UPMAIL Technical Report No. 133
January 16, 1997
ISSN 1100–0686

On Computational Complexity of Basic Decision Problems of Finite Tree Automata

Margus Veanes

email: margus@csd.uu.se

Computing Science Department
Uppsala University

Box 311, S-751 05 Uppsala, Sweden
Phone: +48−18−18 25 00
Fax: +46−18−51 19 25

Abstract

This report focuses on the following basic decision problems of finite tree automata: *nonemptiness* and *intersection nonemptiness*. There is a comprehensive proof of EXPTIME-completeness of the intersection nonemptiness problem, and it is shown that the nonemptiness problem is P-complete. A notion of succinctness is considered with respect to which the intersection nonemptiness problem is in fact a *succinct* version of the nonemptiness problem. The report includes a short survey of closely related problems which shows that there is a rule of thumb: *if a decision problem for (deterministic) finite automata is complete for a certain space complexity then the same decision problem for (deterministic) finite tree automata is complete for the corresponding alternating space complexity*, but alternating space is precisely deterministic time, only one exponential higher.

1 Introduction

Finite tree automata [14, 51] is a natural generalization of classical finite automata to automata that accept or recognize trees of symbols, not just sequences of symbols or strings. In the deterministic case, this generalization is best understood by first looking at a deterministic finite automaton with input alphabet Σ as a finite $(\{\epsilon\} \cup \Sigma)$-algebra with its elements as states, where ϵ is a constant and the symbols in Σ are unary function symbols.[1] The generalization consists of arbitrary (not just unary) function symbols in Σ. The recognizability condition of a ground (or closed) term is, like in the unary case, simply that its value is a final state.

Many decision problems concerned with finite automata (nonemptiness, inequivalence, etc.) have natural counterparts with finite tree automata. Like in the case of finite automata, decision problems of finite tree automata are typically complete for the computational complexity classes they belong to and have, due to their simple formulation, proved to be useful tools in classifying complexity bounds of other problems. In particular, inequivalence [44, 46] and intersection nonemptiness [7, 15, 47] are examples of such decision problems.

The intersection nonemptiness problem of finite tree automata arises naturally in the context of type inference in logic programming [15]. The same decision problem restricted to top-down deterministic finite tree automata arises also in sort inference in typed functional programming [47]. Our main motivation for studying this problem is its close connection with a certain fragment of intuitionistic logic with equality and a subcase of a problem called simultaneous rigid E-unification, or SREU, that arises in the automated theorem proving context [19]. These connections are investigated in a separate joint paper by Degtyarev, Gurevich, Narendran, Veanes and Voronkov [7]. Until SREU was proved undecidable by Degtyarev and Voronkov [9, 10, 11, 12] there appeared many faulty proofs of its decidability [17, 18, 22].

The main contributions of this report can be summarized as follows. We present a comprehensive proof of *EXPTIME-completeness of the intersection nonemptiness problem of finite tree automata*. More precisely, it is proved that the hardness result holds already for *deterministic* finite (bottom-up) tree automata. Although the complexity of this problem has been used in the above mentioned contexts and also in the context of a "decidability" proof of SREU [22], its proof is either merely remarked upon [15], or only briefly outlined and incomplete [22, 47]. The proof of its complexity is however highly nontrivial and in order to trust it we had to prove it ourselves. In general, it was very hard to find complexity results related to the basic decision problems of finite tree automata, as they are scattered throughout the literature, and we decided to make a short survey by collecting the complexity results of the closely related problems. This survey is summarized with Table 1 in the conclusions. We show also that the *nonemptiness problem of finite tree automata is P-complete* by showing its close connection with the two well-known P-complete problems *alternating graph accessibility* [24, 28] and *generability* [24, 30, 31]. We consider a notion of succinctness with respect to which the intersection nonemptiness problem is in fact a *succinct* version of the nonemptiness problem. We believe that these decision problems of finite tree automata will appear in other contexts, and expect that this survey will be useful therein.

In general there is a rule of thumb saying that *if a decision problem for (deterministic) finite automata is complete for a certain space complexity class, then the same decision problem for (deterministic) finite tree automata is complete for the corresponding* alternating *space complexity class*, but alternating space is precisely

[1] The idea is that the interpretation of ϵ is the initial state and that the interpretation of a unary function symbol σ is a function $\boldsymbol{\sigma}$ such that there is a transition with label σ from a state q to a state p iff $\boldsymbol{\sigma}(q) = p$. So the value of a term $\sigma_1(\sigma_2(\cdots\sigma_n(\epsilon)))$ is the state after reading the string $\sigma_n \cdots \sigma_2\sigma_1$. This observation is attributed to Büchi and Wright [1].

deterministic time, only one exponential higher [3].

The rest of the report is organized as follows. In Section 2 we recall the main notions used in this report, namely finite tree automata and alternating Turing machines. In Section 3 we introduce the basic decision problems of finite tree automata that are considered in this report. In Section 4 we show that the nonemptiness problem is P-complete and in Section 5 that the intersection nonemptiness problem is EXPTIME-complete. In Section 6 the report is summarized and we present a table of the computational complexities of the closely related problems.

2 Preliminaries

We introduce here the main notions and definitions used in this report. Given a signature Σ, i.e., a set of function symbols with fixed arities, the set of all ground (or closed) terms over Σ is denoted by $\mathcal{T}_\Sigma$. Unless otherwise stated it is always assumed that Σ is nonempty, finite and includes at least one constant (function symbol of arity 0). We will also assume certain familiarity with some basic notions from term rewriting [13], regarding ground rewriting systems.

2.1 Finite Tree Automata

Finite tree automata, or simply tree automata from here on, is a generalization of classical automata and were first studied by Doner [14] and independently by Thatcher and Wright [51]. The main motivation was to obtain decidability results for the weak monadic second order logic of the binary tree. A remarkable feature of tree autmata is that they provide an alternative characterization of both the class of regular sets and the class context-free languages (see Doner [14]). Here we adopt a definition of tree automata based on rewrite rules. This definition is used for example by Conquidé et al [4] and Dauchet [6].

- A *tree automaton* or *TA* A is a quadruple (Q, Σ, R, F) where
  - Q is a finite set of *states*,
  - Σ is a *signature* or an *input alphabet*,
  - R is a set of *rules* of the form $\sigma(q_1, \ldots, q_n) \to q$, where $\sigma \in \Sigma$ has arity $n \geq 0$ and $q, q_1, \ldots, q_n \in Q$,
  - $F \subseteq Q$ is the set of *final states*.

  A is called a *deterministic* TA or DTA if there are no two different rules in R with the same left hand side.

It is also assumed that Q and Σ are disjoint. Note that if A is deterministic then R is a reduced set of ground rewrite rules, i.e., for any rule $s \to t$ in R t is irreducible and s is irreducible with respect to $R \setminus \{s \to t\}$. So R is a ground canonical rewrite system. Tree automata as defined above are usually also called *bottom-up* tree automata. Top-down tree automata were introduced by Rabin [42] and were also studied by Magidor and Moran [34]. Here we will use the following definition based on rewrite rules.

- A *top-down tree automaton* or *TTA* A is a quadruple (Q, Σ, R, I) where Q and Σ are like above,
  - R is a set of *rules* of the form $q \to \sigma(q_1, \ldots, q_n)$, where $\sigma \in \Sigma$ has arity $n \geq 0$ and $q, q_1, \ldots, q_n \in Q$,
  - $I \subseteq Q$ is the set of *initial states*.

  A is called a *deterministic* TTA or DTTA if I is a *singleton set*, and whenever $q \longrightarrow_R \sigma(\vec{q})$ and $q \longrightarrow_R \sigma(\vec{p})$ then $\vec{q} = \vec{p}$.

Terms are also called trees. A set of terms (or trees) is called a *forest*. Acceptance for tree automata or recognizability is defined as follows.

- The forest *recognized* by a TA $A = (Q, \Sigma, R, X)$ (or a TTA $A = (Q, \Sigma, R^{-1}, X)$) is the set
$$T(A) = \{\, \tau \in \mathcal{T}_\Sigma \mid (\exists q \in X)\ \tau \xrightarrow{*}_R q \,\}.$$
A forest is called *recognizable* if it is recognized by some TA (or TTA).

Two tree automata are *equivalent* if they recognize the same forest. It is wellknown that the nondeterministic and the deterministic versions of TAs have the same expressive power [14, 21, 51], i.e., for any TA there is an equivalent DTA. Clearly there is no essential difference between a TA and a TTA. However, the class of forests recognized by DTTAs are properly contained in the class of all recognizable forests. A simple example of that is the forest $\{f(a,b), f(b,a)\}$ that is clearly recognizable but not by any DTTA [21, Example 2.11].

We say that a TA is *total* if every term over its input alphabet reduces to some state. Every TA can trivially be extended (by adding new rules and a new dummy state) to an equivalent total TA. Every total DTA $A = (Q, \Sigma, R, F)$ can be seen as a pair $(\mathfrak{A}, F)$, where $\mathfrak{A}$ is a Σ-algebra with universe Q whose interpretation function is determined by R as follows: for all $f \in \Sigma$ (of arity n) and $q, q_1, \ldots, q_n \in Q$,

$$f^{\mathfrak{A}}(q_1, \ldots, q_n) = q \quad \Leftrightarrow \quad f(q_1, \ldots, q_n) \longrightarrow_R q.$$

Then we have that

$$T(A) = \{\, \tau \in \mathcal{T}_\Sigma \mid \tau^{\mathfrak{A}} \in F \,\}. \tag{1}$$

Conversely, any pair $(\mathfrak{A}, F)$ where $\mathfrak{A}$ is a finite Σ-algebra and F a subset of its universe, can be seen as a DTA. This is actually the definition of a DTA used by Gécseg and Steinby [21]. For an overview of the notion of recognizability in general algebraic structures see Courcelle [5] and the fundamental paper by Mezei and Wright [38].

We will refer to (classical) nondeterministic finite automata as NFAs and to deterministic finite automata as DFAs. In general, we will follow Hopcroft and Ullman [26] regarding the formal definitions and notational conventions of finite automata.

