Abstract

Coupled-Context-Free Grammars are a generalization of context-free grammars obtained by combining nonterminals to parentheses which can only be substituted simultaneously. Referring to the generative capacity of the grammars, we obtain an infinite hierarchy of languages that comprises the context-free languages as the first and all the languages generated by Tree Adjoining Grammars (TAGs) as the second element. Here, we present a generalization of the context-free LR(k)-notation, which characterizes subclasses of Coupled-Context-Free Grammars and therefore for TAGs — which can be parsed in linear time. The parsing procedure described works incrementally so that it can be used for on-line parsing of natural language. Examples show that important Tree Adjoining Languages, e.g., those modeling cross-serial dependencies, can be generated by LR(k)-Coupled-Context-Free Grammars.

I Introduction

In order to process natural languages, we first have to model the syntax formally. Many investigations as, e.g., [Hi84] show that this cannot be done by context-free grammars (CFGs). For context-sensitive grammars which are powerful enough, it is known that the analysis is PSPACE-complete. Thus, there is a trade-off between the power of the formalism and its analysis complexity. To solve this dilemma, much work has been done to characterize language classes in between context-free and context-sensitive languages being powerful enough to model the syntax of natural languages but endowed with a polynomial time analysis. Coupled-Context-Free Grammars represent such a formalism generalizing CFGs. Their suitability to model syntactical phenomena follows from the fact that they include the languages generated by the Tree Adjoining Grammars (TAGs) of [Jos87] as one subclass. Among other properties, both formalisms are able to model the linguistic phenomenon of cross-serial dependencies, which is not context-free but frequently appears in natural languages (cf. [Sh86]).

The formalism of Coupled-Context-Free Grammars has been introduced in [GH92] and [Giu92]. It belongs to the family of regulated string rewriting systems investigated in [DP89]. The increased generative capacity is obtained by allowing to rewrite simultaneously a certain number of elements. Other regulated string rewriting systems as, e.g., the Scattered Context Grammars of [GH89] generalize CFGs by allowing simultaneous rewriting of arbitrary combinations of elements. In [DP89], it is shown that this results in languages which are not semilinear. But semilinearity is important since it formalizes the "constant-growth property" of natural languages (cf. [Jos81]). In contrast to these, all languages defined by our formalism are semilinear because of two restrictions. First, only those elements can be rewritten simultaneously which were produced by the same rewriting. Second, the Coupled-Context-Free Grammars consider elements rewritten simultaneously as components of a parenthesis. Those can only be substituted if they form a parenthesis and they can only be substituted by sequences of parentheses correctly nested.

When characterizing Coupled-Context-Free Grammars by the maximal number of elements rewritten simultaneously — which we call the rank of the grammar — we get an infinite hierarchy. The generative capacity grows with the rank. The smallest element of the hierarchy — the one of rank 1 — are CFGs. The next element, namely Coupled-Context-Free Grammars of rank 2, generates the same class of languages as the Tree Adjoining Grammars of [Jos87] and [Jos85]. Hence, all notions and algorithms designed for Coupled-Context-Free Grammars of rank 2 can easily be translated onto TAGs (cf. [Giu92]).

Because of the enlarged generative capacity, it is not surprising that the complexity of analysing languages generated by Coupled-Context-Free Grammars is larger than it is in the context-free case. It even increases with growing rank (cf. [HP90]). Therefore, we aim to characterize subclasses of all languages generated by Coupled-Context-Free Grammars which are powerful enough to model the important phenomena of natural languages, but which are of a lower complexity.

The deterministic context-free parsing with LR(k)-grammars leads to a linear time analysis (cf. [Knu65]), the best possible. Therefore, its generalization is very attractive. A first attempt in this direction was done in [SV90]. But there, only TAGs are investigated. Here, we investigate the whole hierarchy of Coupled-Context-Free Grammars. Although their enlarged generative capacity seems to be contradictory to a linear time complexity of the parsing algorithm, we can present an LR(k)-notation for Coupled-Context-Free Grammars describing a class of languages, which can actually be analysed in linear time. This increase in power as to the linear-time analysis is paid by an expensive preprocessing. It is taking into account the complex relations between parentheses that involves the increase in complexity. However, these costs are to be paid only once for each grammar. The subclass described by our LR(k)-notation for a fixed k grows with the rank.

