Zeitschrift: L'Enseignement Mathématique

Herausgeber: Commission Internationale de l'Enseignement Mathématique

Band: 27 (1981)

Heft: 1-2: L'ENSEIGNEMENT MATHÉMATIQUE

Artikel: ALTERNATION AND THE ACKERMANN CASE OF THE DECISION PROBLEM

Autor: Fürer, Martin

DOI: https://doi.org/10.5169/seals-51744

Nutzungsbedingungen

Die ETH-Bibliothek ist die Anbieterin der digitalisierten Zeitschriften auf E-Periodica. Sie besitzt keine Urheberrechte an den Zeitschriften und ist nicht verantwortlich für deren Inhalte. Die Rechte liegen in der Regel bei den Herausgebern beziehungsweise den externen Rechteinhabern. Das Veröffentlichen von Bildern in Print- und Online-Publikationen sowie auf Social Media-Kanälen oder Webseiten ist nur mit vorheriger Genehmigung der Rechteinhaber erlaubt. Mehr erfahren

Conditions d'utilisation

L'ETH Library est le fournisseur des revues numérisées. Elle ne détient aucun droit d'auteur sur les revues et n'est pas responsable de leur contenu. En règle générale, les droits sont détenus par les éditeurs ou les détenteurs de droits externes. La reproduction d'images dans des publications imprimées ou en ligne ainsi que sur des canaux de médias sociaux ou des sites web n'est autorisée qu'avec l'accord préalable des détenteurs des droits. En savoir plus

Terms of use

The ETH Library is the provider of the digitised journals. It does not own any copyrights to the journals and is not responsible for their content. The rights usually lie with the publishers or the external rights holders. Publishing images in print and online publications, as well as on social media channels or websites, is only permitted with the prior consent of the rights holders. Find out more

Download PDF: 16.10.2025

ETH-Bibliothek Zürich, E-Periodica, https://www.e-periodica.ch

ALTERNATION AND THE ACKERMANN CASE OF THE DECISION PROBLEM [1]

by Martin FÜRER [2]

ABSTRACT. The Ackermann prefix class is the set of all formulas of predicate calculus (first order logic without function symbols) with quantifier prefix $\exists \ldots \exists\forall\exists \ldots \exists$. This is one of the few prefix classes for which satisfiability is decidable. Lower bounds for the computational complexity of this decision problem and the $\forall\exists$ sub-problem are presented. The tool to get the main result is the alternating Turing machine. An introduction to alternating Turing machines is given, because they are probably the most remarkable new subject of automata theory, and are well known only to computer scientists.

1. INTRODUCTION AND HISTORICAL BACKGROUND

From the beginning of this century to the thirties, the problem of deciding universal validity of first order formulas, moved slowly to the center of interest of mathematical logic. Especially Hilbert considered it to be a fundamental problem. As it seemed too hard to solve the decision problem (or Entscheidungsproblem) in general, the main approach was to restrict the class of formulas (for which a decision algorithm should work) by very simple syntactic criteria. An earlier example of this kind of restriction was the decidability result of Löwenheim [29] for the monadic (only unary predicate symbols) predicate calculus. Later the main such criterion was the form of the quantifier sequence for formulas in prenex form (see [14], [28], [43] for other syntactically defined classes). There is a duality between universal validity and satisfiability. A closed formula (i.e. a formula of predicate calculus without free variables) is universally valid, iff its negation is not satisfiable. Around 1930 the decidability of the satisfiability

---

[1]) Presented at the *Symposium über Logik und Algorithmik* in honour of Ernst SPECKER, Zürich, February 1980.

[2]) This work was supported by the British Science Research Council and by the Swiss National Fonds.

problem for closed formulas with the following prefixes has been shown ($\exists^*$ ($\forall^*$) stands for finite sequences of existential (universal) quantifiers).

Bernays and Schönfinkel [7]	$\exists^* \forall^*$
Ackermann [1]	$\exists^* \forall\exists^*$
Gödel [16, 17], Kalmár [23], Schütte [33, 34]	$\exists^* \forall\forall\exists^*$

Independent of Ackermann, Skolem [39] solved $\forall\exists^*$. Bernays and Schönfinkel [7] solved the $\forall\exists$ case before.

These results are for predicate calculus without equality. But for the first two cases, the methods can be extended to predicate calculus with equality (see [2] or [14]). Dreben has discovered that the same extension is not obvious in the Gödel-Kalmár-Schütte case. Dreben's conjecture that this case might be more difficult with equality was supported by Aanderaa and Goldfarb with various examples. Recently Goldfarb [18] (see also [14]) has shown this case not to be primitive recursive. It is not known, if it is decidable.