2.2 Alternation and Computational Complexity

Alternation was introduced by Chandra, Kozen and Stockmeyer [3] as a generalization of nondeterminism. First, let us give an intuitive definition of an alternating Turing machine or ATM. An ATM is like a nondeterministic Turing machine (TM), except that every configuration or instantaneous description (ID) is labelled as either "universal" or "existential", actually each state is either universal or existential and an ID is labelled accordingly.[2] We inductively determine if an ID "leads to acceptance" as follows. Any final ID leads to acceptance. For any nonfinal ID we have two cases: an existential ID leads to acceptance if at least one of its successors leads to acceptance; a universal ID leads to acceptance if all of its successors lead to acceptance and it has at least one successor.

All computation models based on a Turing machine can be considered as variants of a TM with different acceptance conditions, this point is emphasized by Johnson [29]. We follow Hopcroft and Ullman [26] regarding the formal definition of a nondeterministic Turing machine. For the sake of clearness we recall here the main definitions.

- A *nondeterministic Turing machine* M is a 7-tuple $(Q, \Sigma_{\text{in}}, \Sigma, \delta, q_0, \text{b}, F)$, where
  - Q is a finite set of *states*,

[2] In the original definition of an ATM there is also a possibility of a "negated" state, but it can be omitted without loss of generality [3, Theorem 2.5].

- Σ is a finite set of *tape symbols*,
- $\b{b}$ is a tape symbol called *blank*,
- Σ_{in} is a subset of Σ called the set of *input symbols*,
- δ is a mapping from $Q \times \Sigma$ to subsets of $Q \times \Sigma \times \{\texttt{left}, \texttt{right}\}$, and is called the *transition function* of M,
- q_0 is the *initial* state of M, and
- $F \subseteq Q$ is the set of *final states*.

By an *instantaneous description* or *ID* of M we mean any string vqw where $q \in Q$ is a state of M and $vw \in \Sigma^*$ (the position of q marks that the tape head points to the first symbol in w).

An ID w is a *successor* of a nonfinal ID v, in symbols $v \vdash w$, if w follows from v in one step according to the transition function of M.

We define an ATM formally as follows.

▶ An *alternating Turing machine* is a pair (M, U) where M is a TM and U a subset of the states of M, called the set of *universal states*. The states of M not in U are called *existential*.

An ATM with an empty set of universal states is simply a TM. An ID of an ATM is said to be *existential* (respectively *universal*, *final*, *initial*) if its state is existential (respectively universal, final, initial). We can now formally define the notion of acceptance for ATMs.

▶ Let M be an ATM with initial state q_0 and x a string over its input alphabet. Then M *accepts* x iff the initial ID q_0x, leads to acceptance, where *leads to acceptance* is defined recursively as follows.

- Any final ID leads to acceptance.
- If v is a nonfinal ID then it leads to acceptance iff
  - v is existential and some successor of v leads to acceptance, or
  - v is universal, all successors of v lead to acceptance and v has at least one successor.

Note that the acceptance condition of an ATM without universal states is the same as the acceptance condition of the underlying TM.

Alternating Space vs Deterministic Time The notion of space (and time) complexity of ATMs is the same as that of TMs. The key property that we are going to use is that, *alternating space is precisely deterministic time, only one exponential higher* [3]. In particular,

- APSPACE = EXPTIME,
- ALOGSPACE = P,

where the classes APSPACE and ALOGSPACE consist of all problems that can be solved by a polynomial space ATM and a logarithmic space ATM, respectively. The class EXPTIME consist of all problems that can be solved by a deterministic TM that is time bounded by 2^{n^c} for some $c > 0$. For a general overview of the relationships between EXPTIME and other complexity classes see Johnson [29] or Papadimitriou [40].

3 Basic Decision Problems of Finite Tree Automata

All the basic decision problems of finite tree automata, like the *nonemptiness problem*, the *inequivalence problem* (or the more general *inclusion problem*) are decidable (see Gécseg and Steinby [21]). The proofs are fairly easy by first transforming a TA into a DTA by a powerset construction and then using a "pumping property" for DTAs. It is also easy to show that recognizable sets of terms are closed under Boolean operations. This is illustrated next.

- **Complementation:** Let $A = (Q, \Sigma, R, F)$ be a total DTA. The *complement* of A is the DTA $\bar{A} = (Q, \Sigma, R, Q \setminus F)$. It follows immediately from (1) that $T(\bar{A}) = \mathcal{T}_\Sigma \setminus T(A)$.

- **Intersection:** Let $A = (Q_1, \Sigma, R_1, F_1)$ and $B = (Q_2, \Sigma, R_2, F_2)$ be TAs. The *direct product* of A and B is the TA $A \times B = (Q_1 \times Q_2, \Sigma, R, F_1 \times F_2)$, where

$$R = \{\, f((a_1, b_1), \ldots, (a_n, b_n)) \to (a, b) \mid f(\vec{a}) \to a \in R_1,\ f(\vec{b}) \to b \in R_2 \,\}$$

  It follows easily that $T(A \times B) = T(A) \cap T(B)$.

Note that if A and B above are total DTAs then so is their direct product. Let A and B be total DTAs. Clearly the inclusion and inequivalence problems for DTAs reduce effectively to the nonemptiness problem, since $T(A) \subseteq T(B)$ iff $T(A) \cap T(\bar{B}) = \emptyset$. It follows for example that

$$\begin{aligned} T(A) = T(B) \quad \Leftrightarrow \quad & (T(A) \cap T(\bar{B})) \cup (T(B) \cap T(\bar{A})) = \emptyset \\ & T(\overline{\overline{A \times \bar{B}} \times \overline{B \times \bar{A}}}) = \emptyset \end{aligned} \tag{2}$$

In the following two sections we will address the following decision problems.

- ▶ *Nonemptiness of TAs* (or, more particularly, of DTAs or DTTAs) is the following decision problem: Given a finite tree automaton A, is $T(A)$ nonempty?

- ▶ *Inequivalence of TAs* (or, more particularly, of DTAs or DTTAs) is the following decision problem: Given finite tree automata A and B with the same signature, are $T(A)$ and $T(B)$ unequal?

- ▶ *Intersection nonemptiness of TAs* (or, more particularly, of DTAs or DTTAs) is the following decision problem: Given a finite sequence $(A_i)_{i<n}$ of finite tree automata, is $\bigcap_{i<n} T(A_i)$ nonempty?

For finite automata the same decision problems are defined analogously. It is clear that, by using (2), inequivalence of DTAs reduces (in logarithmic space) to nonemptiness [21]. For DFAs this was already shown by Moore [39]. It is also clear that for a fixed n, the intersection nonemptiness problem reduces to the nonemptiness problem in logarithmic space.

4 Nonemptiness and Inequivalence of Finite Tree Automata

For finite automata (either deterministic or nondeterministic) the nonemptiness problem is basically the same as the *graph accessibility problem* and is thus complete for nondeterministic logarithmic space or NL-complete [45]. It follows that the inequivalence problem of DFAs is also NL-complete. Analogously, for finite tree automata there is a simple reduction from the *alternating graph accessibility problem* to the nonemptiness problem and vice versa. Alternating graph accessibility was shown P-complete by Immerman [28] by a direct simulation of any ALOGSPACE ATM. There is also a very simple reduction from *generability*, which is another P-complete problem due to Jones and Laaser [30] and Kozen [31], to nonemptiness of DTAs and

vice versa. We follow Greenlaw, Hoover and Ruzzo [23, 24] in our formulation of alternating graph accessibility and generability.[3]

- *Alternating graph accessibility.* Given is a directed graph with a set of vertices V and a set of edges E, a subset U of V, and designated vertices a and b in V. The vertices in U are called *universal* and those in $V \setminus U$ are called *existential*.

  The problem is to decide if $apath(a, b)$ holds, where, for any two vertices x and y, $apath(x, y)$ is true if either

  1. $x = y$, or
  2. x is existential and there exists a vertex z with $(x, z) \in E$ and $apath(z, y)$ is true, or
  3. x is universal and for all vertices z with $(x, z) \in E$, $apath(z, y)$ is true.

- *Generability.* Given is a finite set Q, (the graph of) a binary function $\boldsymbol{f}$ on Q, a subset V of Q and an element q in Q.

  The problem is to decide if q is in the smallest subset of Q that includes V and is closed under $\boldsymbol{f}$.

The generability problem remains in P even with more than one function. More generally, it is the problem of deciding if, given a finite algebra, a subset of its universe and an element in it, this element is in the subalgebra generated by the given subset [31]. (See for example Wechler [54] for definitions.) Actually, as we will see, generability is basically the same problem as nonemptiness of DTAs. In the following proof it is easily seen that all reductions can be carried out within logarithmic space, assuming reasonable representations of the problems, and we will not mention that explicitly.

Theorem 1 *Nonemptiness of DTTAs, DTAs and TAs is P-complete.*

Proof. First we show how alternating graph accessibility reduces to nonemptiness of DTTAs. Consider a directed graph $G = (V, E)$ a subset U of V of universal vertices, and two designated vertices a and b in V. We can assume without loss of generality that the out-degree of any vertex in G is either two or zero. Le A be the TTA $(V, \Sigma, R, \{a\})$, where $\Sigma = \{c, g_1, g_2, f\}$, c is a constant, g_1, g_2 unary function symbols, and f a binary function symbol. Let the rules of A be as follows:

1. $b \longrightarrow_R c$,
2. for each vertex x and edges $(x, y_1), (x, y_2) \in E$,
   (a) if x is universal then $x \longrightarrow_R f(y_1, y_2)$,
   (b) if x is existential then $x \longrightarrow_R g_1(y_1)$ and $x \longrightarrow_R g_2(y_2)$.