The algorithm of [SV90] for LR(k)-TAGs does not fulfill the important Valid Prefix Property. This means that for any prefix of the input already accepted, there exists a suffix such that the whole word is in the language analysed. It allows to detect illegal inputs as soon as possible, which is necessary for efficient parsing. Our algorithm fulfills this property. Additionally, the algorithm as well as the notion defined here represent generalizations of their context-free counterparts which are natural in the sense that they strictly contain the context-free situation as the special case of Coupled-Context-Free Grammars of rank 1.

*This research has been supported by a Graduiertenkolleg-fellowship of the Deutsche Forschungsgemeinschaft.
An example of an important \( LR(k) \)-Coupled-Context-Free Grammar is the one generating the language \( \{w^k \mid \omega \in \{a, b\}^*\} \) which reflects the syntactical construction of cross-serial dependencies.

The paper starts by defining the Coupled-Context-Free Grammars. Then, we shortly recall the context-free \( LR \)-parsing procedure. Subsequently, the deterministic finite automaton used there to guide the analysis is modified such that it can handle Coupled-Context-Free Grammars. Based on it, the parsing algorithm for \( LR(0) \)-Coupled-Context-Free Grammars is derived. This results in the generalized definition of the \( LR(0) \)-action. As for CFGs, the \( LR(0) \)-Coupled-Context-Free Grammars result from the \( LR(0) \)-ones by resolving decision conflicts using a lookahead of at most \( k \) symbols.

2 Coupled-Context-Free Grammars

Coupled-Context-Free Grammars are defined over extended semi-Dyck sets which are a generalization of semi-Dyck sets. Elements of these sets can be regarded as sequences of parentheses that are correctly nested. Semi-Dyck sets play an important role in the theory of formal languages. To extend the family of context-free languages by using them we consider parentheses of arbitrary finite order defined as follows:

Definition 2 (Extended Semi-Dyck Set)

Let \( \mathcal{K} \) be a set of parentheses and \( \mathcal{T} \) an arbitrary set. The following theorem proves in [Gua92] shows that \( CCFG(l) \) builds up an infinite hierarchy of languages and, at the same time, represents a proper extension of CFGs not exceeding the power of context-sensitive grammars:

Theorem 1 (Hierarchy)

Let \( CFL \) be the family of all context-free, \( CSL \) the family of all context-sensitive languages, \( TAL \) the family of all languages generated by TAGs and \( CCFG(l) \) the one generated by \( CCFG(l) \). It holds:

1. \( CFL=CCFL(1)\), \( TAL=CCFL(2)\).
2. \( CCFL(l) \subseteq CCFL(l+1) \) for all \( l \geq 1 \).
3. \( CCFL(l) \subseteq CSL \) for all \( l \geq 1 \).

Sometimes, \( 1 \) is useful to "neglect" the relations between the components of a derivation for a short time. Then, we investigate \( G' := (\text{comp}(K), T, P', S) \) instead of \( G = (K, T, P, S) \) in \( CCFG(l) \) for

\[ P' := \bigcup \left\{ \left\langle \ell_1, \ldots, \ell_t \right\rangle \mid \ell_1, \ldots, \ell_t \subseteq P \right\} \]

Since \( G' \) is certainly a CFG we denote \( G' \) (resp. \( P' \)) by \( G(F(G)) \) (resp. \( G(F(P)) \)) in the sequel. Obviously, \( G \) satisfies \( L(G) \subseteq L(G') \).
3 Context-Free Lr-Parsing

Now, we shortly recall the deterministic context-free Lr(k)-parsing strategy of Knuth (cf. [Knu65]). For simplicity, we restrict ourselves on the case k = 0. The strategy essentially remains unchanged if lookahead is necessary. It uses a deterministic finite automaton (dfa) to drive a pushdown stack while scanning the input from left to right. Thus, it constructs a rightmost derivation bottom-up. The states of the dfa for a given LR(0)-CFG consist of subsets of the set of all context-free items for \( G = (N, T, P, S) \), i.e. of the set \( \{ [X \rightarrow \alpha \beta] : X \rightarrow \alpha \beta \in P \} \). They result from determining the deterministic version of the following nondeterministic automaton for \( G \):

- Each context-free item is a state.
- There are three kinds of state transitions:
  - \( [X \rightarrow \alpha \gamma X \beta] \rightarrow [X \rightarrow \alpha \gamma \beta] \) for any \( \gamma \notin \alpha \beta \).
  - \( [X \rightarrow \alpha \beta] \rightarrow [X \rightarrow \alpha X \beta] \).
  - \( [\gamma \rightarrow \gamma.X] \rightarrow [X \rightarrow \gamma] \).