It took a long time to prove that for predicate calculus without equality the subclasses of $\exists^* \forall^*$ and $\exists^* \forall\forall\exists^*$ are the only decidable prefix classes. The major steps in this direction were: Church [12] showed that the predicate calculus is undecidable. Turing [41] gave a more direct proof of this fact using Turing machines. So in the thirties, it was known that not all prefix classes allow an algorithmic solution. Undecidability results have been obtained by several researchers for prefix classes containing prefixes of arbitrary length (see [2, p. 61]). The gap was narrowed by Surányi [40] who showed that the prefix classes obtained from $\forall\forall\exists \wedge \forall\forall\forall$ are undecidable. Here the refined classification according to conjunctions of formulas in prenex form is used. The formulas of the class $\forall\forall\exists \wedge \forall\forall\forall$ allow a straightforward transformation to formulas of both the prefix classes $\forall\forall\forall\exists$ and $\forall\forall\exists\forall$. With an elegant proof Büchi [8] showed the undecidability for $\exists \wedge \forall\exists\forall$. Wang [43, 44] has invented several versions of domino problems. They represent an intermediate step between computations and formulas. Infinite computations are technically harder to describe by formulas, than the corresponding domino problems. Using dominoes, Kahr, Moore and Wang [22] got rid of the additional $\exists$ (which seemed necessary to describe a start configuration) and showed the undecidability of $\forall\exists\forall$. With this result, the decidability problem for all prefix classes was solved. The undecidability of $\forall\exists \wedge \forall\forall\forall$ follows as a corollary, and all other undecidability results in the refined classification

follow immediately [8], while all decidable classes are contained in the classes of the form

$$\exists^*\forall^* \wedge \exists^* \forall^* \wedge \ldots \wedge \exists^* \forall^*$$

or

$$\exists^*\forall\forall\exists^* \wedge \exists^*\forall\forall\exists^* \wedge \ldots \wedge \exists^*\forall\forall\exists^* .$$

At the same time, as these purely syntactically defined subclasses of the predicate calculus have been investigated, many decidability and undecidability results for mathematical theories (i.e. for classes of sentences that are not selected primarily from a syntactical point of view) have been obtained.

In recent years many researchers have investigated the computational complexity of decidable theories. But very few have looked at this problem for syntactically (in the above vague sense) defined sets of formulas, except for propositional calculus. One of these few is Kozen [25] who got the surprising result that predicate calculus without negation is *NP*-complete. The other is Lewis [27] who has investigated the computational complexity, just of the (above defined) decidable prefix classes and the monadic predicate calculus. (The latter was investigated before by Rackoff [31].) The work of Asser [4], Mostowski [30], Bennett [5], Jones and Selman [21], and of Christen [11] about spectra is related to this field.

The reason, why the computational complexity of these problems is of interest is not that we would like to know how many hours we have to spend, in order to decide if a certain formula is satisfiable. It is very much the same, as the logicians have not been interested to use their decidability results to decide for many formulas, if they are satisfiable. Nevertheless the decidability problem was considered to be a fundamental question of deep mathematical significance. In the same sense, we claim that the precise asymptotic computational complexity of a natural class of formulas is a fundamental mathematical property. Naturally this does not mean that complexity results are of no importance for the computational practice. But at least some results (mostly huge lower bounds) are not so directly applicable. What they can do, is to improve our understanding of the investigated problem, show connections to other problems, and give us hints for a better understanding of the reasons for the complexity of certain problems, and for the different qualities of complexity.

From a practical computational point of view, the deterministic time complexity is certainly the most important complexity measure. But other measures have been developed, such as nondeterministic and space measures

and certain more complex measures concerning alternating Turing machines. The claim that the computational complexity is an important mathematical notion is supported by the fact that natural problems are in nice complexity classes.

Lewis [27] has given good complexity bounds for the decidable prefix classes of the predicate calculus. The largest gap is in his result about the Ackermann class $\exists^* \forall \exists^*$. Lewis claims an upper deterministic time bound $c^{n/\log n}$. His lower bound is polynominal space. Naturally he conjectures that the lower space bound is the correct bound, as problems containing quantifiers do not tend to have good deterministic time bounds. And in any case, there are very few problems known with good deterministic time bounds. The usual method to prove lower time bounds is to describe Turing machine computations. So just as well a nondeterministic Turing machine can be chosen, yielding even a nondeterministic lower time bound.