Clearly A is a DTTA. It follows easily that for any vertex x,

$$apath(x, b) \quad \Leftrightarrow \quad (\exists \tau \in \mathcal{T}_\Sigma)\ x \xrightarrow{*}_R \tau, \tag{3}$$

and thus $apath(a, b)$ iff $T(A)$ is nonempty. The '$\Rightarrow$' direction follows by induction on the size of any alternating path to b and case analysis on x (universal or existential). The base case ($x = b$) is trivial. Let us consider one induction case, namely when x is existential and different from b. Then, for some vertex z,

$$\begin{aligned} apath(x, b) \quad &\Rightarrow \quad (x, z) \in E,\ apath(z, b) \\ &\overset{\text{(IH)}}{\Rightarrow} \quad x \longrightarrow_R g(z),\ z \xrightarrow{*}_R \tau \\ &\Rightarrow \quad x \xrightarrow{*}_R g(\tau), \end{aligned}$$

[3] The book of Greenlaw, Hoover and Ruzzo [24] includes an excellent up-to-date survey of around 150 P-complete problems.

where $\tau \in \mathcal{T}_\Sigma$ and g is either g_1 or g_2. The '$\Leftarrow$' direction follows also easily by induction on the length of reductions.

We prove now that the nonemptiness problem of TTAs (and thus TAs) is in P by giving a simple reduction from it to alternating graph accessibility. Let A be a TTA (Q, Σ, R, I). Assume without loss of generality that there is only one constant c in Σ and that I is a singleton set $\{q_0\}$. We construct a graph $G = (V, E)$ with designated vertices a and b and a subset U as the set of universal vertices as follows. Let $V = Q \cup U$ where U is the collection $\{ u_t \mid q \to t \in R \} \cup \{u_c\}$ of new vertices. Let $a = q_0$ and $b = u_c$. Let

$$E \quad = \quad \{ (q, u_t), (u_t, q_1), \ldots, (u_t, q_n) \mid q \to \underbrace{f(q_1, \ldots, q_n)}_{t} \in R \}.$$

Like above, statement (3) is proved for all $x \in Q$ by induction. It follows that $apath(a, b)$ iff $T(A)$ is nonempty.

Finally, we give a simple reduction from generability to the nonemptiness problem of DTAs to show that it's P-hard. Let Q be a finite set, $\boldsymbol{f}$ a binary function on Q, $V \subseteq Q$ and $q_\mathrm{f} \in Q$. Let A be the DTA $(Q, \Sigma, R, \{q_\mathrm{f}\})$, where Σ consists of a binary function symbol f and a constant c_q for each $q \in V$. Let R be the following set of rules:

$$R \quad = \quad \{ c_q \to q \mid q \in V \} \cup \{ f(q_1, q_2) \to q \mid \boldsymbol{f}(q_1, q_2) = q \}.$$

It follows easily that $T(A)$ is nonempty iff q_f is in the least subset of Q including V that is closed under $\boldsymbol{f}$. ⊠

Nonemptiness of DTAs is in fact the same problem as (the more general formulation of) generability given above. Consider a total DTA A with signature Σ as the pair $(\mathfrak{A}, F)$ where $\mathfrak{A}$ is a Σ-algebra and F a subset of its universe. Nonemptiness of $T(A)$ is simply the question of whether there exists a term $\tau \in \mathcal{T}_\Sigma$ such that $\tau^{\mathfrak{A}} \in F$, or in other words, if the subalgebra of $\mathfrak{A}$ generated by the empty set intersects with F.

The nonemptiness problem is clearly a particular case of the inequivalence problem. It is also easy to see that there is logspace reduction from any two DTAs A and B to the DTA in (2). It follows thus that inequivalence of DTAs is also P-complete. From a statement in Seidl [46, Theorem 4.3: P-completeness of inequivalence of m-ambiguous TTAs] follows that inequivalence of DTTAs is P-complete as well. For TAs in general the situation is different however. In order to reduce the inequivalence problem of two TAs into the nonemptiness problem by using (2) it is necessary to first transform the TAs in question into DTAs which in general implies an exponential increase in the number of states (this is true already in the case of NFAs [43, 36]). In fact, Seidl has proved that the inequivalence problem of TAs is EXPTIME-complete [46, Theorem 2.1]. The inequivalence problem of NFAs and regular expressions is PSPACE-complete [37]. For more recent developments regarding complexity of word problems see Mayer and Stockmeyer [35].

5 Intersection Nonemptiness of Finite Tree Automata

We proceed in two steps. First we prove that intersection nonemptiness of DTAs is EXPTIME-hard. Then we show that intersection nonemptiness of TAs is in EXPTIME.

EXPTIME-hardness of these problems has been stated before (without detailed proofs) and used in various contexts. EXPTIME-hardness of intersection nonemptiness of TAs has been remarked by Frühwirth et al [15] and used in the context of type inference of logic programs. Goubault gives an incomplete EXPTIME-hardness proof of the intersection nonemptiness problem of DTAs in the context of a faulty EXPTIME-completeness proof of simultaneous rigid E-unification [22]. (Note that this proof is faulty by the result of Degtyarev and Voronkov [10, 11, 9, 12].) Seidl [47]

uses EXPTIME-hardness of the intersection nonemptiness of DTTAs and outlines a proof in the context of sort inference in typed functional programming. The proof presented here is a generalization of the proof of PSPACE-hardness of the intersection nonemptiness of DFAs by Kozen [32]. It's general outline is the same as in the remarks or proof outlines provided in the above references.

We reduce the intersection nonemptiness problem of TAs to a wellkown problem in EXPTIME [2]. It is also remarked by Frühwirth et al that this problem is in EXPTIME [15]. It should be noted that informally this is clear already from the fact that the size of a direct product of an unbounded number of TAs is exponential and to test nonemptiness takes polynomial time in the size of that product by Theorem 1.

We state the main result of this section as the following theorem. Formally, it follows from Lemma 7 and Lemma 11 below.

Theorem 2 *Intersection nonemptiness of TAs and DTAs is EXPTIME-complete.*

Any signature can ofcourse be encoded with just one binary function symbol and a collection of constants. In particular, by examining the construction of the DTAs in the hardness part of the proof of Theorem 2 we see that the signature Σ of the DTAs consists of one binary function symbol f, one constant c and a collection of unary function symbols. For any DTA $A = (Q, \Sigma, R, F)$ let $A' = (Q', \Sigma', R', F)$ denote the following DTA. For each unary function symbol g in Σ let c_g be a new constant and q_g a new state. Let Σ' consist of f, c and those new constants, and let Q' be Q extended with those new states. Let R' be like R except that each rule $g(q) \to p$ in R is replaced with the rules $c_g \to q_g$ and $f(q_g, q) \to p$. Given DTAs A_1 and A_2 with signature Σ it follows easily that $T(A_1') \cap T(A_2')$ is nonempty iff $T(A_1) \cap T(A_2)$ is nonempty. We obtain thus the following corollary.

Corollary 3 *Intersection nonemptiness of DTAs is EXPTIME-hard even when restricted to signatures consisting of constants and one binary function symbol.*

It is wellknown that the use of intersection can shorten a regular expression by an exponential amount. So for example the inequivalence problem for regular expressions is PSPACE-complete [37], but becomes EXPSPACE-complete when intersection is added [16, 27]. (Similar effect if obtained with interleaving [35].) In case of finite automata or finite tree automata, taking their intersection corresponds to taking their direct product. In some cases the size of a finite automaton or TA, can be decreased by an exponential amount by representing it by a sequence of finite automata or TAs, resepectively. To be precise let us consider the following notion.

- Given a sequence $\mathcal{A} = (A_i)_{i \leq n}$ of TAs, let $\Pi_{\mathcal{A}}$ denote the TA $A_0 \times A_1 \times \cdots \times A_{n-1}$. The sequence $\mathcal{A}$ is called a *product* representation of any TA that is isomorfic with $\Pi_{\mathcal{A}}$.

It follows immediately from Theorem 2 and the property $T(A \times B) = T(A) \cap T(B)$ (for TAs A and B), that the *product nonemptiness* problem of finite tree automata (i.e.: Given a product representation of a TA A, is $T(A)$ is nonempty?), is EXPTIME-complete. For finite automata the product nonemptiness is PSPACE-complete by Kozens result [32]. Let us note that the usual notion of *succinct representation of a graph* is a boolean circuit which given as input binary representations of two integers (representing two nodes in that graph) computes the corresponding entry of the adjacency matrix of that graph [20]. For example, the succinct graph accessibility problem is PSPACE-complete [41] (also for undirected graphs [33]).

In general one can define product representation of an abitrary finite first order structure in the above manner, i.e., as sequence of first order structures (with the same type) denoting the corresponding direct product. It follows for example from Corollary 3 that product generability is EXPTIME-complete. Let us also note that

it is generally believed that EXPTIME is nothing else but P on exponentially more succinct input [40]

5.1 EXPTIME-hardness of Intersection Nonemptiness of DTAs

We give a polynomial time reduction of polynomial space ATMs to the intersection nonemptiness problem of DTAs. It follows that the problem is APSPACE-hard and thus EXPTIME-hard. For the rest of this section let

$$M = ((Q, \Sigma_{\text{in}}, \Sigma, \delta, q_0, \text{ƀ}, F), U)$$

be a fixed ATM that is space-bounded by some polynomial S such that $S(m) \geq m$. We can assume without loss of generality that M has a single tape, this follows from a straightforward generalization of the corresponding property for TMs [26, Theorem 12.2]. Let $x \in \Sigma_{\text{in}}^+$ be a fixed string and $n = S(|x|)$. Let ID stand for the set of all possible strings that represent IDs of M that may be padded with extra blanks at the end so that each string represents the first n tape symbols of M, i.e.,

$$ID \quad = \quad \bigcup_{0 \leq k < n} \Sigma^{(k)} Q \Sigma^{(n-k)}.$$

From here on we will by ID mean any element of ID. We can assume without loss of generality that M satisfies the following conditions:

- The initial state q_0 is existential and occurs only in the initial ID ($ID_0 = q_0 x \text{ƀ}^{(n-|x|)}$).
- M has exactly one final state q_{f} and the final ID has the form $ID_{\text{f}} = q_{\text{f}} \text{ƀ}^{(n)}$.
- Each universal ID has 0 or 2 successors.

