Contents

Chapter 10

Computational complexity of parsing

10.1 Parsing with shallow logical dependencies

This section describes two parsing methods for ordinary grammars, which are both based on a single underlying idea of using an augmented deduction system that allows *shallow proof trees*. Using this idea, one can construct a recognitions procedure that uses only $O((\log n)^2)$ bits of memory, at the expense of running time $n^{O(\log n)}$, which is super-polynomial: this is the algorithm by Lewis, Stearns and Hartmanis [8]. Another application of the same idea, due to Brent and Goldschlager [1] and to Rytter [11], is to construct a Boolean circuit, which works in time $O((\log n)^2)$ using $O(n^6)$ processing units.

10.1.1 Height of a parse tree

Height of logical dependencies. If long strings are concatenated, the height may be as low as logarithmic. The worst case is linear concatenation, for which the parse tree is always of linear height. In particular, the following ultimately simple grammar for the language a^* can be regarded as the worst case with respect to the height of parse tree:

$$S \to aS \mid \varepsilon$$

The goal is to replace ordinary parse trees with an equivalent system of logical dependencies, in which the height will be bounded by a logarithmic function.

A proposition $\frac{A}{D}(u : v)$, with $A, D \in N$ and $u, v \in \Sigma^*$, means that there exists a parse tree with a root $A \in N$, with a gap represented by a node labelled D without descendants, and with $|u| + |v|$ labelled labelled u and v, to the right and to the left of D, respectively. Extra deduction rules:

$$\begin{aligned}
X_1(u_1), X_{i-1}(u_{i-1}), X_{i+1}(u_{i+1}), X_\ell(u_\ell) &\vdash \frac{A}{X_i}(u_1 \ldots u_{i-1} : u_{i+1} \ldots u_\ell) && \text{(creating a gap for } A \to X_1 \ldots X_\ell \in R\text{)}\\
\frac{A}{D}(u : v), D(w) &\vdash A(uwv) && \text{(filling the gap)}\\
\frac{A}{E}(u : v), \frac{E}{D}(x : y) &\vdash \frac{A}{D}(ux : yv) && \text{(combining conditional propositions)}
\end{aligned}$$

It is claimed that:

1. whatever can be proved in this extended system, can be proved in the main system using only propositions of the form $A(w)$;

2. every proposition $\frac{A}{D}(u : v)$ or $D(w)$ has a proof of height $O(\log n)$, where $n = |uv|$ or $n = |w|$ is the number of leaves.

The latter statement is established using the following property.

Lemma 10.1. *Let $G = (\Sigma, N, R, S)$ be an ordinary grammar in Chomsky's normal form. Then every parse tree with n leaves contains a middle node that spans over at least $\frac{1}{3}n$ and at most $\frac{2}{3}n$ leaves.*