But there are a few methods to prove deterministic lower time bounds, using tools of automata theory, namely the auxiliary pushdown automaton of Cook [13], or the auxiliary stack automaton of Ibarra [20], or the alternating versions of them, investigated by Ladner, Lipton and Stockmeyer [26], or the alternating Turing machine. The latter seems to be the most interesting, but so far it has not had too many applications. And most applications are to problems involving games or sequences of quantifiers with an unbounded number of quantifier alternations. We apply alternating Turing machines to get a $c^{n/\log n}$ deterministic lower time bound for the Ackermann case of the decision problem. This is an application of alternating Turing machines in a new field, where it is not obvious that this tool can be successful.

Here is a summary of the rest of this paper. In the next section a short presentation of some notions from logic, and in section three from computational complexity is given. In section four, the alternating Turing machine is introduced.

In section five, a transformation from the full Ackermann class to the monadic Ackermann class is described. This is a good transformation for the classes $\exists^p \forall \exists^*$ (only p existential quantifiers in front of the universal quantifier) with constant p. But for the class $\exists^* \forall \exists^*$ this transformation is via length order $n^2/\log n$ instead of n. It is not clear, if a better transformation exists. Alternating Turing machines can test the satisfiability of $\exists^* \forall \exists^*$ formulas more directly than deterministic Turing machines. They are used here to get the optimal upper bound of Lewis [27] for the monadic case, because with alternating Turing machines a polynomial

space upper bound for the $\exists^* \forall \exists$ subcase is obtained at the same time. It is easy to see that the class $\exists^* \forall$ is *NP*-complete.

Section six contains the main result, namely the $c^{n/\log n}$ lower bound for the $\forall\exists\exists$ case, and also a tight lower bound for the $\forall\exists$ case, as well as some *NP*-complete problems. In the last section are some conclusions.

2. Some notions from logic

The formulas of first order logic (see e.g. Shoenfield [36]) are built from:

— variables $y, x_1, x_2, \ldots z_1, z_2, \ldots$

— function symbols $f, g, f_L, f_R, f_1, f_2, \ldots$
(we use $c, c_1, c_2, \ldots$ for 0-any function symbols, i.e. constants)

— predicate symbols $P, P_1, P_2, \ldots$ (and other capitals)

— auxiliary symbols (,)

— equality symbol $=$

— propositional symbols $\wedge, \vee, \neg, \rightarrow, \leftrightarrow$

— quantifiers $\forall, \exists$

We use $F[x/t]$ to denote the result of the *substitution* of the term t for the variable x in the formula F.

A formula $Q_1 x_1 Q_2 x_2 \ldots Q_n x_n F_0$ with Q_i quantifiers (for $i = 1, \ldots, n$) and F_0 quantifier-free is in *prenex form*. F_0 is called the *matrix* of the formula.

We are investigating decision procedures for formulas of first order logic without equality and without function symbols. But we use the functional form of formulas.

The *functional form* of a formula in prenex form is constructed by repeatedly changing

$\forall x_1 \forall x_2 \ldots \forall x_n \exists y F$ (F may contain quantifiers) to

$\forall x_1 \forall x_2 \ldots \forall x_n F[y/f_i(x_1, \ldots, x_n)]$

using each time a new n-ary function symbol f_i until no more existential quantifiers appear.

A formula is satisfiable, iff its functional form is satisfiable. In addition, both are satisfiable by structures of the same cardinality.

We use α, α' to denote structures. A *structure* α for a first order language L consists of:

— a nonempty set $|\alpha|$ (the universe of α),
— a function $f^\alpha : |\alpha|^n \to |\alpha|$ for each n-ary function symbol f of L, (in particular an individual (= element) c^α of $|\alpha|$ for each constant c of L),
— a predicate $P^\alpha : |\alpha|^n \to \{\text{true, false}\}$ for each n-ary predicate symbol P in L.

f^α and P^α are called interpretations of f and P.

A structure for a language L defines a truth-value for each closed formula (i.e. formula without free variables) of L in the obvious way (see e.g. [36]). A structure α is a *model* of a set of closed formulas, if all the formulas of the set get the value true (i.e. are *valid* in α). A formula F is *satisfiable*, if its negation $\neg F$ is not valid.

Let α be the following structure for a language L without equality:

The universe $|\alpha|$ (the Herbrand universe) is the set of terms built with the function symbols of L (resp. of L together with the constant c, if L contains no constants (= 0-ary function symbols)). Each function symbol f is interpreted by f^α with the property: For each term t, $f^\alpha(t)$ is the term $f(t)$. We call such an α a Herbrand structure. If a formula F (in the language L) is valid in α, then we call α a *Herbrand model* of F.

The following version of the Löwenheim Skolem theorem is very useful for our investigations.