Let all the symbols in $\Sigma \cup Q$ have arity 1,i.e., treat them like unary function symbols. Let also $\langle\rangle$ and *nil* be new function symbols with arities 2 and 0, respectively. Let $\Gamma = \Sigma \cup Q \cup \{\langle\rangle, \mathit{nil}\}$. We will represent "computations trees" of M by certain terms in $\mathcal{T}_\Gamma$. For a string $v = c_1 c_2 \cdots c_m$ over $\Sigma \cup Q$ and τ a term we write τv for the term $c_m(c_{m-1}(\cdots c_1(\tau)\cdots))$, and for any two terms τ_1 and τ_2 we write $\langle \tau_1, \tau_2 \rangle$ for the term $\langle\rangle(\tau_1, \tau_2)$.

- *ID-trees* is the least class of terms in $\mathcal{T}_\Gamma$ that satisfies:
  - *nil* is an ID-tree, called the *empty* ID-tree;
  - if τ_1 and τ_2 are ID-trees such that either both are empty or only τ_2 is empty and $v \in ID$ then $\tau = \langle \tau_1, \tau_2 \rangle v \text{ƀ}$ is an ID-tree.

  We refer to τ_1 and τ_2 as the *left* and *right subtrees* (or collectively *immediate subtrees*) of τ, v is called the *root* of τ. We will use the notations $\text{Left}(\tau)$, $\text{Right}(\tau)$ and $\text{Root}(\tau)$. We let also $\text{Root}(\mathit{nil}) = \epsilon$.

Let τ and τ' be ID-trees. We say that τ' is an *m-fold subtree* of τ if either $m = 0$ and $\tau' = \tau$ or τ' is an $(m-1)$-fold subtree of some immediate subtree of τ. By *subtree* we mean m-fold subtree for some $m \geq 0$. The *depth* of τ is the largest $m \geq 0$ such that there exists an m-fold subtree of τ, e.g., the depth of *nil* is 0.

The roots of all the nonempty subtrees of τ are called its *nodes*. A nonempty subtree of τ with empty immediate subtrees is called *external*. A nonempty subtree of τ that is not external is called *internal*. The root of any external subtree of τ is called a *leaf* of τ. We will use the following definitions.

- An *ID-triple* is any element of $ID \times ID \times (ID \cup \{\epsilon\})$, where ϵ denotes the empty string. By a *move* of M we mean any ID-triple (v, v_1, v_2) where either

– v is existential, $v \vdash v_1$ and $v_2 = \epsilon$, or

– v is universal, $v \vdash v_1$, $v \vdash v_2$ and $v_1 \neq v_2$.

We write $v \rhd (v_1, v_2)$ iff (v, v_1, v_2) is a move.

▶ A *move-tree* is any ID-tree τ such that for each internal subtree τ' of τ,

$$\mathrm{Root}(\tau') \rhd (\mathrm{Root}(\mathrm{Left}(\tau')), \mathrm{Root}(\mathrm{Right}(\tau'))).$$

A move-tree is *valid* if its root is the initial ID and its leaves are final IDs.

The notion of a valid move-tree is a straightforward generalization of the notion of a valid computation of M on input x. We will exploit the following obvious characterization of acceptance in terms of valid move-trees: *M accepts x iff there exists a valid move-tree.*

5.1.1 Main Construction The kernel of the hardness proof is a polynomial time construction of a collection of tree automata such that their intersection is precisely the set of all valid move-trees. We will construct two kinds of automata, one for each k, $1 \leq k \leq n$.

1. The first kind recognizes all move-trees the leaves of which are final IDs and which satisfy the following additional property. Roughly, for all internal m-fold subtrees τ where m is *even*, the ID-triple (v, v_1, v_2), where v is the root of τ and v_1 and v_2 the roots of the left and right subtrees of τ, is a possible move by looking only at the tape symbols immediately surrounding the k'th symbol.

2. The second kind recognizes all move-trees the root of which is the initial ID and which satisfy the same additional property as above, except for *odd* m.

First we will formally define the sets of ID-trees correspeonding to items 1 and 2, and show that their intersection gives us precisely all the valid move-trees. Then we present formal constructions of DTAs that recognize these sets. We need some additional notations and definitions.

By a *position* we mean any integer k such that $1 \leq k \leq n$. Let k be a position and $v = a_1 \cdots a_{i-1} q a_i \cdots a_n \in ID$ where $q \in Q$. We will write $v[k]$ and $\mathrm{View}(v, k)$ for the following substrings of v,

$$v[k] = \begin{cases} qa_k, & \text{if } k = i; \\ a_k, & \text{otherwise.} \end{cases}$$

$$\mathrm{View}(v, k) = \begin{cases} v[k]v[k+1], & \text{if } k = 1; \\ v[k-1]v[k], & \text{if } k = n; \\ v[k-1]v[k]v[k+1], & \text{otherwise.} \end{cases}$$

We let also $\mathrm{View}(\epsilon, k) = \epsilon$ and for any ID-triple (v, v_1, v_2),

$$\mathrm{View}((v, v_1, v_2), k) = (\mathrm{View}(v, k), \mathrm{View}(v_1, k), \mathrm{View}(v_2, k)).$$

Consider a fixed position k.

▶ A *k-move* is an ID-triple $\vec{v} = (v, v_1, v_2)$ such that the following holds.

1. If $v[k] \in Q\Sigma$ then there exist a move $\vec{w}$ such that $\mathrm{View}(\vec{w}, k) = \mathrm{View}(\vec{v}, k)$.
2. If $v[k] = a \in \Sigma$ then $v_1[k] \in \{a\} \cup Qa$ and either $v_2 = \epsilon$ or $v_2[k] \in \{a\} \cup Qa$.

We write $v \rhd_k (v_1, v_2)$ iff (v, v_1, v_2) is a k-move.

The following propositions follow easily and we leave their proofs to the reader.

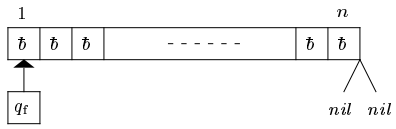

Figure 1: Base case of T_k.

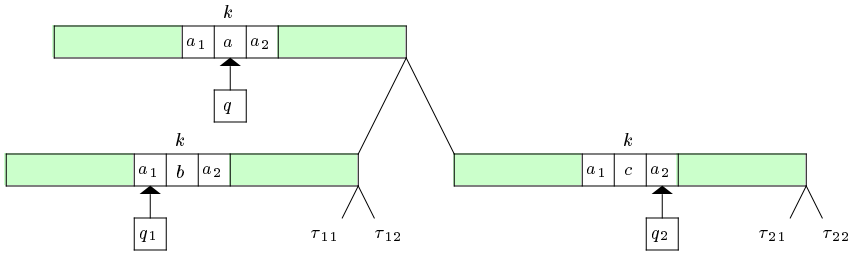

Figure 2: One possible induction case of T_k (q is universal, $\tau_{11}, \tau_{21} \in T_k$ and $\tau_{12}, \tau_{22} \in T_k \cup \{nil\}$).

Proposition 4 *An ID-triple is a move iff it is a k-move for all positions k.*

Proposition 5 *For all positions k and all ID-triples $\vec{v}$ and $\vec{w}$. If $\vec{v}$ is a k-move and* $\mathrm{View}(\vec{v}, k) = \mathrm{View}(\vec{w}, k)$ *then $\vec{w}$ is a k-move.*

For all positions k, let T_k denote the following set of terms. Below we will show that T_k is recognizable and that the time complexity to construct a tree automaton that recognizes T_k is polynomial in n.

- T_k is the set T of all *ID*-trees such that
  1. $\langle nil, nil\rangle ID_f ƀ \in T$,
  2. $\langle \tau_1, \tau_2\rangle v ƀ \in T$ if τ_1 is nonempty and,
     (a) $v \rhd_k (\mathrm{Root}(\tau_1), \mathrm{Root}(\tau_2))$,
     (b) $\mathrm{Left}(\tau_1) \in T$ and $\mathrm{Right}(\tau_1) \in T \cup \{nil\}$, and
     (c) either τ_2 is empty, or $\mathrm{Left}(\tau_2) \in T$ and $\mathrm{Right}(\tau_2) \in T \cup \{nil\}$.

So any *ID*-tree in T_k has external subtrees of the form shown in Figure 1. A possible induction case is illustrated in Figure 2. For each position k, we let T_k^* denote the following sets of terms. Also in this case we will show that each T_k^* is recognizable by a tree automaton that can be constructed in polynomial time.

- T_k^* is the set of all $\langle \tau_1, \tau_2\rangle ID_0 ƀ$ where $\tau_1, \tau_2 \in T$ and either both are empty or only τ_2 is empty, where T is the set of all *ID*-trees where q_0 doesn't occur such that
  1. $nil \in T$,
  2. $\langle \tau_1, \tau_2\rangle v ƀ \in T$ if τ_1 is nonempty and (a–c) hold,
     (a) $v \rhd_k (\mathrm{Root}(\tau_1), \mathrm{Root}(\tau_2))$,
     (b) $\mathrm{Left}(\tau_1), \mathrm{Right}(\tau_1) \in T$, and
     (c) either τ_2 is empty or $\mathrm{Left}(\tau_2), \mathrm{Right}(\tau_2) \in T$.

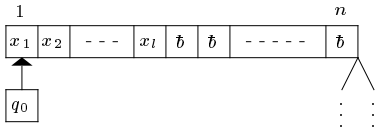

Figure 3: All *ID*-trees in T_k^* have this form.

All *ID*-trees in T_k^* are illustrated in Figure 3.

Let $\tau \in T_k \cap T_k^*$ and let τ' be any internal m-fold subtree of τ for some $m \geq 0$. If m is even (odd) then it follows by definition of T_k (T_k^*), that

$$\mathrm{Root}(\tau') \rhd_k (\mathrm{Root}(\mathrm{Left}(\tau')), \mathrm{Root}(\mathrm{Right}(\tau'))).$$

We have thus the following property.