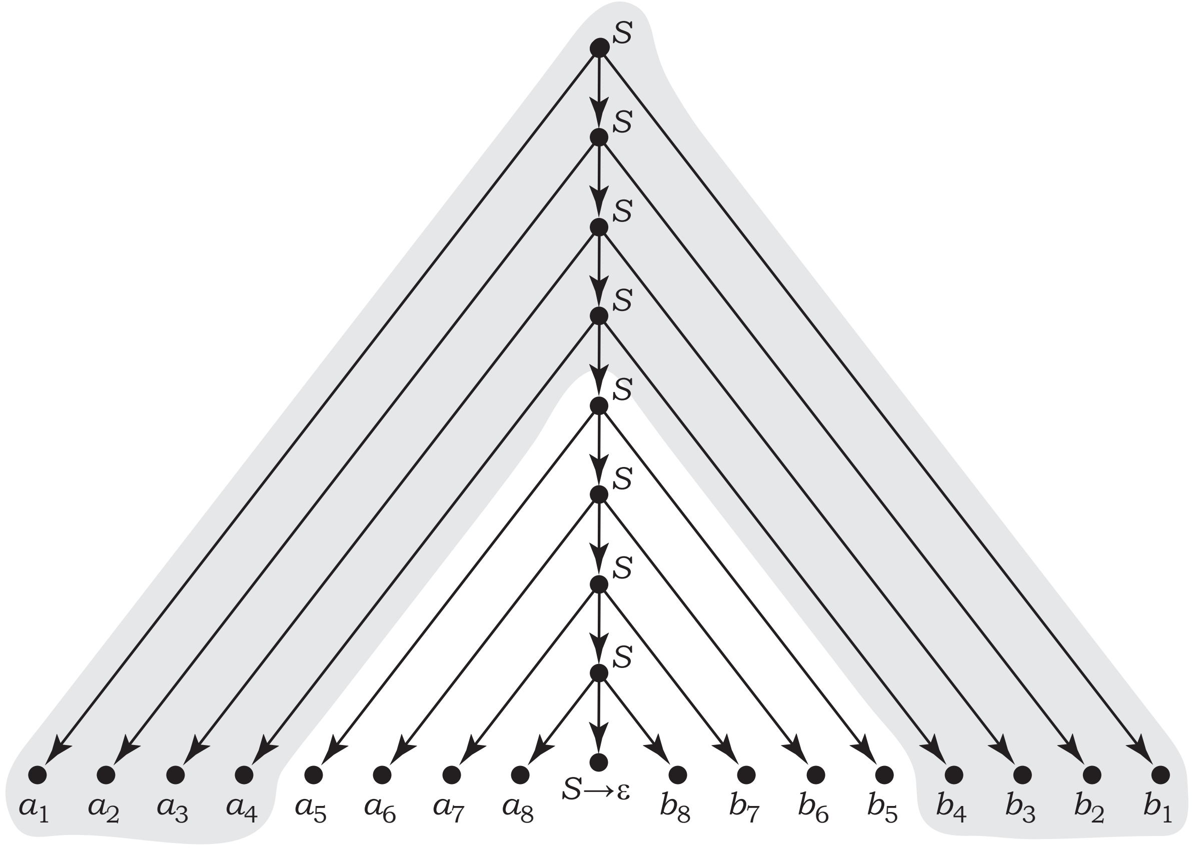

Figure 10.1: A parse tree for the string a^8b^8 according to the grammar in Example 10.1: symbols are marked with numbers as $a_1 \ldots a_8 b_8 \ldots b_1$; the grey area marks the subtree with a hole represented by the conditional proposition $\frac{S}{S}(a_1 \ldots a_4 : b_4 \ldots b_1)$.

Example 10.1. *Consider the usual grammar for the language $\{\, a^n b^n \mid n \geqslant 0\}$.*

$$S \to aSa \mid bSb \mid \varepsilon$$

The parse tree for the string $w = a^8b^8$ is given in Figure 10.1. The next Figure 10.2 illustrates a shallow proof of the same proposition $S(aaaaaaaa\,bbbbbbbb)$. The last step of this shallow proof is

$$\frac{S}{S}(a^4 : b^4), S(a^4b^4) \vdash S(a^8b^8).$$

In Figure 10.1, it is shown as the subtree with a hole representing $\frac{S}{S}(a^4 : b^4)$ (a combination of the grey area) and the regular subtree representing $S(a^4b^4)$.

10.1.2 Recognition in space $(\log n)^2$

Theorem 10.1. *For every ordinary grammar G in the Chomsky normal form, the algorithm correctly determines whether a given string of length n is in $L(G)$. Its depth of recursion is*

