, I_k(k) B_k \emptyset, \epsilon]) \ , \\
&\vdash^k (\gamma' B' w')^I ([B, \epsilon, \epsilon] \wedge [B, \epsilon, \epsilon] \wedge \cdots \wedge [B, \epsilon, \epsilon]) \\
&\vdash [B, (\gamma' B' w')^I, \epsilon] \\
&\vdash [q, (\gamma' B' w' B)^I, \epsilon] \\
&= [q, (\gamma B)^I, \epsilon]
\end{aligned}$$

where the first sequence transitions is by the outer induction hypothesis; the second is by `shifts`; the third is by `split`, because $B \in s_k(g(I_{S'}, \gamma))$; the forth is by the outer induction hypothesis; the fifth is by `preparation for collapse`; the sixth is by `collapse`; and the last is by `completion of collapse`. Such `shift` transitions exist, because $\gamma' B' w'$ is an S'-viable prefix. □

Proof of Proposition 51. The proof is by double induction: the outer induction is on the number m of proper conjunctive transitions in computation (13) and the inner induction is on the length n of that computation.

Outer induction basis: $m = 0$. We shall prove by induction on n that

$$[q, \gamma'^I, u] \vdash^n [q, \gamma''^I, \epsilon] \tag{17}$$

implies (14). The proof is similar to the proof of the second part of [1, Lemma 2.25].

Inner induction basis: $n = 0$. Then $\gamma' = \gamma''$ and $u = \epsilon$, and (14) follows immediately.

Inner induction step: Let $n \geq 2$. Then (17) is of the form

$$[q, \gamma'^I, u] \vdash^{n-1} [q, \gamma^I, w] \vdash [q, \gamma''^I, \epsilon] , \tag{18}$$

where $u = u'w$. Therefore, $[q, \gamma'', u'] \vdash^{n-1} [q, \gamma^I, \epsilon]$ and, by the induction hypothesis for $n - 1$,

$$\gamma \Rightarrow^* \gamma' u' . \tag{19}$$

If the last transition of (18) is by `shift`, then $w \in \Sigma$ and $\gamma'' = \gamma w$, which, together with (19), implies

$$\gamma'' = \gamma w \Rightarrow^* \gamma' u' w = \gamma' u .$$

If the last transition of (18) is by `reduce`, then $w = \epsilon$, implying $u = u'$, $\gamma = \widetilde{\gamma}\alpha$, and $\gamma'' = \widetilde{\gamma} B$, for some viable prefix $\widetilde{\gamma}$ and some production $B \to \alpha$.

This, together with (19), implies

$$\gamma'' = \widetilde{\gamma} B \Rightarrow \widetilde{\gamma}\alpha = \gamma \Rightarrow^* \gamma' u' = \gamma' u .$$

Outer induction step: Let $m \geq 1$ and assume that the implication is true for all m' less than m. Since $m \geq 1$, (13) is of the form

$$\begin{aligned}
[q, \gamma_1^I, u_1 u_2 u_3] &\vdash^* [q, \gamma_2^I, u_2 u_3] \\
&\vdash \gamma_2^I ([q, g(\gamma_2)_k(1), u_2 u_3] \wedge \cdots \wedge [q, g(\gamma_2)_k(k), u_2 u_3]) \\
&\vdash^* \gamma_2^I ([q, g(\gamma_2)_k(1) B_1 \emptyset, u_3] \wedge \cdots \wedge [q, g(\gamma_2)_k(k) B_k \emptyset, u_3]) \\
&\vdash^k \gamma_2^I ([B, \epsilon, u_3] \wedge \cdots \wedge [B, \epsilon, u_3]) \qquad , \\
&\vdash [B, \gamma_2^I, u_3] \\
&\vdash [q, (\gamma_2 B)^I, u_3] \\
&\vdash^* [q, \gamma_3^I, \epsilon]
\end{aligned}$$

where $u = u_1 u_2 u_3$, $\gamma' = \gamma_1$, $\gamma'' = \gamma_3$, and the proper conjunctive transition comes from a degree-k production with B in the left side.

By the outer induction hypothesis, $\gamma_2 \Rightarrow^* \gamma_1 u_1$; $B_i \Rightarrow^* u_2$, $i = 1, 2, \ldots, k$, implying $B \Rightarrow^* u_2$; and $\gamma_3 \Rightarrow^* \gamma_2 B u_3$. Combining these derivations, we obtain

$$\gamma_3 \Rightarrow^* \gamma_2 B u_3 \Rightarrow^* \gamma_2 u_2 u_3 \Rightarrow^* \gamma_1 u_1 u_2 u_3 = \gamma_1 u\,.^{21} \qquad \square$$

Remark 53. If at some stage of a computation both `shift` and `reduce` transitions are possible and the automaton proceeds by `shift`, it will be never able to reduce by the handle of the "skipped" `reduce` transition. This is, because after shifts and possible splits or subsequent reductions, the handle will be never exposed at the top of the stack. A similar argument holds for the `split-reduce` combination. Of course, such situations do not happen for a deterministic item automaton.

4.2. LR(0) conjunctive grammars

At last, we can define $LR(0)$ conjunctive grammars.

Definition 54. A set of items I is *conflict free* if it satisfies the following conditions.

1. If I is split, then
   (a) there is an integer $k \geq 2$ such that for all split items $A \to \alpha \cdot B\beta \in I$, $\deg B = k$ (in which case we also write $\deg I = k$),
   (b) I contains no complete item, and
   (c) contains no item of the form $A \to \beta \cdot \sigma\gamma$, where $\sigma \in \Sigma$.
2. If I is ordinary and contains a complete item, then
   (a) it contains no other complete item and
   (b) contains no item of the form $A \to \beta \cdot \sigma\gamma$, where $\sigma \in \Sigma$.

Clause 1(a) prevents *split–split conflicts*, clause 1(b) prevents *split–reduce conflicts*, clause 1(c) prevents *split–shift conflicts*, clause 2(a) prevents *reduce–reduce conflicts*, and clause 2(b) prevents *shift–reduce conflicts*.

Definition 55. A conjunctive grammar G is $LR(0)$ if all elements of $\mathcal{C}_G$ are conflict free.[22]

Example 56. Grammar G' from Example 41 is $LR(0)$.

Remark 57. Note that $\mathcal{C}_G$ depends on the ϵ-transitions of $\boldsymbol{g}$ which, in turn, depend on the order of conjuncts in (7). This is because there may be another grouping of the item sets $I_k(i)$ in (11) for which one of the subsequent item sets has a conflict.

Theorem 58. *If a language is generated by an $LR(0)$ conjunctive grammar, then it is accepted by a DSAPDA.*

Proof. If G is an $LR(0)$ conjunctive grammar, then item SAPDA $\boldsymbol{A}_G$ is deterministic, and the theorem follows from Corollary 52. $\square$

Corollary 59. *$LR(0)$ conjunctive languages can be parsed in linear time.*

Proof. The corollary immediately follows from Theorems 15 and 58. $\square$

Note that when applied to ordinary $LR(0)$ grammars, the parsing algorithm is identical to the classical $LR(0)$ algorithm.

We conclude this section with the following observation. Even though both classical and conjunctive $LR(0)$ languages are not closed under complementation (because the prefix property is not maintained) and are not closed under union, our linear parser can be applied to parse the languages in the boolean closure of conjunctive $LR(0)$ languages, cf. [11, Theorem 4].