THEOREM. *The functional form of a closed formula without equality is satisfiable iff it has a Herbrand model.* □

This theorem can be proved with the methods developed by Löwenheim [29] and completed as well as simplified by Skolem [38]. The version of Skolem [37] which uses the axiom of choice, has less connections with this theorem. Also in Ackermann [2] and Büchi [8] versions of the above theorem are present. Probably for the first time, Ackermann [1] constructs a kind of Herbrand model, the other authors use natural numbers instead.

3. SOME NOTIONS FROM COMPUTATIONAL COMPLEXITY

We use one-tape Turing machines and multi-tape Turing machines with a two-way read-only input tape and, if necessary, a one-way write-only output tape. The other tapes are called work tapes. The Turing machine

alphabet Γ and the input alphabet Σ are any finite set of symbols with $\Sigma \subseteq \Gamma$ and $B \in \Gamma - \Sigma$, where B is the blank symbol.

A language L is a set of (finite) words over a finite alphabet. Σ^* is the set of all words over the alphabet Σ.

We write $f(n)$ if we mean the function $f : \mathbf{N} \to \mathbf{R}$. This unprecise notation is standard in computational complexity. In connection with Turing machines n denotes always $|w|$, the length of the input word,

$$f(n) = O(g(n)) \text{ means } \exists c\, \exists n_0\, \forall n \geqslant n_0\, f(n) \leqslant c\, g(n)$$

A language L_1 (over Σ) is *logspace transformable* to a language L_2 (over Σ) *via length order* $g(n)$, if there exists a function $f : \Sigma^* \to \Sigma^*$ such that:

$$f(w) \in L_2 \text{ iff } w \in L_1 \quad \text{for all } w \in \Sigma^*,$$

$|f(w)| = O(g(|w|))$, and there exists a multi-tape Turing machine which computes f, scanning only $O(\log n)$ tape squares of the work tapes.

We use the following complexity classes:

$DTIME(f(n)) = \{L \mid L$ is accepted by a deterministic Turing machine in at most $f(n)$ steps (for all $w \in \Sigma^*$ with $n = |w|)\}$

$NTIME(f(n)) = \{L \mid L$ is accepted by a nondeterministic Turing machine in at most $f(n)$ steps (for all $w \in L$ with $n = |w|)\}$

$DSPACE(f(n)) = \{L \mid L$ is accepted by a deterministic Turing machine using at most $f(n)$ tape squares on each work tape$\}$

$NSPACE(f(n)) = \{L \mid L$ is accepted by a nondeterministic Turing machine using at most $f(n)$ tape squares on each work tape$\}$

$$P = \bigcup_{c,k \in \mathbf{N}} DTIME(c + n^k)$$

$$NP = \bigcup_{c,k \in \mathbf{N}} NTIME(c + n^k)$$

$$POLYSPACE = \bigcup_{c,k \in \mathbf{N}} DSPACE(c + n^k) = \bigcup_{c,k \in \mathbf{N}} NSPACE(c + n^k)$$

It is easy to see that $DSPACE(f(n)) = DSPACE([c\, f(n)])$ for all positive constants c. This linear *speed-up* is done by increasing the alphabet size. The same theorems hold for nondeterministic and alternating (defined below) Turing machines. In the corresponding theorems for time complexity (and for one-tape space complexity) $c\, f(n)$ is replaced by $\max(n+1, c\, f(n))$, and they hold if $\lim n/f(n) = 0$.

A *configuration* of a Turing machine consists of its state, the position(s) of its head(s), and the contents of the tape(s).

Configurations are described by instantaneous descriptions (ID). An *ID* of a one-tape (infinite only to the right) Turing machine with finite tape contents (i.e. almost everywhere is the blank symbol) is the representation of its configuration by a word, built from an initial segment of the tape inscription which contains the non-blank part. In this segment the scanned symbol s is replaced by (s, q), where q is the state of the configuration.

4. Alternating Turing machines

We assume that the reader is familiar with (one version of) deterministic Turing machines. In nondeterministic Turing machines (see e.g. [3]), the scanned symbol(s) do not determine a move (new symbol(s) and shift of head(s)), but a finite set of moves. By choosing any move of this set, the Turing machine follows a computation path. The nondeterministic Turing machine accepts, iff at least one computation path leads to an accepting configuration (i.e. a configuration with accepting state). Chandra and Stockmeyer [10] and Kozen [24] have extended the concept of nondeterministic Turing machines to alternating Turing machines [9]. Nondeterministic machines involve an existential quantification (there exists a path). Alternating machines are a natural extension involving universal as well as existential quantification. This extension from nondeterministic to alternating machines, works for all kinds of abstract machine models, but we look here only at alternating Turing machines.