In the deterministic version, all those context-free items are grouped in one state which can be reached from the initial state by the same sequence of symbols, with any possible number of \( \varepsilon \)-transitions in-between.

The stack symbols are the states of the dfa. At first, the state containing the item \( \{ S \rightarrow \} \) is pushed. (The additional production \( S' \rightarrow S \) serves to define exactly the start and the end of the analysis.) Then, we iterate the following actions depending on the topmost state \( q \):

(Shift) If \( q \) contains \( [X \rightarrow \alpha \beta] \) and \( \alpha \) is the next input symbol to be read, we push the state reached from \( q \) via \( \alpha \). (It contains at least \( [X \rightarrow \alpha \beta] \).)

(Reduce) If \( q \) contains \( [X \rightarrow \alpha] \), we pop the \( \alpha \) next topmost states. Let \( q' \) be the state now on top of the stack. Then, we push the state reached via \( X \) from \( q' \). \( q' \) contains at least one item \( [Y \rightarrow \gamma.X] \) and \( [X \rightarrow \alpha] \) while the new topmost state contains \( [Y \rightarrow \gamma.X.] \).

The pushdown is driven deterministically by the dfa if this dfa contains no state where there are two different Reduce-items (Reduce-Reduce conflict) or as well a Shift- as a Reduce-Shift conflict. A CFG is LR(0) iff the states of its dfa show no Shift-Reduce and no Reduce-Reduce conflict. For LR(k)-grammars, conflicts in the LR(0)-dfa are looked by a lookahead of \( k \) symbols.

4 The Finite Automaton

One possibility to generalize dfa is to construct the usual dfa for \( C(F(G), G \in CFGF \). In principle, this idea is used in [SV90]. The following example shows that this produces unnecessary conflicts: Let \( G = (\{ S(X, X), D \}, \{ a, b, c, d \}, P, S) \in CFGF \) with \( P = \{ S \rightarrow XXD, D \rightarrow Dd \ | d, \{ X, X \rightarrow (b, c) \ | (a, b, c, d) \) and \( L(G) = \{ \text{hol} a \text{ hol} b \text{ hol} c \text{ hol} d, n \geq 1 \}\). Its dfa is shown in Figure 1. \( G \) is not LR(0) in this way since this dfa obviously has a Shift-Reduce conflict (in the box doubly lined). This conflict cannot be solved by lookahead since at this point, the lookahead is always \( \varepsilon \). Therefore, \( G \) is not LR(k) for any \( k \geq 0 \). But this conflict is not necessary. E.g., when analysing \( hol \) bottom-up, we first have to reduce \( X \rightarrow b \). This implies that before coming to the conflict state, we have to choose \( X \rightarrow a \) in order to get a correct derivation. This is the case because \( X \) and \( X \) resulting from applying the production \( S \rightarrow XXD \) are coupled and therefore have to be substituted by coupled productions.

To avoid these conflicts, we extend the dfa. If we use the context-free LR-parsing strategy, we know which production we have to choose for any \( X \rightarrow \alpha \beta \notin \alpha \beta \) because we first encounter and reduce the corresponding \( X \rightarrow \alpha \beta \). Suppose that we can store the information about \( X \rightarrow \alpha \beta \), \( X \rightarrow \alpha \beta \in \alpha \beta \), when \( X \) is reduced, let us say as the "future". (How to do this is shown in Section 5.) Can we use this to avoid the conflict? Now, our automaton needs additional transitions under such \( p \in p^F \) where \( S(p) \notin \beta \) holds. Thus, we split ways inside the dfa which lead to conflict states. To formalize our automaton, we need the following Definition 6 (1-Closure)

For all \( X \in \text{comp}(K) \), let reachable \((X) := \{X \rightarrow \beta \mid X \rightarrow \beta \in \text{comp}(K) \} \) denote its reflexive transitive closure. For any \( \beta \in \varphi(\{[X \rightarrow \alpha \beta] \mid X \rightarrow \alpha \beta \in \text{comp}(K) \}) \), we define the 1-Closure \((\beta) := \{ q \mid q \in \text{reachable}((X) \} \).