A parsing algorithm is by recursion on the complexity of a boolean combination. The boundary case is Corollary 59, and the recursion itself is immediate: $w \in \overline{L}$ if and only if $w \notin L$, $w \in L' \cup L''$ if and only if $w \in L'$ or $w \in L''$, and $w \in L' \cap L''$ if and only if $w \in L'$ and $w \in L''$. That is, the algorithm actually consists of several parsing runs, one for each language in the boolean combination.

[21] In fact, this derivation is rightmost.
[22] Recall that G is simple.

5. Constructing an *LR*(0) grammar from a DSAPDA

In this section we address the converse of Theorem 58, i.e., the construction of an $LR(0)$ conjunctive grammar from a DSAPDA.

Theorem 60. *If a language is accepted by a DSAPDA, then it is generated by an $LR(0)$ conjunctive grammar.*

For the proof of the theorem we make the following simplifying assumption on the structure of a DSAPDA $\boldsymbol{A} = (Q, \Sigma, \Gamma, \delta, r_0, \bot)$, cf. [3, Section 4.2].

1. Proper conjunctive transitions are all ϵ-transitions, and write exactly one symbol into each branch, i.e., they are of the form $\delta(q, \epsilon, X) = (q_1, Y_1) \wedge \cdots \wedge (q_k, Y_k)$.
2. For no two conjuncts (q_i, Y_i) and (q_j, Y_j) of a proper conjunctive transition $(q_1, Y_1) \wedge \cdots \wedge (q_k, Y_k)$, $(q_i, Y_i) = (q_j, Y_j)$, unless $i = j$.
3. The first move of the automaton is not by a proper conjunctive transition.
4. For all proper conjunctive transitions $(q_1, Y_1) \wedge \cdots \wedge (q_k, Y_k)$, $\delta(q_i, \epsilon, Y_i)$ is *not* a proper conjunctive transition, $i = 1, 2, \ldots, k$.[23]

This assumption can be easily shown not to be limiting.
Consider the conjunctive grammar $G_{\boldsymbol{A}} = (V, \Sigma, P, S)$, where

- V consists of
  1. the start variable S,
  2. a new branch start variable $\widehat{S}$,
  3. $[q, X, p]$ for $q, p \in Q$ and $X \in \Gamma$,
  4. $[q, X, p]_i$, $i = 1, 2, \ldots, k$, for $q, p \in Q$, $X \in \Gamma$, and a proper conjunctive transition $\delta(q, \epsilon, X)$ of degree k, and
  5. $A_{q,\sigma,X}$ for $q \in Q$, $\sigma \in \Sigma \cup \{\epsilon\}$, and $X \in \Gamma$;

  and
- the productions of $G_{\boldsymbol{A}}$ are as follows.
  1. $S \to [q_0, \bot, p]$, for all $p \in Q$.
  2. $\widehat{S} \to [q, X, p]_i$, $i = 1, 2, \ldots, k$, for all proper conjunctive transitions $\delta(q, \epsilon, X)$ of degree k, $k = 2, 3, \ldots$, and all $p \in Q$.
  3. If
  $$\delta(q, \epsilon, X) = (q_1, Y_1) \wedge \cdots \wedge (q_k, Y_k),$$
  $k \geq 2$, then for all $p \in Q$,
  (*a*) there is a production
  $$[q, X, p] \to (A_{q,\epsilon,X}[q_1, Y_1, p] \,\&\, \cdots \,\&\, A_{q,\epsilon,X}[q_k, Y_k, p])$$
  and
  (*b*) for all $i = 1, 2, \ldots, k$, there is a production
  $$[q, X, p]_i \to A_{q,\epsilon,X}[q_i, Y_i, p].$$
  4. If
  $$\delta(q, \sigma, X) = (p_1, X_1 X_2 \cdots X_m)$$
  for $m \geq 0$, then for all sequences of states $p_2, p_3, \ldots, p_{m+1}$, there is a production
  $$[q, X, p_{m+1}] \to A_{q,\sigma,X}[p_1, X_1, p_2][p_2, X_2, p_3] \cdots [p_m, X_m, p_{m+1}].$$
  5. For all $q \in Q$, $\sigma \in \Sigma \cup \{\epsilon\}$, and $X \in \Gamma$, there is a production $A_{q,\sigma,X} \to \sigma$.

Note that each $A_{q,\sigma,Y}$ variable corresponds to the specific automaton transition from q on σ with X at the top of the stack. This construction is very similar to the one in [5, pp. 256–260], and is essentially a modification of the translation of an SAPDA into a conjunctive grammar, see [3, Section 4.2]. The main difference is the addition of the $A_{q,\sigma,X}$ variables, needed for the proof that the grammar is $LR(0)$.

For the proof of Theorem 60 we have to show that $G_{\boldsymbol{A}}$ is $LR(0)$ and that $L(\boldsymbol{A}) = L(G_{\boldsymbol{A}})$. The latter follows from the construction in [3, Section 4.2], and, for the former, we shall show that the canonical collection of sets of items C_{G_A} is conflict-free.[24] We break the proof into three parts.

[23] Loosely speaking, there are no two consecutive moves each by a proper conjunctive transition.
[24] Note that, by definition, $G_{\boldsymbol{A}}$ is simple.

Proposition 61. *Split sets of items in C_{G_A} are conflict-free.*

Proposition 62. *If an ordinary set of items from C_{G_A} contains a complete item, then it does not contain an item of the form $A_{r,\tau,Z} \to \cdot\, \tau$, where $\tau \in \Sigma$.*

Proposition 63. *Each ordinary set of items from C_{G_A} contains at most one complete item.*

Actually the proofs that there are no shift–reduce or reduce–reduce conflicts in ordinary (i.e., non-split) sets of items are very similar to those in [5, Section 10.7] and are presented here only for the sake of completeness.

Proof of Proposition 61. Let $I \in C_{G_A}$ be a split set of items. By simplifying assumptions 3 and 4 in the beginning of this section, items of the form $S \to \cdot\,[q_0, \bot, p]$ and $\widehat{S} \to \cdot\,[q, X, p]_i$ are not split. Also, by the definition of $G_{\boldsymbol{A}}$, variables $A_{q,\sigma,Y}$ are not split. Therefore, $I = \boldsymbol{g}(J, W)$, for an item set J and $W \in V$.[25]

Let ι be a split item in I. Since variables $A_{q,\sigma,Y}$ are not split, ι is of the form

$$[q, X, p_{m+1}] \to A_{q,\sigma,X}[p_1, X_1, p_2]\cdots[p_{j-1}, X_{j-1}, p_j] \cdot [p_j, X_j, p_{j+1}]\cdots[p_m, X_m, p_{m+1}]\,.$$

That is, the production with $[p_j, X_j, p_{j+1}]$ in the left side is of the form

$$[p_j, X_j, p_{j+1}] \to (A_{p_j,\sigma,X_j}[q_1, Y_1, p_{j+1}] \,\&\, \cdots \,\&\, A_{p_j,\sigma,X_j}[q_k, Y_k, p_{j+1}])\,,$$

where

$$\delta(p_j, \epsilon, X_j) = (q_1, Y_1) \wedge \cdots \wedge (q_k, Y_k)\,, \quad k \geq 2\,. \tag{20}$$

Next, $I = \boldsymbol{g}(J, W)$ implies that $I = g(I_{S'}, \gamma)$, for $S' = S$ or $S' = \widehat{S}$, and some S'-viable prefix γ. Therefore,

$$\gamma = \gamma' A_{q,\sigma,X}[p_1, X_1, p_2]\cdots[p_{j-1}, X_{j-1}, p_j]\,.$$

Since $\boldsymbol{A}$ is deterministic, because of $A_{q,\sigma,X}$, $I = [I']$, where I' is the set of items of the form

$$[q, X, p_{m+1}] \to A_{q,\sigma,X}[p_1, X_1, p_2]\cdots[p_{j-1}, X_{j-1}, p_j] \cdot [p_j, X_j, p'_{j+1}]\cdots[p_m, X_m, p'_{m+1}]\,,$$

$p'_{j+1}, \ldots, p'_{m+1} \in Q$.