Definitions (Automata theory)

Alternating Turing machines have two disjoint sets of states, existential and universal states. Configurations and successor configurations (reachable in one move) are defined as for nondeterministic Turing machines, but the conditions for acceptance are different.

An *accepting computation tree* of an alternating Turing machine M with input w is a finite tree T whose nodes are labeled with configurations of M according to the conditions:

a) The root of T is labeled with the start configuration of M with input w.

b) If a node is labeled with a configuration C, then all descendants are labeled with successor configurations of C.

c) Nodes labeled with a non accepting existential configuration have at least one descendant.

d) If C' is a successor configuration of a universal configuration C, and a node N is labeled by C, then at least one descendant of N is labeled with C'.

e) All leaves are labeled with accepting configurations.

The *language* $L(M)$ accepted by an alternating Turing machine M is the set of all words w, such that there exists an accepting computation tree of M with input w.

Hence a nondeterministic Turing machine is an alternating Turing machine with only existential states.

A very intuitive way of looking at alternating Turing machines is: Two players A and E make moves (maybe not strictly alternating) beginning in the start configuration of M with input w. Player A moves from universal configurations, and E from existential configurations to successor configurations. E wins if (after finitely many moves) an accepting configuration is reached. The input w is in $L(M)$ iff E has a winning strategy.

One might first think alternating Turing machines accept all arithmetic sets and even more. But naturally, exactly the recursively enumerable sets are accepted by alternating Turing machines, because every deterministic Turing machine is an alternating Turing machine, and alternating Turing machines can easily be simulated by deterministic Turing machines.

What goes wrong if we want a player (A or E) of an alternating Turing machine to choose any natural number, is that this player could decide for ever that he wants to choose an even bigger number (computation trees have only finite branching).

The situation changes drastically if Turing machines do not accept by entering one accepting state, but by infinitely often entering accepting states. Then nondeterministic Turing machines accept exactly the Σ_1^1-sets of the analytical hierarchy, while deterministic Turing machines accept exactly the Π_2^0-sets of the arithmetical hierarchy. (It is not known, if the sets accepted by alternating Turing machines in this way have such a nice characterisation.) This remark is just to indicate that automata theory might be useful for non-recursive sets too.

Alternating Turing machines are important for several reasons. First they are a very natural extension of nondeterministic Turing machines, and they are closely related to the fundamental concept of quantifiers. Second they are a basic model of parallel computation, which is of growing importance with modern technology. And third, there are beautiful relations

between the power of time and space bounded versions of ordinary and alternating Turing machines. Some versions of alternating Turing machines with restricted alternating power (see Berman [6], Ruzzo [32]) are able to bridge the gaps between deterministic and nondeterministic time and space complexity classes. And the problems about the relation of these classes (e.g. $P = NP$? $P = POLYSPACE$?) are still the most challenging open questions in computational complexity.

Definitions (Complexity)

$ATIME(T(u))$ is the class of languages L accepted by alternating Turing machines M, such that for each input w, there exists an accepting computation tree (of M with input w) of depth $\leqslant T(n)$ (for $n = |w|$) if $w \in L$, and there exists no accepting computation tree if $w \notin L$.

$ASPACE(S(n))$ is the class of languages L accepted by alternating Turing machines M, such that for each input w, there exists an accepting computation tree (of M with input w), whose labels are $S(n)$-space bounded configurations (for $n = |w|$) if $w \in L$, and there exists no accepting computation tree if $w \notin L$. (A configuration is $S(n)$-space bounded if at most $S(n)$ tape squares on work tapes are used.)

The fundamental complexity relations between alternating and non-alternating Turing machines are (Chandra and Stockmeyer [10], Kozen [24]):

For $S(n) \geqslant n$

$$ATIME\ (S(n)) \subseteq DSPACE(S(n))$$

and

$$NSPACE(S(n)) \subseteq ATIME\ (S(n)^2)$$

For $T(n) \geqslant \log n$

$$ASPACE(T(n)) = \bigcup_{c>0} DTIME(c^{T(n)})$$

We sketch the proof of $DTIME(c^{T(n)}) \subseteq ASPACE(T(n))$, because we use this fact for the complexity result about the Ackermann case.

Let $C_{t,s}$ be the s-th symbol in the ID (instantaneous description) of the configuration at time t, of a $c^{T(n)}$ time bounded deterministic one-tape Turing machine M with input w. The computation of M is simulated backwards.