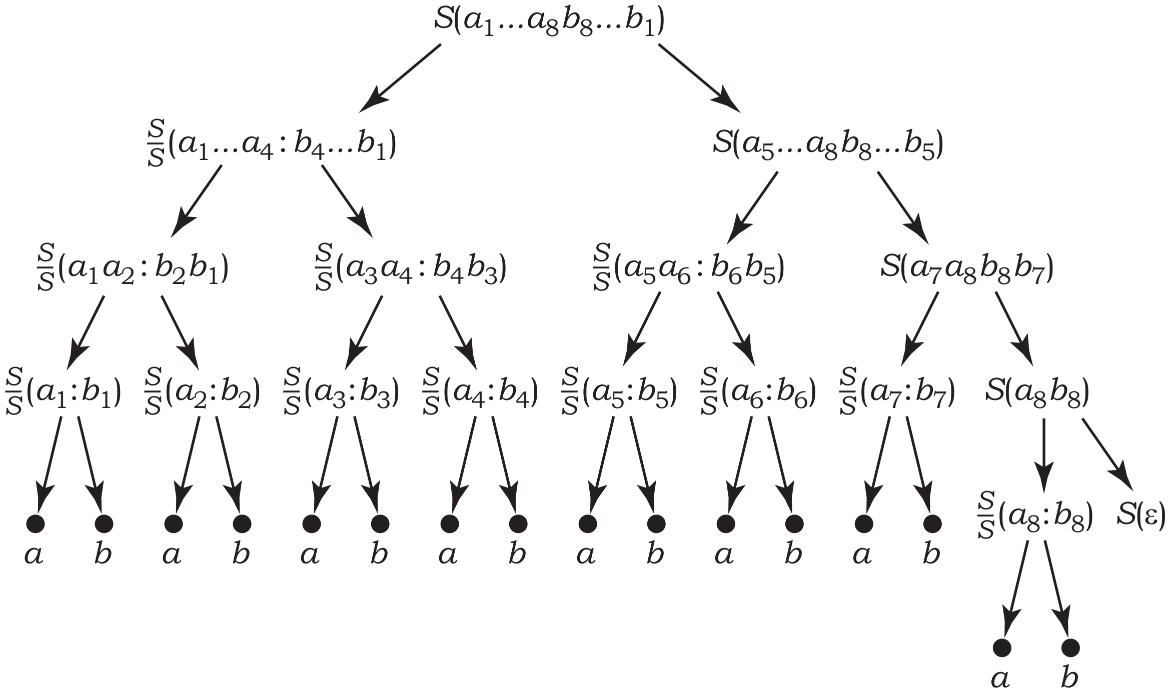

Figure 10.2: A shallow proof for the string a^8b^8 according to the grammar in Example 10.1; symbols are marked with numbers as $a_1 \ldots a_8b_8 \ldots b_1$.

$O(\log n)$ and each stack frame uses $O(\log n)$ bits, to the total of $O((\log n)^2)$ bits. The running time is $n^{O(\log n)}$.

The running time is super-polynomial, because the same propositions are reproved multiple times; the algorithm lacks memory to remember any intermediate results. The next subsection shows essentially the same computations implemented on an entirely different model of computation, which does not have memory restrictions.

10.1.3 Recognition by a circuit of depth $(\log n)^2$

The existence of a recognizer circuit of depth $(\log n)^2$ with polynomially many gates was first discovered by Ruzzo [10]. The Brent–Goldschlager–Rytter algorithm, given independently by Brent and Goldschlager [1] and by Rytter [11], constructs such a circuit with $O(n^6)$ gates.

Algorithm 10.1 The Lewis–Stearns–Hartmanis algorithm

Let $G = (\Sigma, N, R, S)$ be an ordinary grammar in the Chomsky normal form. Let $w = a_1 \ldots a_n$, with $n \geqslant 1$ and $a_i \in \Sigma$, be an input string. The algorithm is comprised of the following recursive procedures:

- for each $A \in N$, the procedure $A(i, j, h)$ tests whether $A(a_{i+1} \ldots a_j)$ has a proof of height at most h in the extended deduction system;
- for all $A, D \in N$, the procedure $\frac{A}{D}(i, k, \ell, j, h)$ tests whether $\frac{A}{D}(a_{i+1} \ldots a_k : a_{\ell+1} \ldots a_j)$ can be proved in this system in height h.

Each procedure tries all possibilities of deducing the proposition, and decrements h as it makes recursive calls. For $h = 0$ each procedure returns `false`.

This algorithm came unforeseen: Cook [3] wrote "I see no way of showing $DCFL \subseteq NC$" (where $DCFL$ denotes the class of deterministic languages).