By (20), for all states $p'_{j+1} \in Q$, productions with $[p_j, X_j, p'_{j+1}]$ in the left side are of the form

$$[p_j, X_j, p'_{j+1}] \to (A_{p_j,\epsilon,X_j}[q_1, Y_1, p'_{j+1}] \,\&\, \cdots \,\&\, A_{p_j,\epsilon,X_j}[q_k, Y_k, p'_{j+1}])\,.$$

Therefore, all variables

$$[p_j, X_j, p'_{j+1}]\,, \quad p'_{j+1} \in Q\,, \tag{21}$$

are split and are of the same degree k. Thus, $I = [I'] = I'$, implying that there are no split–reduce or split–shift conflicts. Since all variables (21) are of the same degree k, there are no split–split conflicts either. □

Remark 64. It follows from the proof of Proposition 61 that, for a split set of items $I \in \mathcal{C}_{G_A}$, there are $p \in Q$ and $X \in \Gamma$ such that $I_k(i)$ is of the form

$$I_k(i) = \{\widehat{S} \to [p, X, q]_i : q \in Q\}\,,$$

where $\deg I = k$ and $i = 1, 2, \ldots, k$. Namely, in the proof of Proposition 61, p is p_j, X is X_j, and q is p'_{j+1}.

Proof of Proposition 62. Let $I \in C_{G_A}$ be an ordinary set of items and let $I = g(I_{S'}, \gamma)$, for $S' = S, \widehat{S}$, and some S'-viable prefix γ. Let $B \to \beta\cdot$ be a complete item from I. We shall distinguish among the following cases.

(i) $B \to \beta$ is of type 1.
(ii) $B \to \beta$ is of type 2.
(iii) $B \to \beta$ is of type 3(b).
(iv) $B \to \beta$ is of type 4.
(v) $B \to \beta$ is of type 5.

[25] This is because in a right-sentential form of $G_{\boldsymbol{A}}$, a variable cannot be preceded by a terminal.

Case (i) Assume that for some $p \in Q$, $B \to \beta$ is $S \to [q_0, \bot, p]$. Since S does not appear on the right side of any production,

$$\gamma = \beta = [q_0, \bot, p] .$$

If $A_{r,\tau,Z} \to \cdot\, \tau$ is valid for $[q_0, \bot, p]$, then, for some $w \in \Sigma^*$, there is a derivation

$$S \Rightarrow_R^* [q_0, \bot, p] A_{r,\tau,Z} w \Rightarrow_R [q_0, \bot, p] \tau w .$$

However, at the second step of this derivation, $[q_0, \bot, p]$ must be replaced by a word beginning with a subscripted A. That is, it cannot stay intact till the end the derivation.

Case (ii) The proof is similar to that of case (i). We just substitute $\widehat{S}$ for S, $[q, X, p]_i$ for $[q_0, \bot, p]$, and "third" for "second."

Case (iii) Again, the proof is similar to that of case (i). Assume that for some proper conjunctive transition

$$\delta(q, \epsilon, X) = (q_1, Y_1) \wedge \cdots \wedge (q_k, Y_k)$$

some $p \in Q$, and some $i = 1, 2, \ldots, k$, $B \to \beta$ is $[q, X, p]_i \to A_{q,\epsilon,X}[q_i, Y_i, p]$.

Since $[q, X, p]_i$ does not appear on the right side of any production,

$$\gamma = \beta = A_{q,\epsilon,X}[q, Y_i, p] .$$

If $A_{r,\tau,Z} \to \cdot\, \tau$ is valid for $A_{q,\epsilon,X}[q, Y_i, p]$, then, for some $w \in \Sigma^*$, there is a derivation

$$\widehat{S} \Rightarrow_R [q, X, p]_i \Rightarrow_R A_{q,\epsilon,X}[q, Y_i, p] \Rightarrow_R^* A_{q,\epsilon,X}[q, Y_i, p] A_{r,a,Z} w \Rightarrow_R A_{q,\epsilon,X}[q, Y_i, p] \tau w .$$

However, at the third step of this derivation, $[q, Y_i, p]$ must be replaced by a word beginning with a subscripted A. That is, it cannot stay intact till the end the derivation.

Case (iv) Assume that for some transition

$$\delta(q, \sigma, X) = (p_1, X_1 X_2 \cdots X_m)$$

and for some sequence of states $p_2, p_3, \ldots, p_{m+1}$, $B \to \beta$ is

$$[q, X, p_{m+1}] \to A_{q,\sigma,X}[p_1, X_1, p_2][p_2, X_2, p_3] \cdots [p_m, X_m, p_{m+1}] .$$

Then $\gamma = \gamma' \beta$ for some $\gamma' \in V^*$. However, in any rightmost derivation, after production $B \to \beta$ of type 4 is applied, the last symbol of β is immediately expanded by rules productions of type $3(a)$, 4, or 5, so β could not appear intact in an ordinary S'-right-sentential form followed by $A_{r,\tau,Z}$.

Case (v) Assume that for some $q \in Q$, some $\sigma \in \Sigma \cup \{\epsilon\}$, and some $Z \in \Gamma$, $B \to \beta$ is $A_{p,\sigma,X} \to \sigma$. Then σ must be ϵ, else $\gamma A_{r,\tau,Z}$ which is a viable prefix, has a terminal σ in it. As $A_{p,\epsilon,X} \to \cdot$ is valid for γ, it follows that $\gamma A_{p,\epsilon,X}$ is a viable prefix.

That is, $S' \Rightarrow_R^* \gamma A_{p,\epsilon,X} w$ and $S' \Rightarrow_R^* \gamma A_{r,\tau,Z} w'$, for some $w, w' \in \Sigma^*$. We shall treat the cases of $\gamma \neq \epsilon$ and $\gamma = \epsilon$ separately.

Let $\gamma \neq \epsilon$. Then

$$\gamma = \gamma' A_{r',\rho,X'}[r_1, X_1, r_2][r_2, X_2, r_3] \cdots [r_i, X_i, r_{i+1}]$$

where $\delta(r', \rho, X') = (r_1, X_1 X_2 \cdots X_m)$, $m \geq i$ and $r_1, r_2, \ldots, r_{i+1} \in Q$. Therefore, for some $r_{i+1}, r_{i+2}, \ldots, r_{m+1} \in Q$.

$$[r_{i+1}, X_{i+1}, r_{i+2}][r_{i+2}, X_{i+2}, r_{i+3}] \cdots [r_m, X_m, r_{m+1}] \Rightarrow_R^* A_{p,\epsilon,X} w$$

and

$$[r_{i+1}, X_{i+1}, r_{i+2}][r_{i+2}, X_{i+2}, r_{i+3}] \cdots [r_m, X_m, r_{m+1}] \Rightarrow_R^* A_{r,\tau,Z} w' .$$

Thus, $p = r = r_{i+1}$ and $X = Z = X_{i+1}$, implying that both $\delta(r_{i+1}, \epsilon, X_{i+1})$ and $\delta(r_{i+1}, \tau, X_{i+1})$ are nonempty. This, however, contradicts the determinism of $\boldsymbol{A}$.