Proposition 6 *For all positions k, if $\tau \in T_k \cap T_k^*$ then for all internal subtrees τ' of τ,*

$$\mathrm{Root}(\tau') \rhd_k (\mathrm{Root}(\mathrm{Left}(\tau')), \mathrm{Root}(\mathrm{Right}(\tau'))).$$

We can now state our main lemma.

Lemma 7 *The intersection nonemptiness problem of DTAs is EXPTIME-hard.*

Proof. Construct tree automata A_k and A_k^* for $1 \leq k \leq n$ such that $T(A_k) = T_k$ and $T(A_k^*) = T_k^*$ (see Lemma 8 and Lemma 9). Each one is constructed in time that is polynomial in $n = S(|x|)$, and thus the total time complexity of the costruction of all the automata is polynomial in $|x|$. It is sufficient to show that

$$\{\, \tau \mid \tau \text{ is a valid move-tree} \,\} = \bigcap_{k=1}^{n} (T_k \cap T_k^*)$$

The direction '$\subseteq$' (i.e., that each valid move tree is in T_k and T_k^*) is easy to check. (Note that the property that all computation paths of M have even length is needed here.) We prove the direction '$\supseteq$'. Let $\tau \in \bigcap_{k=1}^{n} (T_k \cap T_k^*)$. It follows immediately from the definition of any T_k that the leaves of τ are final IDs. It follows also immediately from the definition of any T_k^* that the root of τ is the initial ID. It remains to prove that τ is a move-tree, i.e., that for any internal subtree τ' of τ,

$$\mathrm{Root}(\tau') \rhd (\mathrm{Root}(\mathrm{Left}(\tau')), \mathrm{Root}(\mathrm{Right}(\tau'))),$$

but this follows by first applying Proposition 6 and then Proposition 4. ⊠

5.1.2 Recognizability of T_k Consider a fixed position k distinct from 1 and n. The handling of positions 1 and n is similar. We will construct a tree automaton A_k that recognizes T_k. It will be clear that one can easily extract an algorithm from this construction that has polynomial time complexity in n. Let

$$\begin{aligned} \Delta &= \Sigma\Sigma\Sigma \cup Q\Sigma\Sigma\Sigma \cup \Sigma Q\Sigma\Sigma \cup \Sigma\Sigma Q\Sigma, \\ I &= \Delta \times (\Delta \cup \{\epsilon\}). \end{aligned}$$

As the main part in the construction of A_k we will use a family $\{M_i\}_{i \in I \cup \{0\}}$ of DFAs, where each M_i is a DFA that accepts *ID* and for each $v \in ID$ simply scans v and accepts it in the final state $p_{(\alpha,i)}$ iff $\mathrm{View}(v,k) = \alpha$. Formally, we let, for all $i \in I \cup \{0\}$,

$$M_i = (P_i, \Sigma \cup Q, \delta_i, p_{(0,i)}, \{\, p_{(\alpha,i)} \mid \alpha \in \Delta \,\}), \quad L(M_i) = ID,$$

such that for all $\alpha \in \Delta$ and $v \in ID$,

$$\delta_i(p_{(0,i)}, v) = p_{(\alpha,i)} \quad \Leftrightarrow \quad \mathrm{View}(v,k) = \alpha.$$

Furthermore, all the P_i's are assumed to be pairwise disjoint. In particular we can take all the members to be copies of say M_0. It is easy to construct M_0 in time that is polynomial in n. Let also M_f be a DFA (with new states) such that

$$M_\mathrm{f} = (P_\mathrm{f}, \Sigma \cup Q, \delta_\mathrm{f}, p_{(0,\mathrm{f})}, \{p_\mathrm{f}\}), \quad L(M_\mathrm{f}) = \{ID_\mathrm{f}\}.$$

Let now R_i for $i \in I \cup \{0,\mathrm{f}\}$ denote following sets of rules:

$$R_i \quad = \quad \{\, c(p) \to p' \mid \delta_i(c,p) = p',\ c \in \Sigma \cup Q,\ p, p' \in P_i \,\}.$$

For any string $v = c_1 c_2 \ldots c_{m-1} c_m$ over $\Sigma \cup Q$ and state p we will write pv for the term $c_m(c_{m-1}(\cdots c_2(c_1(p))\cdots))$. It is clear that for any string v over $\Sigma \cup Q$, and any two states p and p' in P_i,

$$\delta_i(p,v) = p' \quad \Leftrightarrow \quad pv \xrightarrow{*}_{R_i} p'.$$

Let $\{t_\epsilon, t_\mathrm{f}\} \cup \{\, t_\alpha \mid \alpha \in \Delta \,\}$ be a set of new state symbols.

- A_k is the following tree automaton:

$$\begin{aligned}
Q^{A_k} &= \{t_\epsilon, t_\mathrm{f}\} \cup \{\, t_\alpha \mid \alpha \in \Delta \,\} \cup P_0 \cup P_\mathrm{f} \cup \bigcup_{i \in I} P_i, \\
\Sigma^{A_k} &= \Gamma, \\
R^{A_k} &= \bigcup_{i \in I} R_i \cup R_0 \cup R_\mathrm{f} \cup \\
&\quad \{\, nil \to t_\epsilon \,\} \cup \\
&\quad \{\, \langle t_\epsilon, t_\epsilon \rangle \to p_{(0,\mathrm{f})} \,\} \cup \\
&\quad \{\, \langle t_\mathrm{f}, t_\epsilon \rangle \to p_{(0,0)},\ \langle t_\mathrm{f}, t_\mathrm{f} \rangle \to p_{(0,0)} \,\} \cup \\
&\quad \{\, \langle t_\beta, t_\gamma \rangle \to p_{(0,\beta,\gamma)} \mid (\beta,\gamma) \in I \,\} \cup \\
&\quad \{\, p_{(\alpha,0)} \natural \to t_\alpha \mid \alpha \in \Delta \,\} \cup \\
&\quad \{\, p_{\mathrm{View}((v,v_1,v_2),k)} \natural \to t_\mathrm{f} \mid v \rhd_k (v_1, v_2) \,\} \cup \\
&\quad \{\, p_\mathrm{f} \natural \to t_\mathrm{f} \,\}, \\
F^{A_k} &= \{\, t_\mathrm{f} \,\}.
\end{aligned}$$

Note that $p_{\mathrm{View}((v,v_1,v_2),k)}$ is the final state $p_{(\mathrm{View}(v,k),i)}$ in M_i, where i is the index $(\mathrm{View}(v_1,k), \mathrm{View}(v_2,k))$. It is easy to check that A_k is indeed a deterministic tree automaton. The structure of A_k is illustrated in Figure 4.

Lemma 8 $T(A_k) = T_k$.

Proof.

[**Proof of** $T(A_k) \subseteq T_k$] Let $\tau \in T(A_k)$, i.e., $\tau \in \mathcal{T}_\Gamma$ and $\tau \xrightarrow{*}_{R^{A_k}} t_\mathrm{f}$. We prove that $\tau \in T_k$. The proof is by induction on the length of the reduction $\tau \xrightarrow{*} t_\mathrm{f}$. There are two cases, depending on the last step of the reduction.

1. $\tau \xrightarrow{*} p_\mathrm{f} \natural \longrightarrow t_\mathrm{f}$, or
2. $\tau \xrightarrow{*} p_{\mathrm{View}(\vec{v},k)} \natural \longrightarrow t_\mathrm{f}$ for some k-move $\vec{v}$. Let $\mathrm{View}(\vec{v},k) = (\alpha,\beta,\gamma)$.

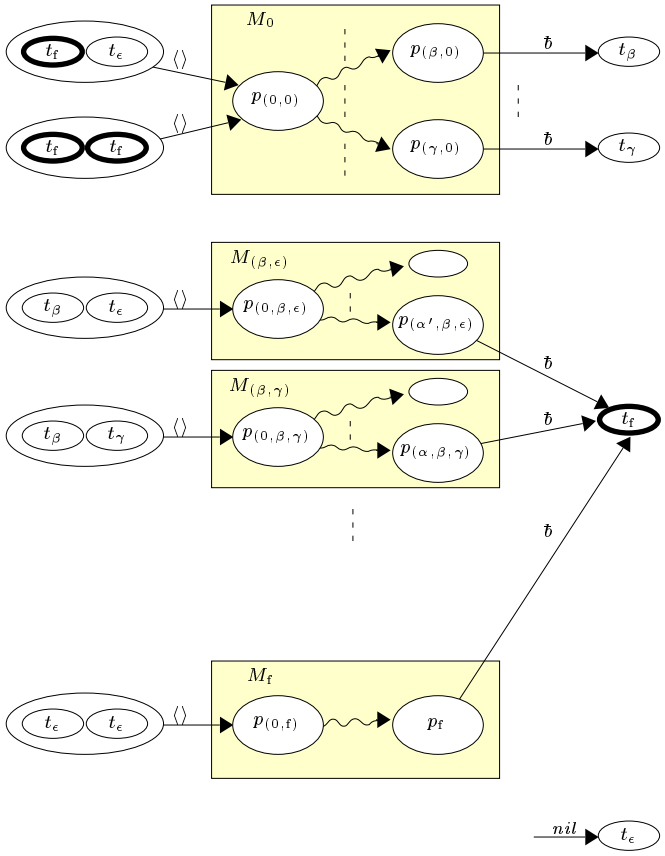

Figure 4: Tree automaton A_k. A transition from $p_{(\alpha,\beta,\gamma)}$ to t_f exists only if $(\alpha,\beta,\gamma) = \mathrm{View}(\vec{v},k)$ for some k-move $\vec{v}$.