1-Closure formalizes the construction of the deterministic version of a nondeterministic finite automaton as it is done for the dfa of CFGs. Its special feature is that it uses only those \( X \rightarrow \alpha \beta \in \text{comp}(K) \) fulfilling \( X \rightarrow \beta \in \text{comp}(K) \). If \( X \rightarrow \beta \notin \text{comp}(K) \), the extending production is determined by the corresponding first component.

Definition 7 (DFA(G))

Let \( G = (\{ N, T, P, S \} \in CFGF \). The Deterministic Finite Automaton for \( G \) is defined as \( DFA(G) := \{ (Q_G, \Sigma_G, b_G, S_G, F_G) \mid \) where \( S_G := 1-\text{Closure}(\{ [S \rightarrow S] \}) \) is the initial state, \( \Sigma_G := \text{comp}(K) \cup \Sigma \cup \{ p \in CF(P) \mid \Sigma(p) \notin \text{comp}(K) \} \) is the input alphabet, \( b_G \) the transition function defined for \( \xi \in \text{comp}(K) \cup \Sigma \cup \{ p \in CF(P) \mid \Sigma(p) \notin \text{comp}(K) \} \), and \( F_G := \{ q \mid \exists \alpha \in \text{comp}(K) \cup \Sigma \cup CF(P)^* : b_G(q, \alpha) = q \} \).

403
The first difference to the usual context-free automaton is that we allow transitions under \( f_i \in CF(P) \), if we have \( S(f_i) \notin compl(K) \). The second point is that we use 1-Closure instead of the usual closure. \( DFA(G) \) for the example grammar is shown in Figure 2. The conflict is removed because we can now distinguish two cases by looking at the information additionally stored. In [SV90], only the first idea was realized obviously leading to a weaker automaton.

5 The Analysis

To use \( DFA(G) \), the usual pushdown is extended by a data-structure consisting in a list of partial derivation trees. This list future collects all information determined by Reduce's relative to first nonterminal components and is used to drive the transitions under \( p \in CF(P) \) in \( DFA(G) \) as soon as we have to investigate nonterminal components \( X_i \notin compl(K) \). The change between the two different kinds of control leads to a new characterization of conflicts.

For better explanation, we use a list past parallel to future where all Reduce operations performed so far are stored. An example for the new data-structures is shown in Figure 3. We use it to explain how they are built up during the analysis. The first operations on this past were Shift\((w_1)\), Shift\((w_2)\), Reduce\((A \rightarrow w_3)\). From CFGs, we know that any Reduce takes place at the end of the sentential form generated so far. This remains true. Thus, we can argue completely analogous as far as past is concerned.

But we investigate coupled productions as, e.g., \((Z_1, Z_2) \rightarrow (w_1 A, U_1 U_2)\). We know that coupled nonterminal components are located at the same depth of the derivation tree and that they are substituted by components of the same coupled productions. Therefore, when inserting any \( p \), \( S(p) \in compl(K) \), in past, e.g., \( Z_1 \rightarrow w_4 A \), we additionally insert the coupled productions, e.g., \( Z_2 \rightarrow U_1 U_2 \), in future. In general, there are two cases to distinguish depending on \( p_j \) inserted in future. If \( D(p_j) \) contains only symbols in \( K \cup \gamma \) (i.e. only uncoupled ones), the coupled \( p_{j_1}, \ldots, p_{j_l} \) are inserted as the first up to the \((r-1)\)th element in future. (E.g. for \((Z_1, Z_2) \rightarrow (w_1 A, U_1 U_2)\). Otherwise, we behave as it is done for \((Y_1 Y_2) \rightarrow (Z_1 Z_2 Q_1 Q_2)\) in Figure 3. I.e. the subtrees in future for those symbols in \( D(p_j), \ldots, D(p_u) \) coupled to first components in \( D(p_u) \) become the sons of these elements. Thus, we maintain the property that the symbols at each fixed depth in past and future together form an element of ED(K, \( \gamma \)).