Let $\gamma = \epsilon$. Since $\tau \in \Sigma$, $A_{r,\tau,Z}$ cannot appear in productions of type $3(b)$ and it follows that $S' = S$. Therefore,

$$S \Rightarrow [q_0, \bot, s] \Rightarrow A_{p,\epsilon,X}[r_1, X_1, r_2][r_2, X_2, r_3] \cdots [r_k, X_k, s] \Rightarrow_R^* A_{p,\epsilon,X} w$$

and

$$S \Rightarrow [q_0, \bot, s'] \Rightarrow A_{r,\tau,Z}[r'_1, X'_1, r'_2][r'_2, X'_2, r'_3] \cdots [r'_{k'}, X_{k'}, s'] \Rightarrow_R^* A_{r,\tau,Z} w' .$$

Thus, $p = r = q_0$ and $X = Z = \bot$, implying that both $\delta(q_0, \epsilon, \bot)$ and $\delta(q_0, \tau, \bot)$ are nonempty, which, again, contradicts the determinism of $\boldsymbol{A}$. □

Proof of Proposition 63. Let $I \in C_{G_A}$ be an ordinary set of items and let $I = g(I_{S'}, \gamma)$, for $S' = S, \widehat{S}$, and some S'-viable prefix γ. Assume to the contrary that I contains two different complete items $A \to \alpha\cdot$ and $B \to \beta\cdot$. Then both α and β are suffixes of γ. We shall distinguish among the following cases (each having a number of subcases) arriving at contradiction in each case.

(i) Neither $A \to \alpha\cdot$ nor $B \to \beta\cdot$ is of type 5.
(ii) Both $A \to \alpha\cdot$ and $B \to \beta\cdot$ are of type 5.
(iii) $A \to \alpha$ is of type 5, but $B \to \beta$ is of type 1, 2, 3(b), or 4, or vice versa.

Case (i) There are the following four subcases.

(a) One of the productions, $A \to \alpha$, say, is of type 1,
(b) none of the productions is of type 1, but one of them, $A \to \alpha$, say, is of type 2,
(c) none of the productions is of type 1 or 2, but one of them, $A \to \alpha$, say, is of type 3(b), and
(d) both productions are of type 4.

Subcase (a) In this subcase $A = S$ and

$$\gamma = \alpha = \beta = [q_0, \bot, p]\,,$$

for some $p \in Q$. Since, in a derivation from S, right-sentential forms not containing a subscripted A or a terminal can appear only at the first step of the derivation, $B = A = S$.

Subcase (b) In this subcase $A = \widehat{S}$ and

$$\gamma = \alpha = \beta = [q, X, p]_i\,,$$

for some $p, q \in Q$ and some positive integer i. Since ordinary right-sentential forms $[q, X, p]_i$ can appear only at the first step of a derivation, $B = A = \widehat{S}$.

Subcase (c) In this subcase, for some production

$$[q, X, p] \to (A_{q,\epsilon,X}[q_1, Y_1, p] \,\&\, \cdots \,\&\, A_{q,\epsilon,X}[q_k, Y_k, p])$$

and some $i = 1, 2, \ldots, k$, $A \to \alpha$ is $[q, X, p]_i \to A_{q,\epsilon,X}[q_i, Y_i, p]$.

Thus,

$$\gamma = A_{q,\epsilon,X}[q_i, Y_k, p]) = \alpha\,.$$

We contend that $\beta = \gamma$ as well. Indeed, since β is a suffix of γ, $\beta = \gamma$, $\beta = [q_i, Y_k, p]$, or $\beta = \epsilon$. The last equality is impossible, because $B \to \beta$ is not of type 5, and the second equality is impossible, because $B \to \beta$ is not of type 1.

Thus, $B \to \beta$ is $[q, X, p]_j \to A_{q,\epsilon,X}[q_i, Y_i, p]$. By simplifying assumption 2 at the beginning of this section, $i = j$ and the equality of $A \to \alpha$ and $B \to \beta$ follows.

Subcase (d) In this case, $\alpha = \beta$, because of the form of productions of type 4. Thus, because of the subscripted A in the corresponding productions, $A = B$ as well.

Case (ii) The proof is similar to that of case (v) Proposition 62. Let $A \to \alpha$ and $B \to \beta$ be $A_{q',\sigma',Y'} \to \sigma'$ and $A_{q'',\sigma'',Y''} \to \sigma''$, respectively. There are the following three subcases.

(a) Both σ' and σ'' are ϵ,
(b) $\sigma' \neq \epsilon$, but $\sigma'' = \epsilon$ or vice versa, and
(c) both σ' and σ'' are not ϵ.

Subcase (a) We have, $S' \Rightarrow_R^* \gamma A_{q',\epsilon,Y'} w'$ and $S' \Rightarrow_R^* \gamma A_{q'',\epsilon,Y''} w''$, for some $w', w'' \in \Sigma^*$. Like in case (v) Proposition 62, we shall treat the cases of $\gamma \neq \epsilon$ and $\gamma = \epsilon$ separately.

Let $\gamma \neq \epsilon$. Then

$$\gamma = \gamma' A_{p,\tau,X}[p_1, X_1, p_2][p_2, X_2, p_3]\cdots[p_i, X_i, p_{i+1}]$$

where $\delta(p, \tau, X) = (p_1, X_1 X_2 \cdots X_m)$, $m \geq i$ and $p_1, p_2, \ldots, p_{i+1} \in Q$. Therefore, for some $p'_{i+2}, p'_{i+3}, \ldots, p'_{m+1} \in Q$,

$$[p_{i+1}, X_{i+1}, p'_{i+2}][p'_{i+2}, X_{i+2}, p'_{i+3}]\cdots[p'_m, X_m, p'_{m+1}] \Rightarrow_R^* A_{q',\epsilon,Y'} w'$$

and for some $p''_{i+2}, p''_{i+3}, \ldots, p''_{m+1} \in Q$,

$$[p_{i+1}, X_{i+1}, p''_{i+2}][p''_{i+2}, X_{i+2}, p''_{i+3}]\cdots[p''_m, X_m, p''_{m+1}] \Rightarrow_R^* A_{q'',\epsilon,Y''} w''\,.$$

Thus, $q' = q'' = p_{i+1}$ and $Y' = Y'' = X_{i+1}$, contradicting our assumption that $A_{q',\epsilon,Y'} \to \cdot$ and $A_{q'',\epsilon,Y''} \to \cdot$ are different.

Let $\gamma = \epsilon$. If $S' = S$, then

$$S \Rightarrow [q_0, \bot, s'] \Rightarrow A_{q',\epsilon,Y'}[p'_1, X'_1, p'_2][p'_2, X'_2, p'_3]\cdots[p'_{m'}, X'_{m'}, s'] \Rightarrow^*_R A_{q',\epsilon,Y'}w'$$

and

$$S \Rightarrow [q_0, \bot, s''] \Rightarrow A_{q'',\epsilon,Y''}[p''_1, X''_1, p''_2][p''_2, X''_2, p''_3]\cdots[p''_{m''}, X''_{m''}, s''] \Rightarrow^*_R A_{q'',\epsilon,Y''}w''.$$

Thus, $q' = q'' = q_0$ and $Y' = Y'' = \bot$, contradicting our assumption that $A_{q',\epsilon,Y'} \to \cdot$ and $A_{q'',\epsilon,Y''} \to \cdot$ are different.