Player E says, M accepts w by entering the accepting state q_a at time t with headposition s_t. And to prove this, he presents $t-1$, a_{t-1}, b_{t-1}, c_{t-1} and $s = s_{t-1}$ ($t-1$ and s in binary), claiming that $C_{t-1,s-1} = a_{t-1}$,

$C_{t-1,s} = b_{t-1}$ and $C_{t-1,s+1} = c_{t-1}$. (Naturally these values must imply the state q_a and the headposition s_t at time t.) Now player A is allowed to doubt one of these three claims, by playing the integer $s' \in \{s-1, s, s+1\}$, and player E has to justify his claim for $C_{t-1,s'}$ by claiming values for $C_{t-2'\,s'-1}$, $C_{t-2,s'}$ and $C_{t-2,s'+1}$ which imply his value for $C_{t-1,s'}$ etc. Finally the value claimed for $C_{0s''}$ is checked by comparison with the s''-th input symbol. If it is correct, then player E, otherwise player A wins.

If w is accepted by M, then the winning strategy for player E is to make always correct claims. If w is not accepted by M, then player A has a winning strategy. He always doubts one of the wrong claims of player E.

5. Upper bounds

Proposition. 1. *For all* $p \geqslant 0$, *the* $\exists^p \forall \exists^*$ *class is logspace transformable to the monadic* $\exists \forall \exists^*$ *class via length order* n.

2. *The* $\exists^* \forall \exists^*$ *class is logspace transformable to the monadic* $\exists^* \forall \exists^*$ *class via length order* $n^2/\log n$.

Proof. The main ideas of this proof are due to Lewis [27, Lemma 7.1] and Ackermann [2, Section VIII.1]. Given a formula F of the class $\exists^p \forall \exists^q$ with prefix $\exists x_1 \ldots \exists x_p \forall y \exists z_1 \ldots \exists z_q$ and matrix M, let S be the set of atomic formulas in M. We define the set S' by $S' = S \cup \{A[y/x_i] \mid A \in S$ and $1 \leqslant i \leqslant p\}$.

Let $S' = \{A_1, \ldots, A_r\}$.
Then $|S'| = r \leqslant (p+1)|S|$.

Now we change the matrix M of F to get the formula F' with matrix M' by replacing (for $j = 1, \ldots, r$) all occurrences of the atomic formula A_j by $P_j(y)$ (for a new monadic predicate symbol P_j) and by adding — as a conjunct to M — a set B of biconditionals.

The set B is constructed to ensure that every Herbrand model α' of the functional form of the formula F' (with matrix M') defines immediately a model α of the functional form of F by $|\alpha| = |\alpha'|$,

$c_k^\alpha = c_k^{\alpha'} = c_k$, $k = 1, \ldots, p$ (where c_k is the replacement of x_k in the functional forms of F and F'),

$f_k^\alpha = f_k^{\alpha'}$, $k = 1, ..., q$ (where $f_k(y)$ is the replacement of z_k in the functional forms of F and F'),

$P^\alpha(b_1, ..., b_n) = P_j^{\alpha'}(b)$, if $A_j \in S'$, $b \in |\alpha'|$, $b_1, ..., b_n \in |\alpha|$ and there exist variables $v_1, ..., v_n$ fulfilling for all i, k the following properties:

a) $A_j = P(v_1, ..., v_n)$,

b) if $v_i = x_k$ then $b_i = c_k^\alpha$,

c) if $v_i = y$ then $b_i = b$,

d) if $v_i = z_k$ then $b_i = f_k^\alpha(b)$.

$P^\alpha(b_1, ..., b_n)$ is defined arbitrarily (e.g. false) if no such A_j and b exist. There might exist several A_j and b having these properties. To ensure that in this case the definition of $P^\alpha(b_1, ..., b_n)$ is correct, i.e. independent of the particular choice of A_j and b, we conjoin the set B of biconditionals to the matrix M.

Take any n-tupel $(b_1, ..., b_n) \in |\alpha|^n$. In the following cases, several $A_j \in S'$ and $b \in |\alpha|$ might satisfy the conditions a), b), c), d):

1. $\{b_1, ..., b_n\} \subseteq \{c_1^\alpha, ..., c_p^\alpha\}$.
2. There is a b' in $\{c_1^\alpha, ..., c_p^\alpha\}$ such that $\{b_1. ..., b_n\} \subseteq \{c_1^\alpha, ..., c_p^\alpha, f_1^\alpha(b'), ..., f_q^\alpha(b')\}$.
3. There is a b'' in $\{b_1, ..., b_n\}$, such that $\{b_1, ..., b_n\} \subseteq \{c_1^\alpha, ..., c_p^\alpha, b''\}$.