Thereby, in addition to Shift's which are handled as usual, we know what to do for Reduce operations relative to elements of \( compl(K) \). Now, let us be in the situation that we have to use the information in future, e.g. a transition under \( B_5 \rightarrow a =: p_i \) from the topmost state. Then, we create a pointer presence walking on future. We push \( h_{c}(q_{top}, p_i) \) and make presence point onto the first son \( \xi \) of \( D(p_i) \). Let \( q \) be the new topmost state. We have to distinguish three cases:

\[ \xi \in \gamma ? \]  
If \( \xi \) is the next input symbol, we push \( h_{c}(q, \xi) \).

Otherwise, the whole input is rejected. presence now points on the brother of \( \xi \).

\[ \xi \in compl(K) \setminus compl(K) : \text{future already stores the expansion } \xi \rightarrow \beta. \text{ We push } h_{c}(q, \xi \rightarrow \beta). \] presence now points on the first symbol in \( \beta \).

\[ \xi \in compl(K) : \text{future does not store information about } \xi, \text{ but } \xi \text{ and its coupled components represent a complete independent analysis problem which has to be solved recursively. E.g., this is the case for } D, \quad (U_1, U_2), \quad \text{and } (Q_1, Q_2). \] The recursive call of the procedure starts with the topmost state since it contains all items \( \xi \rightarrow \alpha \). Each of these recursions needs separate data-structures. Details are described in [Pit93].

If presence encounters no brother, we have to reduce. Let \( Y \rightarrow \alpha \gamma \) be the production at whose last symbol presence points. We pop \( [ \gamma ] \) from states. The additional pop compared to the context-free case results from the transition under \( Y \rightarrow \gamma \). presence walks to the brother of \( Y \) in future and we push \( h_{c}(q', Y) \) if \( q' \) is the new topmost state. If \( Y \) is the root of the first tree in future, its complete subtree is moved from future to past and presence is deleted.

We output \( p \in \gamma \) when reducing its last component. Thus, our result is a rightmost derivation in inverse order.

6 The Definition

So far, we did not discuss the situation that there are distinct transitions fitting for the same state in \( DFA(G) \). Shift-Reduce and Reduce-Reduce conflicts are forbidden as they are for CFGs. The new conflicts result if we have to decide whether we push \( h_{c}(q_{top}, f_j) \), \( f_j \in CF(P) \), or Shift resp. Reduce as usual. If a state \( q \) shows such a "new" conflict, it contains two items of the kind \((Z, \gamma, Y, \eta)\) and \([X_1 \rightarrow \alpha_1 \gamma \beta_1, \xi \in \gamma \), or \([X_1 \rightarrow \alpha_1 \gamma \beta_1, \xi \in \gamma \). This is easy to decide as far as we are walking on future, since the information necessary is stored there. Thus, we only have a real conflict if \( i = 1 \) and \( I = 1 \) holds for the above items. Obviously, this cannot be decided deterministically, since we would have to know about the structure of the derivation tree not constructed so far. E.g., in the first conflict, we would have to say whether \( \xi \) is a son of \( Y_1 \) (choose \( h_{c}(q, Y_1 \rightarrow \mathcal{D}(f_j)) \)) or whether \( \xi \) is a son of \( X_1 \) (choose \( h_{c}(q, \xi) \)). It follows that we need a modified definition of "conflicts" compared to CFGs.
Definition 8 (Conflict)
For any \( G = (\{X, Y, \beta\}, \{\alpha\}, \text{comp}(\beta), \beta, \alpha) \in \text{CCFG} \), \( \text{DFA}(G) \) shows a conflict if at least one of its states contains a subset of the following kind:

\[
\begin{align*}
(\text{R-R}) & \quad \{X \rightarrow \alpha \}, \{Y \rightarrow \beta\} \mid X, Y \in \text{comp}(\beta), \\
(\text{S-R}) & \quad \{X \rightarrow \alpha, Y \rightarrow \beta\} \mid X, Y \rightarrow \alpha, Y \rightarrow \beta \in \text{CF}(\beta), \\
(\text{S-E}) & \quad \{X \rightarrow \alpha, Y \rightarrow \beta\} \mid X, Y \rightarrow \gamma \in \text{comp}(\gamma), \\
(\text{R-E}) & \quad \{X \rightarrow \alpha\} \mid X \rightarrow \alpha \in \text{comp}(\alpha), \gamma \rightarrow \beta \in \text{CF}(\beta), \\
\end{align*}
\]

Definition 9 (LR(0) in CCFG)
\( G \in \text{CCFG} \) is LR(0) \( \Rightarrow \) \( \text{DFA}(G) \) has no conflicts.