If $S = \widehat{S}$, then, by Remark 64,

$$\widehat{S} \Rightarrow [r, X, s']_i \Rightarrow A_{q',\epsilon,Y'}[p'_1, X'_1, p'_2][p'_2, X'_2, p'_3]\cdots[p'_{m'}, X'_{m'}, s'] \Rightarrow^*_R A_{q',\epsilon,Y'}w'$$

and

$$\widehat{S} \Rightarrow [r, X, s'']_i \Rightarrow A_{q'',\epsilon,Y''}[p''_1, X''_1, p''_2][p''_2, X''_2, p''_3]\cdots[p''_{m''}, X''_{m''}, s''] \Rightarrow^*_R A_{q'',\epsilon,Y''}w''.$$

Thus, $q' = q'' = r$ and $Y' = Y'' = X$, contradicting our assumption that $A_{q',\epsilon,Y'} \to \cdot$ and $A_{q'',\epsilon,Y''} \to \cdot$ are different.

Subcase (b) In this case, γ is of the form $\gamma'\sigma'$ and $S' \to^*_R \gamma A_{q'',\epsilon,Y''}$, which is impossible, because a variable occurring in an ordinary right-sentential form cannot be preceded by a terminal.

Subcase (c) In this case, for some $\sigma \in \Sigma$, $\sigma' = \sigma'' = \sigma$, because both σ' and σ'' are the last symbol of γ. That is, γ is of the form $\widetilde{\gamma}\sigma$,

$$S' \Rightarrow^*_R \widetilde{\gamma}A_{q',\sigma,Y'}w',$$
$$S' \Rightarrow^* \widetilde{\gamma}A_{q'',\sigma,Y''}w'',$$

and we can proceed exactly like in subcase (a), just substituting σ for ϵ and $\widetilde{\gamma}$ for γ.

Case (*iii*) In this case, $A \to \alpha$ is of the form $A_{r,\epsilon,Z} \to \epsilon$, $\gamma A_{r,\epsilon,Z}$ is an ordinary right-sentential form, and β is a suffix of γ. There are the following four subcases to consider.

(a) $B \to \beta$ is of type 1,
(b) $B \to \beta$ is of type 2,
(c) $B \to \beta$ is of type 3(*b*), and
(d) $B \to \beta$ is of type 4.

Subcase (a) This subcase case is similar to case (*i*) of Proposition 62. Assume that for some $p \in Q$, $B \to \beta$ is $S \to [q_0, \bot, p]$. Since S does not appear on the right side of any production,

$$\gamma = \beta = [q_0, \bot, p].$$

If $A_{r,\epsilon,Z} \to \cdot$ is valid for $[q_0, \bot, p]$, then, for some $w \in \Sigma^*$, there is a derivation

$$S \Rightarrow^*_R [q_0, \bot, p]A_{r,\epsilon,Z}w \Rightarrow_R [q_0, \bot, p]w.$$

However, at the second step of this derivation, $[q_0, \bot, p]$ must be replaced by a word beginning with a subscripted A. That is, it cannot stay intact till the end the derivation.

Subcase (b) This subcase case is similar to case (*ii*) of Proposition 62. Assume that for some proper conjunctive transition $\delta(r, \epsilon, X)$ of degree k, some $p \in Q$, and some $i = 1, 2, \ldots, k$, $B \to \beta$ is $\widehat{S} \to [q, X, p]_i$.

Since $\widehat{S}$ does not appear on the right side of any production,

$$\gamma = \beta = [q, X, p]_i.$$

If $A_{r,\epsilon,Z} \to \cdot$ is valid for $[q, X, p]_i$, then, for some $w \in \Sigma^*$, there is a derivation

$$\widehat{S} \Rightarrow^*_R [q, X, p]_i A_{r,\epsilon,Z}w \Rightarrow_R [q, X, p]_i w.$$

However, at the second step of this derivation, $[q, X, p]_i$ must be replaced by a word beginning with a subscripted A. That is, it cannot stay intact till the end the derivation.

Subcase (c) This subcase case is similar to case (*iii*) of Proposition 62. Assume that for some proper conjunctive transition

$$\delta(q, \epsilon, X) = (q_1, Y_1) \wedge \cdots \wedge (q_k, Y_k)$$

some $p \in Q$, and some $i = 1, 2, \ldots, k$, $B \to \beta$ is $[q, X, p]_i \to A_{q,\epsilon,X}[q_i, Y_i, p]$.

Since $[q, X, p]_i$ does not appear on the right side of any production,

$$\gamma = \beta = A_{q,\epsilon,X}[q, Y_i, p]\,.$$

If $A_{r,\epsilon,Z} \to \cdot$ is valid for $A_{q,\epsilon,X}[q, Y_i, p]$, then, for some $w \in \Sigma^*$, there is a derivation

$$\widehat{S} \Rightarrow_R [q, X, p]_i \Rightarrow_R A_{q,\epsilon,X}[q, Y_i, p] \Rightarrow_R^* A_{q,\epsilon,X}[q, Y_i, p]A_{q,a,Y}w \Rightarrow_R A_{q,\epsilon,X}[q, Y_i, p]aw.$$

However, at the third step of this derivation $[q, Y_i, p]$ must be replaced by a word beginning with a subscripted A. That is, it cannot stay intact till the end the derivation.

Subcase (d) This subcase case is similar to case (iv) of Proposition 62. Assume that for some transition

$$\delta(q, \tau, X) = (p_1, X_1X_2 \cdots X_m)$$

and for some sequence of states $p_2, p_3, \ldots, p_{m+1}$, $B \to \beta$ is

$$[q, X, p_{m+1}] \to A_{q,a,Y}[p_1, X_1, p_2][p_2, X_2, p_3] \cdots [p_m, X_m, p_{m+1}]\,.$$

Then $\gamma = \gamma'\beta$ for some $\gamma' \in V^*$. However, in any rightmost derivation, after production $B \to \beta$ of type 4 is applied, the last symbol of β is immediately expanded by rules productions of type $3(a)$, 4, or 5, so β could not appear intact in an ordinary S'-right-sentential form followed by $A_{r,\epsilon,Z}$. □

We conclude this section with the note, that, like in the classical case, a language is accepted by a DSAPDA by final state (whereas collapsings are by empty stack, cf. footnote 5) if and only if it can be parsed with only one *lookahead* symbol (needed to determine the end of the input), cf. [5, Section 10.8].

6. Recognition power of DSAPDA

In this section we present an example of a DSAPDA language that is not a finite intersection of context-free languages. The language is

$$L_{\text{inf}} = \{a^{i_1}b\,a^{i_2}b^2 \cdots a^{i_k}b^k\,¢\,ba^{i_1}\,ba^{i_2} \cdots ba^{i_k}\,\$: k \geq 1,\ i_1, \ldots, i_k \geq 1\}\,.$$

Proposition 65. *The language L_{inf} is accepted by a DSAPDA.*

Proposition 66. *The language L_{inf} is not a finite intersection of context-free languages.*

Before proving these propositions, we state their immediate corollary.