Let us consider the first case first. From the definition of the rules of A^k and the disjointness of the underlying DFAs it follows that the reduction has to be of the following form: (simply trace the arrows backwards in Figure 4)

$$\begin{array}{rcl} p_\mathrm{f}\flat & \longrightarrow & t_\mathrm{f} \\ p_{(0,\mathrm{f})}ID_\mathrm{f} & \stackrel{*}{\longrightarrow}_{R_\mathrm{f}} & p_\mathrm{f} \\ \langle nil, nil\rangle & \stackrel{*}{\longrightarrow} & p_{(0,\mathrm{f})}, \end{array}$$

which shows that $\tau = \langle nil, nil\rangle ID_\mathrm{f}\flat$, and thus $\tau \in T_k$. We now consider the second case. Then

$$\begin{array}{rcll} p_{(\alpha,\beta,\gamma)}\flat & \longrightarrow & t_\mathrm{f} & \\ p_{(0,\beta,\gamma)}w & \stackrel{*}{\longrightarrow}_{R_{(\beta,\gamma)}} & p_{(\alpha,\beta,\gamma)} & (\text{some } w \in ID \text{ such that } \mathrm{View}(w,k) = \alpha) \\ \langle t_\beta, t_\gamma\rangle & \longrightarrow & p_{(0,\beta,\gamma)}. & \end{array}$$

So $\tau = \langle \tau_1, \tau_2\rangle w\flat$ where $\tau_1 \stackrel{*}{\longrightarrow} t_\beta$ and $\tau_2 \stackrel{*}{\longrightarrow} t_\gamma$. Since $\beta \neq \epsilon$ it follows that the reduction $\tau_1 \stackrel{*}{\longrightarrow} t_\beta$ must have the following form:

$$\begin{array}{rcll} p_{(\beta,0)}\flat & \longrightarrow & t_\beta & \\ p_{(0,0)}w_1 & \stackrel{*}{\longrightarrow}_{R_0} & p_{(\beta,0)} & (\text{some } w_1 \in ID \text{ such that } \mathrm{View}(w_1,k) = \beta), \end{array}$$

and either $\langle t_{\rm f}, t_\epsilon\rangle \longrightarrow p_{(0,0)}$ or $\langle t_{\rm f}, t_{\rm f}\rangle \longrightarrow p_{(0,0)}$. Assume (without loss of generality) that the former reduction step took place and that $\gamma = \epsilon$ (and thus $\tau_2 = nil$).

Under these conditions $\mathrm{Root}(\tau_1) = w_1$, $\mathrm{Left}(\tau_1) \stackrel{*}{\longrightarrow} t_{\rm f}$ and $\mathrm{Right}(\tau_1) = nil$. It follows by the induction hypothesis that $\mathrm{Left}(\tau_1) \in T_k$. Let $\vec{w} = (w, w_1, \epsilon)$, since $\mathrm{View}(\vec{w}, k) = \mathrm{View}(\vec{v}, k)$ and $\vec{v}$ is a k-move, it follows by Proposition 5 that $\vec{w}$ is a k-move. Now $\tau \in T_k$ by the definition of T_k.

[**Proof of** $T_k \subseteq T(A_k)$] Let $\tau \in T_k$. Clearly $\tau \in \mathcal{T}_\Gamma$. We must show that $\tau \stackrel{*}{\longrightarrow} t_{\rm f}$. The proof is by induction on the size of τ. The base case is $\tau = \langle nil, nil\rangle ID_{\rm f}\natural$ and it follows by above that $\tau \stackrel{*}{\longrightarrow} t_{\rm f}$. The induction case is $\tau = \langle \tau_1, \tau_2\rangle v\natural$, where $\tau_1 = \langle \tau_{11}, \tau_{12}\rangle v_1\natural$,

1. $v \rhd_k (v_1, \mathrm{Root}(\tau_2))$,
2. $\tau_{11} \in T_k$ and $\tau_{12} \in T_k \cup \{nil\}$, and
3. either τ_2 is empty or $\mathrm{Left}(\tau_2) \in T_k$ and $\mathrm{Right}(\tau_2) \in T_k \cup \{nil\}$.

We can assume without loss of generality that τ_2 and τ_{12} are empty. Let $(\alpha, \beta, \epsilon) = \mathrm{View}((v, v_1, \epsilon), k)$. By using the induction hypothesis and the rules of A_k we obtain the following reduction:

$$\begin{array}{rcl}
\tau & \stackrel{*}{\longrightarrow}_{(\mathrm{IH})} & \langle\langle t_{\rm f}, t_\epsilon\rangle v_1\natural, t_\epsilon\rangle v\natural \\
& \longrightarrow & \langle p_{(0,0)} v_1\natural, t_\epsilon\rangle v\natural \\
& \stackrel{*}{\longrightarrow}_{R_0} & \langle p_{(\beta,0)}\natural, t_\epsilon\rangle v\natural \\
& \longrightarrow & \langle t_\beta, t_\epsilon\rangle v\natural \\
& \longrightarrow & p_{(0,\beta,\epsilon)} v\natural \\
& \stackrel{*}{\longrightarrow}_{R_{(\beta,\epsilon)}} & p_{(\alpha,\beta,\epsilon)}\natural.
\end{array}$$

But $v \rhd_k (v_1, \epsilon)$, and thus $p_{(\alpha,\beta,\epsilon)}\natural \longrightarrow t_{\rm f}$. ⊠

5.1.3 Recognizability of T_k^* Like above, we consider a fixed position k distinct from 1 and n, and construct a tree automaton A_k^* that recognizes T_k^*. It will be clear that the construction has polynomial time complexity in n. We will not be as detailed as we were in the previous section due to the similarity of the construction.

Let Δ and I be as in Section 5.1.2 except that the initial state q_0 of M is omitted from Q. Let also M_i for $i \in I \cup \{0\}$ have the same definition (except for that same restriction). Let $M_{\rm f}$ be the following DFA: (with new states)

$$M_{\rm f} = (P_{\rm f}, \Sigma, \delta_{\rm f}, p_{(0,{\rm f})}, \{p_{\rm f}\}), \quad L(M_{\rm f}) = \{x\natural^{(n-|x|)}\}.$$

Let now R_i for $i \in I \cup \{0, {\rm f}\}$ denote the same sets of rules as defined above. Let $\{t, t_\epsilon, t_{\rm f}\} \cup \{\, t_\alpha \mid \alpha \in \Delta \,\}$ be a set of new state symbols.

- A_k^* is the following tree automaton:

$$\begin{array}{rcl}
Q^{A_k^*} & = & \{t, t_\epsilon, t_{\rm f}\} \cup \{\, t_\alpha \mid \alpha \in \Delta \,\} \cup P_0 \cup P_{\rm f} \cup \bigcup_{i\in I} P_i, \\
\Sigma^{A_k^*} & = & \Gamma, \\
R^{A_k^*} & = & \bigcup_{i\in I} R_i \cup R_0 \cup R_{\rm f} \cup \{q_0(p_{(0,0)}) \to p_{(0,{\rm f})}\} \cup \\
& & \{\, nil \to t_\epsilon \,\} \cup \\
& & \{\, \langle t_\epsilon, t_\epsilon\rangle \to p_{(0,0)},\ \langle t, t_\epsilon\rangle \to p_{(0,0)},\ \langle t, t\rangle \to p_{(0,0)} \,\} \cup \\
& & \{\, \langle t_\beta, t_\gamma\rangle \to p_{(0,\beta,\gamma)} \mid (\beta, \gamma) \in I \,\} \cup \\
& & \{\, p_{(\alpha,0)}\natural \to t_\alpha \mid \alpha \in \Delta \,\} \cup
\end{array}$$

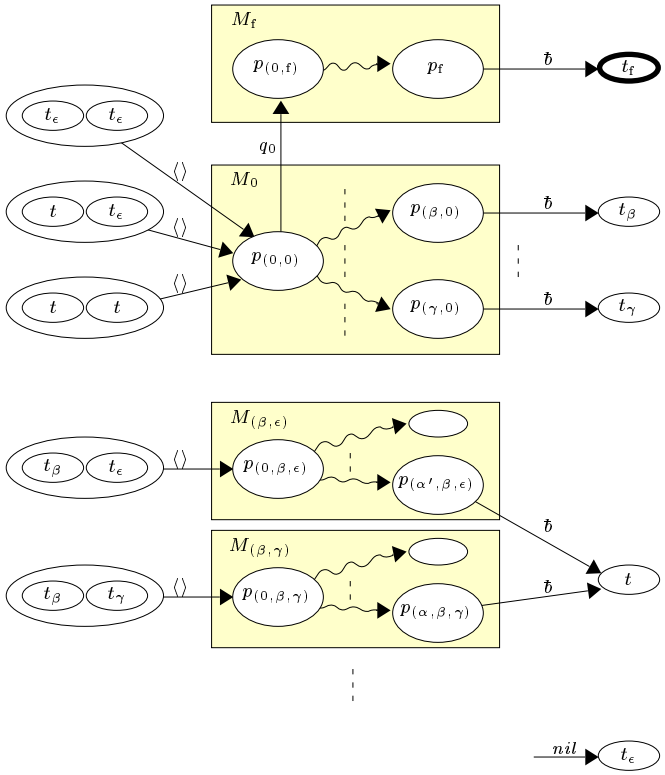

Figure 5: Tree automaton A_k^*. A transition from $p_{(\alpha,\beta,\gamma)}$ to t exists only if $(\alpha,\beta,\gamma) = \mathrm{View}(\vec{v},k)$ for some k-move $\vec{v}$.

$$
\begin{array}{rcl}
 & & \{\, p_{\mathrm{View}((v,v_1,v_2),k)}\hbar \to t \mid v \rhd_k (v_1,v_2) \,\} \cup \\
 & & \{\, p_{\mathrm{f}}\hbar \to t_{\mathrm{f}} \,\}, \\
F^{A_k^*} & = & \{\, t_{\mathrm{f}} \,\}.
\end{array}
$$

Note that A_k^* is indeed a deterministic tree automaton (in particular note that the q_0-transition from $p_{(0,0)}$ to $p_{(0,\mathrm{f})}$ doesn't violate the determinism). The structure of A_k^* is illustrated in Figure 5. The proof of Lemma 9 i analogous to the proof of Lemma 8.

Lemma 9 $T(A_k^*) = T_k^*$.

5.2 The Intersection Nonemptiness Problem of TAs is in EXPTIME

We reduce the intersection nonemptiness problem of TAs to the *inference problem for full implicational dependencies* or FIDs. An FID is just a universal relational Horn sentence, we write it as an implication $\varphi \leftarrow \psi$ where φ is an atom and ψ a conjunction of atoms. The only function symbols in an FID are constants. The inference problem is simply the question of whether a given conjunction of FIDs implies another given FID. This problem can be solved in exponential time (actually it is EXPTIME-complete [2, 52]).