Theorem 2
Let \( G \in \text{CCFG} \) be LR(0). Our algorithm deterministically solves the word problem for any \( w \in T^* \), \( n := |w| \), in time \( O(n) \) by constructing a leftmost derivation relative to \( G \) if \( w \in L(G) \), and, if \( w \notin L(G) \), by rejecting the input. In addition, the algorithm shows the Valid Prefix Property.

Proof: The linear time complexity follows since we only need a constant amount of additional states per context-free step for past and future. \( \text{DFA}(G) \) is determined only once for each \( G \). The \( \text{VPP} \) holds since it holds for the context-free algorithm and future additionally ensures that the coupling is correct. □

LR(1)-Coupled-Context-Free Grammars result from the above by resolving conflicts in \( \text{DFA}(G) \) by adding a lookahead set to the items which are involved in a conflict. For this purpose, we use the mappings \( \text{FIRST}_A \) and \( \text{FOLLOW}_A \) as defined for \( LL(k) \)-Coupled-Context-Free Grammars in [Pit94]. There, these mappings are generalized such that they take the coupling between the components of each nonterminal into account instead of working simply on \( \text{CF}(G) \). Thus, we treat only complete parentheses as a context-free nonterminal and the result is much more exact as, e.g., in [SV90]. This results in an adequate generalization of the \( LR(k) \)-notion for CCFG.

Example 2
The language \( \{a^n b^n c^n d^n \mid n \geq 1\} \) generated by the grammar in Example 1 shows the \( LR(0) \)-property. \( \text{DFA}(G) \) is shown in Figure 4.

Example 3
The language \( \{a^n b^n \mid w \in \{a, b\}^*\} \) modelling cross-serial dependencies can be generated by the \( LR(1) \)-grammar \( (\{S, \{X, \bar{X}\}\}, \{\alpha, \beta\}, P, S) \in \text{CCFG} \) where \( P := \{S \rightarrow aX \bar{X}, (X, \bar{X}) \rightarrow (aX, aX) \mid (bX, bX) \mid (c, e)\} \).

References
[BP88] J. Dassow, G. Páñ: Regulated Rewriting in Formal Language Theory. Springer 1989
[GI69] S.A. Greibach, J.E. Hopcroft: Scattered Context Grammars. J. Comput. Syst. Sci. 3 (1969), pp. 232-247
[GRH92] Y. Guan, G. Hatz, A. Reichert: Tree Grammars with Multilinear Interpretation. Technical Report, Univ. Saarbrücken 1992
[Gu92] Y. Guan: Klammern grammaticken, Netzgrammatiken und Interpretationen von Netzen. PhD thesis, Univ. Saarbrücken 1992
[HP94] G. Hatz, G. Pitsch: Fast Uniform Analysis of Coupled-Context-Free Grammars. In Proc. of ICAI’94
[Hig84] J. Higginsbotham: English is not a Context-Free Language. Ling. Inquiry 15 (1984), pp. 225-235
[JLS75] A.K. Joshi, L.S. Levy, M. Takahashi: Tree Adjunct Grammars. J. Comput. Syst. Sci. 10 (1975), pp. 136-163
[Jos85] A.K. Joshi. Tree Adjoining Grammars: How Much Context-Sensitivity is Required to Provide Reasonable Structural Descriptions. In Natural Language Parsing, Cambridge University Press 1985, pp. 206-253
[Jos87] A.K. Joshi: Introduction to Tree Adjoining Grammars. In Mathematics of Language, John Benjamin Publishing Company 1987
[Knu65] D.E. Knuth. On the Translation of Languages from Left to Right. Information and Control 8 (1965), pp. 607-639
[Pit93] G. Pitsch: Analyse von Klammergrammatiken. PhD thesis, Univ. Saarbrücken 1993
[Pit94] G. Pitsch: \( LL(k) \)-Parsing of Coupled-Context-Free Grammars. to appear in Computational Intelligence 1994
[Shi86] S. Shieber. Evidence against Context-Freeness of Natural Language. Linguistics and Philosophy 8 (1986), pp. 333-343
[SV90] Y. Schabes, K. Vijay-Shanker: Deterministic Left to Right Parsing of Tree Adjoining Languages. AIC’90, pp. 276-283

Figure 4: \( L := \{a^n b^n c^n d^n \mid n \geq 1\} \)