Corollary 67. *(Cf. the observation at the end of Section 4.) The language L_{inf} does not belong to the boolean closure of deterministic context-free languages.*

Proof. Since deterministic context-free languages are closed under complementation, the boolean closure of these languages consists of their positive boolean combinations. Each positive boolean combination is an intersection of unions and, since context free languages are closed under union, each boolean combination of deterministic context-free languages is an intersection of context-free languages. Therefore, the corollary follows from Proposition 66. □

Proof of Proposition 65. For the proof we describe two DSAPDA, $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$, each in charge of a specific aspect of the language. DSAPDA $\boldsymbol{A}_1$ verifies that the series of bs before the separator ¢ starts with one b and increases by one b at each step. DSAPDA $\boldsymbol{A}_2$ verifies that the numbers of as before and after ¢ match up appropriately. Then the proposition will follow from the fact that (D)SAPDA languages are closed under intersection.

Actually, $\boldsymbol{A}_1$ is the "conjunction" of DSAPDA $\boldsymbol{A}'_1$ and an ordinary deterministic finite automaton $\boldsymbol{A}''_1$ such that

$$L(\boldsymbol{A}'_1) = \{a^{i_1}b^{\ell}\,a^{i_2}b^{\ell+1} \cdots a^{i_j}b^{\ell+j-1} \cdots a^{i_k}b^{\ell+k-1}\,¢\,w\,\$: \ell \geq 1,\ k \geq 1,\ i_1, \ldots, i_k \geq 1,\ \text{and } w \in \{a, b\}^*\}$$

and

$$L(\boldsymbol{A}''_1) = \{a^{i_1}b\,a^{i_2}w'\,¢\,w''\,\$:\ i_1, i_2 \geq 1,\ \text{and } w', w'' \in \{a, b\}^*\}\,.$$

Every time $\boldsymbol{A}'_1$ reads the first b in a series, it recursively opens two branches with $\perp$ at the bottom. The left branch accumulates the bs to compare to the next series of bs and the right branch perpetuates the recursion. The comparison is done by popping one b from the stack for every b read. If the next series of b is longer by 1, then the last b will be

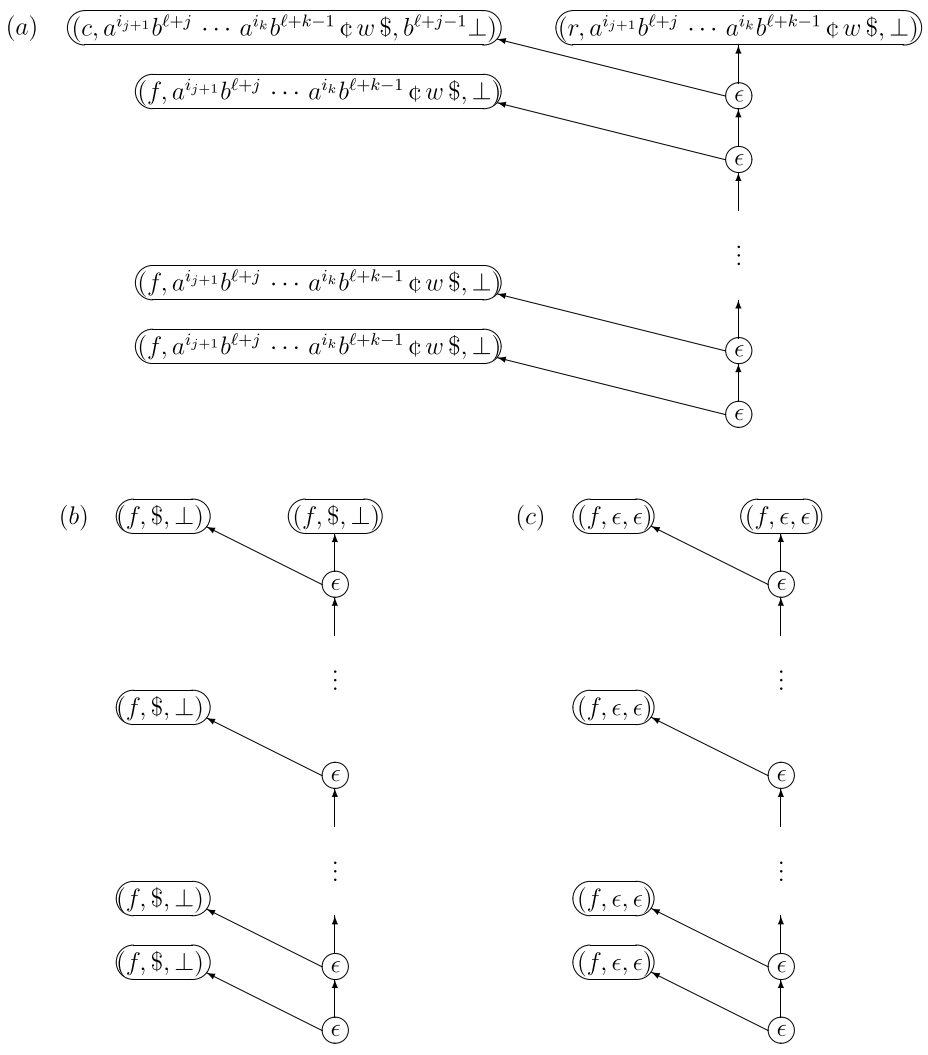

Fig. 5. The configuration of $\boldsymbol{A}'_1$ on input (22) after (*a*) all branches read $a^{i_1}b^{\ell}\,a^{i_2}b^{\ell+1} \cdots a^{i_j}b^{\ell+j-1}$, (*b*) all branches read $a^{i_1}b^{\ell}\,a^{i_2}b^{\ell+1} \cdots a^{i_j}b^{\ell+j-1} \cdots a^{i_k}b^{\ell+k-1}\,¢\,w$, and (*c*) all branches read the whole input.

read when $\bot$ is exposed, and the next character will be either a or ¢. After that the branch skips the rest of the input and empties the stack when the end marker \$ is reached. Thus, all branches collapse at the end of the input.

In particular, after all branches read the prefix $a^{i_1}b^{\ell}\,a^{i_2}b^{\ell+1} \cdots a^{i_j}b^{\ell+j-1}$ of the input

$$a^{i_1}b^{\ell}\,a^{i_2}b^{\ell+1} \cdots a^{i_j}b^{\ell+j-1} \cdots a^{i_k}b^{\ell+k-1}\,¢\,w\,\$ \tag{22}$$

$j+1$ branches are open. The first $j-1$ branches are labeled

$$(f, a^{i_{j+1}}b^{\ell+j} \cdots a^{i_k}b^{\ell+k-1}\,¢\,w\,\$, \bot),$$

where state f indicates that the series of b processed at the branch is shorter by 1 than the next series; the jth branch is labeled

$$(c, a^{i_{j+1}}b^{\ell+j} \cdots a^{i_k}b^{\ell+k-1}\,¢\,w\,\$, b^{\ell+j-1}\bot),$$

where state c indicates that the branch is waiting for comparison or is in process of comparison; and the $(j+1)$th branch is labeled

$$(r, a^{i_{j+1}}b^{\ell+j} \cdots a^{i_k}b^{\ell+k-1}\,¢\,w\,\$, \bot),$$

where state r indicates that the branch is waiting for the recursion step. See Fig. 5 for configurations of $\boldsymbol{A}'_1$ on input (22).