To make the definition correct in case 1, we add to B the following biconditionals:

If there is an A_j in S' such that $A_j = P(v_1, ..., v_n)$ with $\{v_1, ..., v_n\} \subseteq \{x_1, ..., x_p\}$, we add

$$P_j(y) \leftrightarrow P_j(x_1)$$

If $A_j = P(v_1, ..., v_n)$ with $\{v_1, ..., v_n\} \subseteq \{x_1, ..., x_p, y\}$ and $A_j[y/x_i] = A_{j'}[y/x_k]$ (for $A_j \neq A_{j'}$), then we add

$$P_j(x_i) \leftrightarrow P_{j'}(x_k).$$

Note: Here the length of the monadic formula might grow quadratically in p.

To make the definition correct in the case when 2 but not 3 holds, we add to B for all j, j', i with $A_j[y/x_i] = A_{j'}[y/x_i]$ the formula

$$P_j(x_i) \leftrightarrow P_{j'}(x_i).$$

To make the definition correct, when 3. but not 2. holds, we add to B the following biconditionals.

For all j, j', k such that $A_j = P(v_1, ..., v_n)$ with

$$y \in \{v_1, ..., v_n\} \subseteq \{x_1, ..., x_p, y\}$$

and $A_j[y/z_k] = A_{j'}$, we add

$$P_j(z_k) \leftrightarrow P_{j'}(y)$$

If both 2. and 3. but not 1. hold, and if there are atomic formulas A_j and $A_{j'}$, such that A_j contains y but no variables of $\{z_1, ..., z_q\}$ and $A_j[y/z_k] = A_{j'}[y/x_i]$, we have to make sure that

$$P_j^{\alpha'}(f_k^{\alpha'}(c_i^{\alpha'})) = P_{j'}^{\alpha'}(c_i^{\alpha'}).$$

But in this case S' contains an $A_{j''}$ with

$$A_{j''} = A_j[y/z_k]$$

and we have added the formulas:

$$P_j(z_k) \leftrightarrow P_{j''}(y) \qquad \text{(case 3)}$$

and

$$P_{j''}(x_i) \leftrightarrow P_{j'}(x_i) \qquad \text{(case 2)}$$

Hence

$$P_j^{\alpha'}(f_k^{\alpha'}(c_i^{\alpha'})) = P_{j''}^{\alpha'}(c_i^{\alpha'}) = P_{j'}^{\alpha'}(c_i^{\alpha'})$$

It is not obvious that the transformation from formula F to formula F' can be done in logarithmic space, because F might contain variables or predicate symbols with excessively long indices. But then a simple trick solves the problem. Instead of writing such an index on a work tape, only a pointer (= position number) to its location on the input tape is stored on a work tape.

If $|F| = n$, then at most $O(n/\log n)$ different atomic formulas appear in F (i.e. $|S| = O(n/\log n)$). The number $|S'|$ of different atomic formulas in F' is then bounded by $c(p+1)|S|$. Hence the transformation from F to F' is via length order n for constant p and via length order $n^2/\log n$ in general (i.e. for $p = O(n/\log n)$). □

Problem. Is there an efficient transformation from the $\exists^* \forall \exists^*$ class to the monadic $\exists^* \forall \exists^*$ class via length order n ?

THEOREM (Upper bound). *The satisfiability of the monadic prefix class* $\exists^* \forall \exists^*$ *is decidable by an alternating Turing machine* M *in space*

$O(n/\log n)$. *Furthermore* M *enters no universal states for formulas of the subclass* $\exists^* \forall \exists$.

Proof. Let the input F be the monadic formula

$$\exists x_1 \ldots \exists x_p \forall y \exists z_1 \ldots \exists z_q F_0$$

with F_0 quantifier-free. It is easy to find out if the input has this form or not. Let F_0 contain m different atomic formulas. Then $m = O(n/\log n)$ for $n = |F|$.

Let $(v_1, \ldots, v_{p+q+1})$ be $(x_1, \ldots, x_p, y, z_1, \ldots, z_q)$ and let $A_1, \ldots, A_m$ be the atomic formulas $P_j(v_i)$ of F_0 in lexicographical order according to (i, j).

$T_1, \ldots, T_m$ is a sequence of truth values for the atomic formulas. (The atomic formula A_k is interpreted to be true if T_k = true.)