Let A_i for $1 \leq i \leq n$ for some $n \geq 1$ be TAs with a common input alphabet Σ,

$$A_i = (Q_i, \Sigma, R_i, F_i), \quad (1 \leq i \leq n).$$

Let $A = (Q, \Sigma, R, F)$ be the direct product of all the A_i's. So the states of A are elements of $\prod_{i=1}^{n} Q_i$ and the rules of A are defined as follows, we write $\bar{q}$ for $(q_1, q_2, \ldots, q_n) \in Q$:

$$R \quad = \quad \{\, \sigma(\bar{q}_1, \ldots, \bar{q}_k) \to \bar{q} \mid \sigma(q_{i1}, q_{i2}, \ldots, q_{ik}) \longrightarrow_{R_i} q_i \quad (1 \leq i \leq n) \,\}.$$

We know that $T(A)$ is nonempty iff $\bigcap_{i=1}^{n} T(A_i)$ is nonempty. We will construct a set of FIDs P with a distinguised atom $\texttt{Nonempty}$ such that $P \vdash \texttt{Nonempty}$ iff $T(A)$ is nonempty. Furthermore, it will be obvious that this construction takes polynomial time (actually linear time) in the total size of the A_i's (*not* in the size of A, the size of A is in general eponential in the total size of the A_i's).

First, for $1 \leq i \leq n$ and each k-ary function symbol $\sigma \in \Sigma$, let $\texttt{Rule}_i^\sigma$ be a new relation symbol of arity $k+1$. Let also $\texttt{Final}_i$ for $1 \leq i \leq n$ and $\texttt{Reduce}$ be relation symbols of arity 1 and n, respectively. To simplify matters, we can assume without loss of generality that all function symbols in Σ have arity at most 2. There are following atoms (or atomic FIDs) in P: for each A_i and final state q in it there is an atom

$$\texttt{Final}_i(q)$$

in P; for each A_i and rule $\sigma(q_1, \ldots, q_k) \to q$ (where $k \geq 0$) in R_i there is an atom

$$\texttt{Rule}_i^\sigma(q_1, \ldots, q_k, q)$$

in P. In addition, P includes the following FIDs: for each constant $\sigma \in \Sigma$ the FID

$$\texttt{Reduce}(\bar{x}) \leftarrow \bigwedge_{i=1}^{n} \texttt{Rule}_i^\sigma(x_i),$$

for each unary function symbol $\sigma \in \Sigma$ the FID

$$\texttt{Reduce}(\bar{x}) \leftarrow \bigwedge_{i=1}^{n} \texttt{Rule}_i^\sigma(y_i, x_i) \wedge \texttt{Reduce}(\bar{y})$$

and for each binary function symbol $\sigma \in \Sigma$ the FID

$$\texttt{Reduce}(\bar{x}) \leftarrow \bigwedge_{i=1}^{n} \texttt{Rule}_i^\sigma(y_i, z_i, x_i) \wedge \texttt{Reduce}(\bar{y}) \wedge \texttt{Reduce}(\bar{z}).$$

Finally, P includes the FID

$$\texttt{Nonempty} \leftarrow \texttt{Reduce}(\bar{x}) \wedge \bigwedge_{i=1}^{n} \texttt{Final}_i(x_i).$$

We have the following relationship between derivations from P and reduction in R.

Proposition 10 *For all $\bar{q} \in Q$, $P \vdash \texttt{Reduce}(\bar{q})$ iff there exists a term $\tau \in \mathcal{T}_\Sigma$ such that $\tau \xrightarrow{*}_R \bar{q}$.*

Proof. Let $\bar{q} \in Q$ be fixed and consider the direction '$\Rightarrow$'. Assume that $P \vdash \texttt{Reduce}(\bar{q})$. We prove by induction on the length of the proof of $P \vdash \texttt{Reduce}(\bar{q})$ that there exists a term $\tau \in \mathcal{T}_\Sigma$ such that $\tau \xrightarrow{*}_R \bar{q}$.

The base case is when there is a constant $c \in \Sigma$ such that $P \vdash \texttt{Rule}_i^c(q_i)$ for $1 \leq i \leq n$. Then $c \longrightarrow_{R_i} q_i$ for $1 \leq i \leq n$ and thus $\tau = c \longrightarrow_R \bar{q}$. The induction case is when there is a nonconstant function symbol $f \in \Sigma$ (we can assume that f is binary) and states $\bar{p}, \bar{r} \in Q$ such that

$$P \vdash \texttt{Rule}_i^f(p_i, r_i, q_i) \text{ (for } 1 \leq i \leq n), \quad P \vdash \texttt{Reduce}(\bar{p}), \quad P \vdash \texttt{Reduce}(\bar{r}).$$

By the induction hypothesis follows that there exist terms τ_1 and τ_2 in $\mathcal{T}_\Sigma$ such that $\tau_1 \longrightarrow_R \bar{p}$ and $\tau_2 \longrightarrow_R \bar{r}$. From $P \vdash \mathtt{Rule}_i^f(p_i, r_i, q_i)$ (for $1 \leq i \leq n$) follows that $f(p_i, r_i) \longrightarrow_{R_i} q_i$ (for $1 \leq i \leq n$) and thus $f(\bar{p}, \bar{r}) \longrightarrow_R \bar{q}$. Consequently $\tau = f(\tau_1, \tau_2) \xrightarrow{*}_R \bar{q}$.

The direction '$\Leftarrow$' is equally straightforward to prove by induction on the length of the reduction $\tau \xrightarrow{*}_R \bar{q}$. ⊠

Since $P \vdash \mathtt{Nonempty}$ iff there exists a final state $\bar{q}$ in A such that $P \vdash \mathtt{Reduce}(\bar{q})$, it follows by Proposition 10 that $P \vdash \mathtt{Nonempty}$ iff $T(A)$ is nonempty. The construction of P is clearly linear in the total size of the A_i's. By Chandra et al [2] it follows thus that:

Lemma 11 *The intersection nonemptiness problem of DTAs is in EXPTIME.*

We obtain an alternative proof of Lemma 11 by looking at P as a *logic program* and asking the question if the *goal* `Nonempty` follows from it. It is clear that in any proof tree of `Nonempty` from P the nodes (or intermediate goals) have a size that is linear in n, simply because there are no nonconstant function symbols in P. The computational complexity of the problem of deciding if $P \vdash \mathtt{Nonempty}$ is therefore in EXPTIME by a correspondence between logic programs and ATMs by Shapiro [48, Theorem 4.4] and the relationship EXPTIME = APSPACE.

We can also note that NFAs correspond to *monadic* TAs, i.e., TAs over a signature where there are besides constants only unary function symbols. If we assume the above A_i's to be modadic then the nonemptiness problem of $T(A)$ corresponds to the nonemptiness problem of the intersection of the corresponding NFAs. It is easy to see by looking at P that one can construct an ATM without universal nodes (i.e., a TM) that uses only linear space in n and "accepts `Nonempty`" iff $P \vdash \mathtt{Nonempty}$. Thus the intersection nonemptiness problem of NFAs is in PSPACE. This fact follows already from the proof of the PSPACE-completeness of the intersection nonemptiness problem of DFAs by Kozen [32], where the part of the proof regarding inclusion in PSPACE holds also for NFAs.

6 Conclusions

In this report we considered computational complexity of some basic decision problems of finite tree automata. In particular, we proved EXPTIME-completeness of the intersection nonemptiness problem (Theorem 2) and we showed P-completeness of the nonemptiness problem (Theorem 1). It follows that for a fixed number of finite tree automata, the problem of nonemptiness of their intersection is also P-complete. We discussed a notion of succinctness with respect to which the intersection nonemptiness problem is in fact a succinct version of the nonemptiness problem.

Our main motivation for studying these problems and their computational complexity is their close connection with the decidability and computational complexity of certain fragments of intuitionistic logic with equality and subcases of a certain problem called simultaneous rigid E-unification that arises in the automated theorem proving context [19]. These connections are investigated in a separate joint paper by Degtyarev, Gurevich, Narendran, Veanes and Voronkov [7]. Until SREU was proved undecidable by Degtyarev and Voronkov [9, 10, 11, 12] there appeared many faulty proofs of its decidability [17, 18, 22]. See the survey paper by Degtyarev, Gurevich and Voronkov [8] for the impact of this undecidability result on the automated theorem proving community. Further implications are studied by Veanes [53], and Gurevich and Veanes [25].

The computational complexities of the problems studied in this report and of closely related problems is summarized in Table 1. In general there seems to be a rule of thumb that says that *if a decision problem for (deterministic) finite automata is*

	Nonemptiness	Inequivalence	Intersection nonemptiness
DFA	NL	NL	PSPACE
NFA	NL	PSPACE	PSPACE
DTA	P	P	EXPTIME
DTTA	P	P	EXPTIME
TA	P	EXPTIME	EXPTIME

Table 1: Computational complexities of some basic decision problems of finite automata and finite tree automata. All problems are complete for the respective classes.

complete for a certain space complexity then the same decision problem with (deterministic) finite tree *automata is complete for the corresponding deterministic time complexity, only one exponential higher.* Besides Table 1, further justification for this rule follows by comparing computational complexities of some other decision problems of finite tree automata studied by Seidl [46] with the corresponding decision problems of finite automata studied by Stearns and Hunt III [49, 50]. This relationship between computational complexities of decision problems of finite tree automata and finite automata is reflected by the fact that proofs of the former are usually extensions of proofs of the latter, by going from using nondeterministic Turing machines to using alternating Turing machines.

Remarks about Table 1 The nonemptiness problem of finite automata is in fact the graph accessibility problem and is thus complete for nondeterministic logarithmic space or NL-complete [45]. Using (2), inequivalence of DFAs reduces to nonemptiness [39] and since nonemptiness is a particular case of inequivalence, it follows that inequivalnece of DFAs is NL-complete as well. For finite automata in general, inequivalence is PSPACE-complete by Meyer and Stockmeyer [37]. PSPACE-completeness of nonemptiness of intersection of finite automata was proved by Kozen [32].