Theorem 10.2 (Ruzzo [10], Brent and Goldschlager [1]; Rytter [11])**.** *For every ordinary grammar $G = (\Sigma, N, R, S)$ in the Chomsky normal form and for every number $n \geqslant 1$, there is a Boolean circuit of depth $O(\log n)$, which has $|\Sigma| \cdot n$ inputs to read a string $w = a_1 \ldots a_n$ with $a_i \in \Sigma$, $O(n^6)$ intermediate Boolean gates, and one output to report whether w is in $L(G)$.*

For each grammar, there is a logarithmic-space Turing machine, which, given a number n, prints this circuit.

The circuit contains the following gates:

- for all i, j with $0 \leqslant i < j \leqslant n$, a gate $x_{A,i,j}$, which computes the truth value of $A(a_{i+1} \ldots a_j)$, that is, whether the substring $a_{i+1} \ldots a_j$ is in $L_G(A)$.

- $y_{A,i,j,D,k,\ell}$ with $A, D \in N$, $0 \leqslant i \leqslant k < \ell \leqslant j \leqslant n$ and $(k-i)+(j-\ell) \geqslant 0$. Such a gate represents a parse tree of $a_{i+1} \ldots a_j$ from A with a hole instead of a subtree of $a_{k+1} \ldots a_\ell$ from D, so that it evaluates to `true` if and only if the conditional proposition $\frac{A}{D}(a_{i+1} \ldots a_k : a_{\ell+1} \ldots a_j)$ is true.

Research problems

10.1.1. Reconstruct the circuit in Theorem 10.2 to use $o(n^6)$ gates, and generally as few gates as possible, while maintaining depth $O((\log n)^2)$. Brent and Goldschlager [1] conjectured that the circuit could be reconstructed by embedding a subcircuit implementing fast Boolean matrix multiplication. This would require a careful analysis of the original circuit with $\Theta(n^6)$ gates, as well as of any subcircuit implementing matrix multiplication.

10.4 Linear grammars and logarithmic space

Computational complexity class NLOGSPACE (also called NL): problems solvable on a nondeterministic two-tape Turing machine, with a read-only input tape containing an input string of length n, and with a work tape of size $\log_2 n$.

10.4.1 Uniform membership problem for linear grammars

Definition 10.1. *The uniform membership problem for a family of grammars $\mathcal{G}$: "Given a grammar $G \in \mathcal{G}$ and a string $w \in \Sigma^*$, where Σ is the alphabet, over which G is defined, determine whether w is in $L(G)$".*

Theorem 10.3. *The uniform membership problem for linear grammars is in NLOGSPACE.*

10.4.2 A complete problem for nondeterministic logarithmic space

Complete problems for NLOGSPACE are defined with respect to reductions made by uniformly generated circuits of depth $\log_2 n$, called NC^1 reductions.

Reachability in a directed ordered graph: given a graph with a set of vertices $\{1, \ldots, n\}$ and with a set of arcs (i, j), with $i < j$, determine whether there is a directed path from vertex 1 to vertex n.

Is NLOGSPACE-complete.

10.4.3 An NLOGSPACE-complete linear language

Theorem 10.4 (Sudborough [12]). *For every alphabet Σ with [,], $\# \notin \Sigma$ and for every language $L \subseteq \Sigma^*$, define*

$$f(L) = \{\, [w_{1,1}\#\ldots\#w_{1,k_1}] \ldots [\mathrm{w_{m,1}\#\ldots\#w_{m,k_m}}] \mid \exists \mathrm{i_1, \ldots, i_m :\ w_{1,i_1} w_{2,i_2} \ldots w_{m,i_m} \in L} \,\}.$$

Let $L_0 = \{\, w\$w^R \mid w \in \{a,b\}^* \}$ Then the language $f(L_0)$ is generated by a linear grammar and is NLOGSPACE-complete.

In a string belonging to the language $f(L)$, each block delimited by square brackets lists one or more choices of substrings, and for some set of choices $(i_1, \ldots, i_m)$, the concatenation of these substrings must belong to L.

Reduction from the graph reachability problem.