DSAPDA $\boldsymbol{A}_2$ recursively opens a new branch for every first a in a series a^{i_j} that it sees. These branches subsequently store $a^{i_j}b^{j}$ in their stacks, and wait for the separator ¢. After ¢ is read, branch j "counts" to the jth series of as by popping

one b for each b encountered in the input. Thus, the stack content of branch j, after j bs have been read, will be $a^{i_j}\perp$. If all a^{i_j} series before and after the separator ¢ match, the branch skips the rest of the input and empties the stack when the end marker \$ is reached. Thus, all branches collapse at the end of the input.

In particular, after all branches read the prefix $a^{i_1}b\,a^{i_2}b^2\cdots a^{i_j}b^j\cdots a^{i_k}b^k¢$ of input

$$a^{i_1}b\,a^{i_2}b^2\cdots a^{i_j}b^j\cdots a^{i_k}b^k¢\,ba^{i_1}\,ba^{i_2}\cdots ba^{i_k}\,\$ \tag{23}$$

$k+1$ branches are open. Branch j, $j=1,2,\ldots,k$, is labeled

$$(s, ba^{i_1}\,ba^{i_2}\cdots ba^{i_k}\,\$, b^j a^{i_j})$$

for the "storing" state s and branch $(k+1)$ is labeled

$$(r, ba^{i_1}\,ba^{i_2}\cdots ba^{i_k}\,\$, \perp)\,,$$

for the "recursion" state r; and after all branches read the prefix

$$a^{i_1}b\,a^{i_2}b^2\cdots a^{i_j}b^j\cdots a^{i_k}b^k¢ba^{i_1}ba^{i_2}\cdots ba^{i_j}$$

of (23), the first j branches are labeled

$$(c, ba^{i_{j+1}}\cdots ba^{i_k}\$, \perp)\,,$$

branch ℓ, $\ell=j+1, j+2,\ldots,k$, is labeled

$$(c, ba^{i_{j+1}}\cdots ba^{i_k}\$, b^{\ell-j}a^{i_\ell})\,,$$

for a "comparison" state c, and branch $(k+1)$ is labeled

$$(r, ba^{i_{j+1}}\cdots ba^{i_k}\$, \perp)\,.$$

See Fig. 6 for configurations of $\boldsymbol{A}_2$ on input (23). □

For the proof Proposition 66 we shall use the following results from [19].

Lemma 68. *([19, Lemma 9]) For all positive integers k, the class of intersections of k context-free languages is closed under inverse homomorphisms and intersection with regular languages.*

Theorem 69. *([19, Theorem 8]) For all $k\geq 2$, the language*

$$L_k=\{a_1^{i_1}a_2^{i_2}\cdots a_k^{i_k}¢\,a_1^{i_1}a_2^{i_2}\cdots a_k^{i_k}: i_1,\ldots,i_k\geq 1\}$$

is not an intersection of $k-1$ context-free languages.

We shall also need the lemma below.

Lemma 70. *The language*

$$L'_{\text{inf}}=\{a^{i_1}b\,a^{i_2}b^2\cdots a^{i_k}b^k¢\,ba^{i_1}\,ba^{i_2}\cdots ba^{i_k}: k\geq 1,\ i_1,\ldots,i_k\geq 1\}$$

is not a finite intersection of context-free languages.

Proof. Assume to the contrary that for some positive integer k, L'_{inf} is an intersection of $k-1$ context-free languages.

Let

$$L'_k=L'_{\text{inf}}\cap(a^+b^+)^k¢(ba^+)^k\,.$$

Then

$$L'_k=\{a^{i_1}b\cdots a^{i_k}b^k¢\,ba^{i_1}\cdots ba^{i_k}: i_1,\ldots,i_k\geq 1\}\,.$$

By our assumption and Lemma 68, the language L'_k is also an intersection of $k-1$ context-free languages, because the language $(a^+b^+)^k¢(ba^+)^k$ is regular.

Consider the homomorphism

$$h:\{a_1,\ldots,a_k,b_1,\ldots,b_k,¢\}\to\{a,b,¢\}$$

defined by $h(a_i)=a$, $h(b_i)=b^i$, $i=1,\ldots,k$, and $h(¢)=¢$.

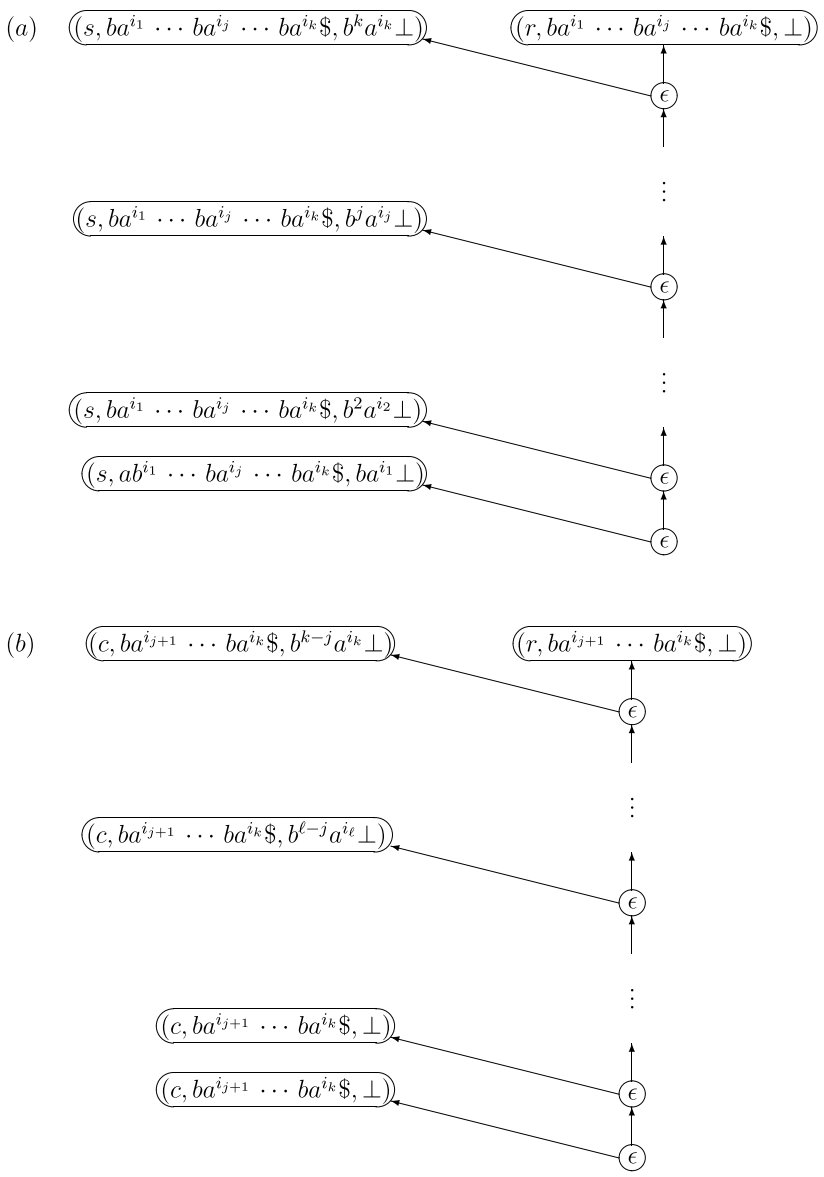

Fig. 6. The configuration of $\boldsymbol{A}'_1$ on input (23) after (*a*) all branches read $a^{i_1} b a^{i_2} b^2 \cdots a^{i_j} b^j \cdots a^{i_k} b^k \mathbin{¢}$ and (*b*) all branches read $a^{i_1} b a^{i_2} b^2 \cdots a^{i_j} b^j \cdots a^{i_k} b^k \mathbin{¢} b a^{i_1} b a^{i_2} \cdots b a^{i_j}$.