Nonemptiness of finite tree automata is closely related to the two wellknown P-complete problems: alternating graph accessibility [28] and generability [30, 31]. It follows by (2) that inequivalence of DTAs is also P-complete. EXPTIME-hardness of the intersection nonemptiness problem of finite tree automata has been observed by other researchers [15, 22, 47]. In particular, Seidl outlines a proof in the case of DTTAs [47]. He has also proved that inequivalence of TAs is EXPTIME-complete [46, Theorem 2.1] and it follows also from a statement by Seidl that when restricted to DTTAs, inequivalnece is P-complete [46, Theorem 4.3].

Acknowledgements

The author wishes to thank Evgeny Dantsin and Andrei Voronkov for valuable comments and discussions. In particular, it was pointed out by Dantsin that the nonemptiness problem of finite tree automata is a generalization of the graph reachability problem and can be decided in polynomial time, and the reduction of the intersection nonemptiness problem of finite tree automata to the inference problem for full implicational dependencies was suggested by Voronkov.

References

[1] J.R. Büchi and J.B. Wright. Mathematical theory of automata. course notes. Communications Sciences 403, University of Michigan, 1960.

[2] A. Chandra, H. Lewis, and J. Makowsky. Embedded implicational dependencies and their inference problem. In *Proc. of 13th Annual ACM Symposium on Theory of Computing (STOC)*, pages 342–354, 1981.

[3] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133, January 1981.

[4] J.L. Conquidé, M. Dauchet, R. Gilleron, and S. Vágvölgyi. Bottom-up tree push-down automata: classification and connection with rewrite systems. *Theoretical Computer Science*, 127:69–98, 1994.

[5] B. Courcelle. On recognizable sets and tree automata. In M. Nivat and H. Ait-Kaci, editors, *Resolution of Equations in Algebraic Structures*. Academic Press, 1989.

[6] M. Dauchet. Rewriting and tree automata. In H. Comon and J.P. Jouannaud, editors, *Term Rewriting (French Spring School of Theoretical Computer Science)*, volume 909 of *Lecture Notes in Computer Science*, pages 95–113. Springer Verlag, Font Romeux, France, 1993.

[7] A. Degtyarev, Yu. Gurevich, P. Narendran, M. Veanes, and A. Voronkov. The decidability of simultaneous rigid E-unification with one variable. Upmail technical report, Uppsala University, Computing Science Department, 1997. To appear.

[8] A. Degtyarev, Yu. Gurevich, and A. Voronkov. Herbrand's theorem and equational reasoning: Problems and solutions. In *Bulletin of the European Association for Theoretical Computer Science*, volume 57. October 1996. The "Logic in Computer Science" column.

[9] A. Degtyarev, Yu. Matiyasevich, and A. Voronkov. Simultaneous rigid E-unification and related algorithmic problems. In *Eleventh Annual IEEE Symposium on Logic in Computer Science (LICS'96)*, pages 494–502, New Brunswick, NJ, July 1996. IEEE Computer Society Press.

[10] A. Degtyarev and A. Voronkov. Simultaneous rigid E-unification is undecidable. UPMAIL Technical Report 105, Uppsala University, Computing Science Department, May 1995.

[11] A. Degtyarev and A. Voronkov. Simultaneous rigid E-unification is undecidable. In H. Kleine Büning, editor, *Computer Science Logic. 9th International Workshop, CSL'95*, volume 1092 of *Lecture Notes in Computer Science*, pages 178–190, Paderborn, Germany, September 1995, 1996.

[12] A. Degtyarev and A. Voronkov. The undecidability of simultaneous rigid E-unification. *Theoretical Computer Science*, 166:10, 1996.

[13] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Methods and Semantics, chapter 6, pages 243–309. North Holland, Amsterdam, 1990.

[14] J. Doner. Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, 4:406–451, 1970.

[15] T. Frühwirth, E. Shapiro, M. Vardi, and E. Yardeni. Logic programs as types of logic programs. In *Proc. 6th Symposium on Logics in Computer Science (LICS)*, pages 300–309, 1991.

[16] M. Fürer. The complexity of the inequivalence problem for regular expressions with intersection. In *Proc. 7th International Colloquium on Autamata, Languages, and Programming (ICALP)*, volume 85 of *Lecture Notes in Computer Science*, pages 234–245, New York, 1980. Springer Verlag.

[17] J. Gallier, P. Narendran, S. Raatz, and W. Snyder. Theorem proving using equational matings and rigid E-unification. *Journal of the Association for Computing Machinery*, 39(2):377–429, 1992.

[18] J.H. Gallier, P. Narendran, D. Plaisted, and W. Snyder. Rigid E-unification is NP-complete. In *Proc. IEEE Conference on Logic in Computer Science (LICS)*, pages 338–346. IEEE Computer Society Press, July 1988.

[19] J.H. Gallier, S. Raatz, and W. Snyder. Theorem proving using rigid E-unification: Equational matings. In *Proc. IEEE Conference on Logic in Computer Science (LICS)*, pages 338–346. IEEE Computer Society Press, 1987.

[20] H. Galperin and A. Wigderson. Succinct representations of graphs. *Information and Control*, 56:183–198, 1983.

[21] F. Gécseg and M. Steinby. *Tree Automata*. Akadémiai Kiodó, Budapest, 1984.

[22] J. Goubault. Rigid $\vec{E}$-unifiability is DEXPTIME-complete. In *Proc. IEEE Conference on Logic in Computer Science (LICS)*. IEEE Computer Society Press, 1994.

[23] R. Greenlaw, H.J. Hoover, and W.L. Ruzzo. A compendium of problems complete for P. Technical Report TR 91–05–01, Department of Computer Science and Engineering, University of Washington, 1991.

[24] R. Greenlaw, H.J. Hoover, and W.L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press, 1995.

[25] Y. Gurevich and M. Veanes. On the Herbrand skeleton problem. Technical report, Uppsala University, Computing Science Department, 1997. In preparation.

[26] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley Publishing Co., 1979.

[27] H.B. Hunt III. The equivalence problem for regular expressions with intersection is not polynomial in tape. Technical Report TR 73-161, Cornell University, Ithaca, NY., 1973.

[28] N. Immerman. Number of quantifiers is better than number of tape cells. *Journal of Computer and System Sciences*, 22(3):384–406, 1981.

[29] D.S. Johnson. A catalog of complexity classes. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A: Algorithms and Complexity, chapter 2, pages 67–161. Elsevier Science, Amsterdam, 1990.

[30] N.D. Jones and W.T. Laaser. Complete problems for deterministic polynomial time. *Theoretical Computer Science*, 3(2):105–117, 1976.

[31] D. Kozen. Complexity of finitely presented algebras. In *Proc. of the 9th Annual Symposium on Theory of Computing*, pages 164–177, New York, 1977. ACM.

[32] D. Kozen. Lower bounds for natural proof systems. In *Proc. 18th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 254–266, 1977.

[33] A. Lozano and J.L. Balcázar. The complexity of graph problems for succinctly represented graphs. In M. Nagl, editor, *Graph-Theoretic Concepts in Computer Science, 15th International Workshop WG'89*, volume 411 of *Lecture Notes in Computer Science*, pages 277–286. Springer Verlag, 1989.

[34] M. Magidor and G. Moran. Finite automata over finite trees. Technical Report 30, Hebrew University, Jerusalem, 1969.

[35] A.J. Mayer and L.J. Stockmeyer. The complexity of word problems – this time with interleaving. *Information and Computation*, 115:293–311, 1994.

[36] A.R. Meyer and M.J. Fisher. Economy of description of automata, grammars and formal systems. In *Proc. 12th IEEE Symposium on Switching and Automata Theory (SWAT)*, pages 188–191, 1971.

[37] A.R. Meyer and L.J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Proc. 13th IEEE Symposium on Switching and Automata Theory (SWAT)*, pages 125–129, 1972.

[38] J. Mezei and J.B. Wright. Algebraic automata and context-free sets. *Information and Control*, 11:3–29, 1967.

[39] E.F. Moore. Gedanken experiments on sequential machines. In *Automata Studies*, pages 129–153. Princeton Univ. Press, Princeton, N.J., 1956.

[40] C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[41] C.H. Papadimitriou and M. Yannakakis. A note on succinct representations of graphs. *Information and Control*, 71:181–185, 1986.

[42] M.O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, 141:1–35, 1969.

[43] M.O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:115–125, 1959.

[44] K. Salomaa, D. Wood, and S. Yu. Complexity of $E0L$ structural equivalence. In *Mathematical Foundations of Computer Science 1994*, number 841 in Lecture Notes in Computer Science, pages 587–596, Košice, Slovakia, 1994. Springer Verlag.

[45] W.J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[46] H. Seidl. Deciding equivalence of finite tree automata. *SIAM Journal of Computing*, 19(3):424–437, 1990.

[47] H. Seidl. Haskell overloading is DEXPTIME-complete. *Information Processing Letters*, 52(2):57–60, 1994.

[48] E.Y. Shapiro. Alternation and the computational complexity of logic programs. *Journal of Logic Programming*, 1:19–33, 1984.

[49] R. Stearns and H. Hunt III. On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. In *Proc. 22th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 74–81, 1981.

[50] R. Stearns and H. Hunt III. On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. *SIAM Journal of Computing*, 14:598–611, 1985.

[51] J.W. Thatcher and J.B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

[52] M. Y. Vardi. The complexity of relational query languages. In *Proc. 14th ACM Symposium on Theory of Computing (STOC)*, pages 137–146, 1982.

[53] M. Veanes. Uniform representation of recursively enumerable sets with simultaneous rigid E-unification. Technical Report 126, Uppsala University, Computing Science Department, July 1996.

[54] W. Wechler. *Universal Algebra for Computer Scientists*, volume 25 of *EATCS Monographs on Theoretical Computer Science*. Springer Verlag, Berlin, 1992.