Given an acyclic graph with n nodes $\{1, \ldots, n\}$ and a set of arcs E, where $(i,j) \in E$ implies $i < j$, construct the following string.

$$[a^1b] \prod_{i=1}^{n} \Big([\Big(\prod_{j:(i,j)\in E} \#a^iba^jb \Big)] \Big) [a^nb\$]\ [\#ba^nba^n]\ [\#ba^{n-1}ba^{n-1}] \ldots [\#ba^1ba^1]$$

For this string to belong to the language $f(L_0)$, for some choices of substrings at each block delimited by square brackets, the concatenation of these choices must be a string of the form $w\$w$, with $w \in \{a,b\}$. The first block gives no alternative: the string must begin with a^1b. Therefore, it must end with ba^1, and this can only be achieved if the substring ba^1ba^1 is chosen in the last block (the alternative there would be to choose the empty string). This in turn forces the left part of the string to continue with a_1b, which choosing one of the arcs $(1,j) \in E$ and taking the alternative a^1ba^jb in the second block. At the moment, the string has the following form.

$$a_1ba_1ba_jb \ldots ba^1ba^1$$

Next, the right part of the string should have ba^jba^j, and therefore the right part must continue with an arc from j to some node, etc., etc. This construction ends with an inner substring $a^nb\$$, which indicates the end of the path in the node a^n.

Example:

$$[a^1b]\,[\#a^1ba^2b\#a^1ba^3b][\#a^2ba^3b\#a^2ba^4b][\#a^3ba^4b][a^4b\$][\#ba^4ba^4][\#ba^3ba^3][\#ba^2ba^2][\#ba^1ba^1]$$

10.4.4 Deterministic linear grammars

LR(1) linear grammars have a DLOGSPACE-complete uniform membership problem. Holzer and Lange [5] constructed an LR(1) linear grammar that defines a DLOGSPACE-complete language.

10.5 Polynomial-time completeness

10.5.1 The circuit value problem

Problems complete for the polynomial time (P-complete problems) are defined with respect to logarithmic-space reductions.

The basic P-complete problem is the problem of testing whether a given Boolean circuit with no inputs and a single output calculates the value 1, known as the *Circuit Value Problem (CVP)*, defined by Ladner [7].

Such a circuit is given as a finite collection of gate definitions, where the first two gates are

$$\begin{aligned} C_0 &= 0, \\ C_1 &= 1, \end{aligned}$$

and each of the subsequent gates $C_2, \ldots, C_n$ is defined as a conjunction or a disjunction of any two earlier gates,

$$\begin{aligned} C_i &= C_j \vee C_k && (i \geqslant 2;\ j,k < i) \\ C_i &= C_j \wedge C_k && (i \geqslant 2;\ j,k < i) \end{aligned}$$

or as a negation of any single earlier gate:

$$C_i = \neg C_j \qquad (i \geqslant 2;\ j < i)$$

The question is, whether the last of these gates evaluates to 1.

A special case of this problem is the *Monotone Circuit Value Problem (MCVP)*, due to Goldschlager [4], in which the input circuit does not use any negation gates. This problem is remains P-complete.

Theorem 10.5 (Ladner [7], with improvements by Goldschlager [4])**.** *For every Turing machine M with an input alphabet Σ running in polynomial time, there exists a logarithmic-space deterministic transducer T_M, which, given an input string $w \in \Sigma^*$, produces such a monotone circuit $T_M(w) = (C_0, C_1, \ldots, C_m)$, that C_m evaluates to 1 if and only if M accepts w.*

10.5.2 Uniform membership problem for ordinary grammars

Theorem 10.6. *P-complete for Boolean grammars, remains P-complete for LL(1) ordinary grammars and the membership of ε.*

Proof. Reduction from the unambiguous MCVP. □

10.5.3 Conjunctive grammar for a P-complete language

First: ensure a circuit with each gate of the form

$$\begin{aligned} C_0 &= 0 \\ C_1 &= 1 \\ C_i &= C_{i-1} \vee C_{j_i} && (j_i < i) \\ C_i &= C_{i-1} \wedge C_{j_i} && (j_i < i) \end{aligned}$$

Defined by the following conjunctive grammar.

$$\begin{aligned} S &\to cAS \& cBS \mid dAS \mid dBS \mid b \\ A &\to aA \mid \varepsilon \\ B &\to aBXA \mid \varepsilon \\ X &\to c \mid d \end{aligned}$$

Can have a linear conjunctive grammar for a similar language. Also, a Boolean LL(1) grammar.