Let

$$L''_k = h^{-1}(L'_k) \cap a_1^+ b_1 \cdots a_k^+ b_k \mathbin{¢} b_1 a_1^+ \cdots b_1 a_k^+ .$$

Then

$$L''_k = \{a_1^{i_1} b_1 \cdots a_k^{i_k} b_k \mathbin{¢} b_1 a_1^{i_1} \cdots b_1 a_k^{i_k} : i_1, \ldots, i_k \geq 1\} .$$

Again, by our assumption and Lemma 68, L''_k is an intersection of $k-1$ context-free languages $L_1, \ldots, L_{k-1}$. Intersecting these languages with the regular language

$$a_1^+ b_1 \cdots a_k^+ b_k ¢ b_1 a_1^+ \cdots b_1 a_k^+ ,$$

if necessary, we may assume that each of them only contains words of the form

$$a_1^+ b_1 \cdots a_k^+ b_k ¢ b_1 a_1^+ \cdots b_1 a_k^+ .$$

Consider the homomorphism

$$g : \{a_1, \ldots, a_k, b_1, \ldots, b_k, ¢\} \to \{a_1, \ldots, a_k, ¢\}$$

defined by $g(a_i) = a_i$, $g(b_i) = \epsilon$, $i = 1, \ldots, k$, and $g(¢) = ¢$.

Then

$$L_k = g\left(L_k''\right) = g\left(\bigcap_{i=1}^{k-1} L_i\right) = \bigcap_{i=1}^{k-1} g(L_i)\,,$$

where the last equality holds because the words in the intersection must align on the $a_1, \ldots, a_k$ symbols, regardless of the presence or absence of the $b_1, \ldots, b_k, ¢$ symbols.

However, $L_k = \bigcap_{i=1}^{k-1} g(L_i)$ contradicts Theorem 69. □

Proof of Proposition 66. Assume to the contrary that for some positive integer k,

$$L_{\text{inf}} = \bigcap_{i=1}^{k} L_i\,,$$

where $L_1, L_2, \ldots, L_k$ are context-free languages. Intersecting these languages with the regular language

$$L = (a^+b^+)^*¢(a^+b^+)^*\$\,,$$

if necessary, we may assume that each of them only contains words from L.

Consider the homomorphism

$$f : \{a, b, ¢, \$\} \to \{a, b, ¢\}$$

defined by $f(a) = a$, $g(b) = b$, $f(¢) = ¢$, and $f(\$) = \epsilon$.

Obviously,

$$f\left(\bigcap_{i=1}^{k} L_i\right) = \bigcap_{i=1}^{k} f(L_i)\,.$$

Therefore,

$$L'_{\text{inf}} = f\left(L_{\text{inf}}\right) = f\left(\bigcap_{i=1}^{k} L_i\right) = \bigcap_{i=1}^{k-1} f(L_i)$$

in contradiction to Lemma 70. □

7. Concluding remarks

We have introduced $LR(0)$ conjunctive grammars as a subfamily of conjunctive grammars and DSAPDA as a subfamily of SAPDA and shown that, like in the classical case, generation by an $LR(0)$ conjunctive grammar is equivalent to acceptance by a DSAPDA. This equivalence also forms the basis for a linear time parsing algorithm for the class of languages comprised of the boolean closure of conjunctive $LR(0)$ languages.

One might think that, similarly to [1, Theorem 5.9, p. 383], the $LR(0)$ condition extends to k lookahead symbols. This, however, will not work, because the parser must know the end of the part of the input along a computation branch, see Remark 14. Thus, the question "what is the $LR(k)$ condition?" remains open.

It would be interesting to explore uses for DSAPDA based compilers. Two directions seem especially promising. The first is to look for examples where $LR(0)$ conjunctive grammars give a more succinct representation of an ordinary $LR(0)$ grammar, therefore leading to more efficient parsing. The second is to find examples of deterministic conjunctive languages which can be used to describe sophisticated constructs beyond the scope of context free languages. Such examples could be useful for areas where context free languages have been known to be lacking, such as natural language parsing.

References

[1] A.V. Aho, J.D. Ullman, The Theory of Parsing, Translation, and Compiling. Vol. I: Parsing, Prentice Hall, Englewood Cliffs, NJ, 1972.
[2] T. Aizikowitz, M. Kaminski, Conjunctive grammars and alternating pushdown automata, in: W. Hodges, R. de Queiroz (Eds.), The 15th Workshop on Logic, Language, Information and Computation – WoLLIC'2008, in: Lect. Notes Artif. Intell., vol. 5110, Springer, Berlin, Heidelberg, 2008, pp. 30–41.
[3] T. Aizikowitz, M. Kaminski, Conjunctive grammars and alternating pushdown automata, Acta Inform. 50 (2013) 175–197.
[4] A.K. Chandra, D.C. Kozen, L.J. Stockmeyer, Alternation, J. ACM 28 (1981) 114–133.
[5] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages and Computation, Addison–Wesley Publishing Co., Reading, MA, 1979.
[6] P.M. Lewis II, R.E. Stearns, Syntax-directed transduction, J. ACM 15 (1968) 465–488.

[7] D.E. Knuth, On the translation of languages from left to right, Inf. Control 8 (1965) 607–639.
[8] O. Kupferman, M.Y. Vardi, Weak alternating automata are not that weak, ACM Trans. Comput. Log. 2 (2001) 408–429.
[9] R.E. Ladner, R.J. Lipton, L.J. Stockmeyer, Alternating pushdown and stack automata, SIAM J. Comput. 13 (1984) 135–155.
[10] A. Okhotin, Conjunctive grammars, J. Autom. Lang. Comb. 6 (2001) 519–535.
[11] A. Okhotin, LR parsing for conjunctive grammars, Grammars 5 (2002) 81–124.
[12] A. Okhotin, Top-down parsing of conjunctive languages, Grammars 5 (2002) 21–40.
[13] A. Okhotin, A recognition and parsing algorithm for arbitrary conjunctive grammars, Theor. Comput. Sci. 302 (2003) 81–124.
[14] A. Okhotin, Boolean grammars, Inf. Comput. 194 (2004) 19–48.
[15] A. Okhotin, Fast parsing for boolean grammars: a generalization of valiant's algorithm, in: Y. Gao, H. Lu, Sh. Seki, Sh. Yu (Eds.), Developments in Language Theory, 14th International Conference, DLT 2010, in: Lect. Notes Comput. Sci., vol. 6224, Springer, Berlin, Heidelberg, 2010, pp. 340–351.
[16] A. Okhotin, Conjunctive and boolean grammars: the true general case of the context-free grammars, Comput. Sci. Rev. 9 (2013) 27–59.
[17] M. Tomita, Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems, Kluwer Academic Publishers, Norwell, MA, 1985.
[18] L.G. Valiant, General context-free recognition in less than cubic time, J. Comput. Syst. Sci. 10 (1975) 308–315.
[19] D. Wotschke, Nondeterminism and boolean operations in PDA's, J. Comput. Syst. Sci. 16 (1978) 456–461.
[20] D.H. Younger, Recognition and parsing of context-free languages in time n^3, Inf. Control 10 (1967) 189–208.