10.7 Representation of the polynomial time by first-order grammars

Representing any language recognized in polynomial time by a first-order grammar. This result was independently obtained by Immerman [6] and by Vardi [13], and adapted to formal grammars by Rounds [9]. The proofs by Immerman and by Vardi worked by simulating an intermediate theoretical model (an alternating logaritmic-space Turing machine), which was proved to be equal to the polynomial time by Chandra, Kozen and Stockmeyer [2]. The work by all these authors is combined into the following theorem.

Theorem 10.7. *For every Turing machine M over any input alphabet Σ recognizing a language $L \subseteq \Sigma^*$ in time $O(n^k)$, there exists and can be effectively constructed a first-order grammar $G = (\Sigma, N, \mathrm{rank}, \langle\varphi_A\rangle_{A \in N}, \sigma)$, that defines the language $L(M)$, in which the largest rank of a predicate is $2k$ and no quantifiers are used.*

Proof. Let Γ, with $\Sigma \subset \Gamma$, be the work alphabet of the Turing machine, let Q be its set of states, with the initial state q_0, accepting state q_{acc} and rejecting state q_{rej}. The transition function is $\delta \colon Q \times \Gamma \to Q \times \Gamma \times \{-1, 0, +1\}$.

Assume that after entering the accepting state q_{acc}, the machine moves to the leftmost square in the state q_{acc}, and stays there indefinitely. Also assume that the machine, given an input string of length n, uses time exactly $(n+1)^k$, rather than $const \cdot n^k$; this can be ensured by the speed-up theorem. Then the machine M also uses space at most n^k, since it does not have time to use more.

This upper bound allows encoding any position on the tape and any number of a step as a k-tuple $(x_1, \ldots, x_k)$ of positions in the input string: any such k-tuple encodes a number $\sum_{i=1}^{k} x_i \cdot (n+1)^{i-1}$, and any integer between 0 and $(n+1)^k - 1$ is representable in this way. Simulation of a Turing machine requires adding 1 to such a representation, as well as subtracting 1 from it. In order to add 1 to $(x_1, \ldots, x_k)$, let ℓ be the least such number, that $x_\ell < n$ (if there is no such number, then $(x_1, \ldots, x_k) = (n, \ldots, n)$ is the largest representable value, which cannot be incremented). Denote this condition by

$$\nu_\ell(x_1, \ldots, x_k) = x_1 = \underline{\mathrm{end}} \wedge \ldots \wedge x_{\ell-1} = \underline{\mathrm{end}} \wedge x_\ell < \underline{\mathrm{end}}$$

If this condition holds, then it is sufficient to replace $x_1, \ldots, x_{\ell-1}$ by zeroes and add 1 to x_ℓ. Similarly, subtracting 1 from $(x_1, \ldots, x_k)$ requires checking the condition

$$\nu_\ell(x_1, \ldots, x_k) = x_1 = \underline{\mathrm{begin}} \wedge \ldots \wedge x_{\ell-1} = \underline{\mathrm{begin}} \wedge x_\ell > \underline{\mathrm{begin}}$$

and then replacing $x_1, \ldots, x_{\ell-1}$ by n and subtracting 1 from x_ℓ.

For every state $q \in Q$, the grammar defines a predicate $A_q(x_1, \ldots, x_k, y_1, \ldots, y_k)$, which states that at the time $(x_1, \ldots, x_k)$ the Turing machine was in the state q, and its head was in the position $(y_1, \ldots, y_k)$ on the tape. Another predicate $C_a(x_1, \ldots, x_k, y_1, \ldots, y_k)$, defined for each symbol $a \in \Gamma$ writeable on the work tape, states that at the time $(x_1, \ldots, x_k)$, the square number $(y_1, \ldots, y_k)$ on the tape contained the symbol a.

$$\begin{aligned}
A_q(x_1, \ldots, x_k, y_1, \ldots, y_k) = \Big(& \bigvee_{\substack{q' \in Q,\, a, a' \in \Sigma: \\ \delta(q', a') = (q, a, +1)}} \; \bigvee_{\ell=1}^{k} \bigvee_{\ell'=1}^{k} \mu_\ell(x_1, \ldots, x_k) \wedge \mu_{\ell'}(y_1, \ldots, y_k) \wedge \\
& \wedge A_{q'}(\underline{\mathrm{end}}, \ldots, \underline{\mathrm{end}}, x_\ell - 1, x_{\ell+1}, \ldots, x_k, \underline{\mathrm{end}}, \ldots, \underline{\mathrm{end}}, y_{\ell'} - 1, y_{\ell'+1}, \ldots, y_k) \wedge \\
& \wedge C_{a'}(\underline{\mathrm{end}}, \ldots, \underline{\mathrm{end}}, x_\ell - 1, x_{\ell+1}, \ldots, x_k, \underline{\mathrm{end}}, \ldots, \underline{\mathrm{end}}, y_{\ell'} - 1, y_{\ell'+1}, \ldots, y_k) \Big) \\
& \vee \Big(\text{two other cases, whether the machine moves right or stays}\Big)
\end{aligned}$$

definition of $C_a(x_1, \ldots, x_k, y_1, \ldots, y_k)$ TBW

Finally, define the initial formula as

$$\sigma = A_{q_{acc}}(\underbrace{\underline{\text{end}}, \ldots, \underline{\text{end}}}_{k}, \underbrace{\underline{\text{begin}}, \ldots, \underline{\text{begin}}}_{k})$$

□

Corollary 10.1 (Chandra, Kozen and Stockmeyer [2]; Immerman [6]; Vardi [13]; Rounds [9])**.** *A language is defined by a first-order grammar if and only if it is recognized by a Turing machine in polynomial time.*

Corollary 10.2. *The uniform membership problem for first-order grammars is not decidable in polynomial time.*

Bibliography

[1] R. P. Brent, L. M. Goldschlager, "A parallel algorithm for context-free parsing", *Australian Computer Science Communications*, 6:7 (1984), 7.1–7.10.

[2] A. K. Chandra, D. Kozen, L. J. Stockmeyer, "Alternation", *Journal of the ACM*, 28:1 (1981), 114–133.

[3] S. A. Cook, "Deterministic CFL's are accepted simultaneously in polynomial time and log squared space", *11th Annual ACM Symposium on Theory of Computing* (STOC 1979, April 30–May 2, 1979, Atlanta, Georgia, USA), 338–345.

[4] L. M. Goldschlager, "The monotone and planar circuit value problems are log space complete for P", *SIGACT News*, 9:2 (1977), 25–29.

[5] M. Holzer, K.-J. Lange, "On the complexities of linear LL(1) and LR(1) grammars", *Fundamentals of Computation Theory* (FCT 1993, Hungary, August 23–27, 1993), LNCS 710, 299–308.

[6] N. Immerman, "Relational queries computable in polynomial time", *Information and Control*, 68:1–3 (1986), 86–104.

[7] R. E. Ladner, "The circuit value problem is log space complete for P", *SIGACT News*, 7:1 (1975), 18–20.

[8] P. M. Lewis II, R. E. Stearns, J. Hartmanis, "Memory bounds for recognition of context-free and context-sensitive languages", *IEEE Conference Record on Switching Circuit Theory and Logical Design*, 1965, 191–202.

[9] W. C. Rounds, "LFP: A logic for linguistic descriptions and an analysis of its complexity", *Computational Linguistics*, 14:4 (1988), 1–9.

[10] W. L. Ruzzo, "Tree-size bounded alternation", *Journal of Computer and System Sciences*, 21:2 (1980), 218–235.

[11] W. Rytter, "On the recognition of context-free languages", *Fundamentals of Computation Theory* (FCT 1985, Cottbus, Germany), LNCS 208, 315–322.

[12] I. H. Sudborough, "A note on tape-bounded complexity classes and linear context-free languages", *Journal of the ACM*, 22:4 (1975), 499–500.

[13] M. Y. Vardi, "The complexity of relational query languages", *STOC 1982*, 137–146.

Index