The alternating Turing machine M executes the following satisfiability test:

Program

1. begin

   for all k such that the atomic formula A_k contains an x_i, choose existentially T_k to be true or false;
   for $r := 1$ to max $(1, p)$ do
   begin

2. for all k, k', j such that A_k is $P_j(y)$ and $A_{k'}$ is $P_j(x_r)$ do $T_k := T_{k'}$;

3. for all k, j such that A_k is $P_j(y)$ and $P_j(x_r)$ does not appear in F do choose existentially a value of $\{$true, false$\}$ for T_k;

4. for counter $:= 1$ to 2^m do
   begin

5. for all k such that A_k is a $P_j(z_i)$ do choose existentially a truth value for T_k; check that the interpretation of the atomic formulas A_k $(k = 1, \ldots, m)$ by T_k gives the value true to the matrix F_0, otherwise stop rejecting;

7. if $q = 0$ then goto E;
   if $q = 1$ then $s := 1$ (i.e. $z_s = z_1$);
   if $q > 1$ then choose universally a value from $\{1, \ldots, q\}$ for s;

8. for all k, k', j such that A_k is $P_j(y)$ and $A_{k'}$ is $P_j(z_s)$ do $T_k := T_{k'}$;

9. for all k such that (for any j) A_k is $P_j(y)$ and $P_j(z_s)$ does not appear in F do choose existentially a truth value for T_k;

end;

E : end;

stop accepting;

end.

To execute this program, the alternating Turing machine M uses only space

m to count to 2^m,

m to store $T_1, \ldots, T_m$,

$\log p < \log m$ to store r,

$c \log n$ for anxillary storage, especially to store position numbers of certain information on the input tape, e.g. long indices, which are not copied to the work tapes.

Because $m = O(n/\log n)$, there is an upper bound $O(n/\log n)$ (independent of p and q) for the space used by M.

We have to show that the above program decides satisfiability of the formula F correctly.

Let $F' = \forall y\, F_0'$ be the functional form of $F = \exists x_1 \ldots \exists x_p \forall y \exists z_1 \ldots \exists z_q\, F_0$, obtained by replacing x_i by c_i and z_i by $f_i(y)$.

a) Let F' (and F) be satisfiable and let α be a model of F'.
We think the program of M extended by:

before 2. $\quad b := c_r^\alpha$

before 8. $\quad b := f_s^\alpha(b)$

Then good existential choices for the truth values T_k are

if $A_k = P_j(x_i)$ then $T_k := P_j^\alpha(c_i^\alpha)$

if $A_k = P_j(y)$ then $T_k := P_j^\alpha(b)$

if $A_k = P_j(z_i)$ then $T_k := P_j^\alpha(f_i^\alpha(b))$

The computation tree defined by these existential choices accepts the formula F.

b) Assume the alternating Turing machine M accepts the formula F. Then each minimal accepting computation tree (without unnecessary branches) of M with input F can be used to construct a Herbrand model α of F'.

Note that the Herbrand universe

$$|\alpha| = \{c_1, \ldots, c_p, f_1(c_1), \ldots, f_2(f_1(c_3)), \ldots\}$$

(as a set of terms) and the functions $f_1^\alpha, \ldots, f_q^\alpha$ of a possible Herbrand model of F' are uniquely defined. We have to define the predicates $P_1^\alpha, P_2^\alpha, \ldots$.

We look at the program extended by

$$b := c_r^\alpha \qquad \text{(before 2) and}$$
$$b := f_s^\alpha(b) \qquad \text{(before 8) as in a).}$$

All elements of $|\alpha|$ with nesting depth $\leqslant 2^m$ are assigned to b somewhere in the accepting computation tree. The current values of the sequence $T_1, \ldots, T_m$ define some truth values of predicates in $c_1^\alpha, \ldots, c_p^\alpha, b, f_1^\alpha(b), \ldots, f_q^\alpha(b)$ by

$$\begin{aligned} P_i^\alpha(c_k^\alpha) &= T_j \quad \text{if} \quad A_j = P_i(x_k) \\ P_i^\alpha(b) &= T_j \quad \text{if} \quad A_j = P_i(y) \\ P_i^\alpha(f_k^\alpha(b)) &= T_j \quad \text{if} \quad A_j = P_i(z_k)\,. \end{aligned}$$

The other truth values of the predicates P_i^α are defined arbitrarily. This method of defining predicates of b is used on each path in the tree $(|\alpha|, f_1^\alpha, \ldots, f_q^\alpha)$, only until the first repetition of all truth values on that path. That happens on each path in a depth $\leqslant 2^m$. Let b' be the node on the path to b with the same truth values for all predicates as b. Then (inductively) the predicates are defined to have the same values on the subtree with root b as on the subtree with root b'. The so constructed structure α is a model of F. □

COROLLARY 1 (Lewis [27]). *The set of satisfiable formulas of the monadic* $\exists^* \forall \exists^*$ *class is (for a constant* $c > 1$*) in* $DTIME(c^{n/\log n})$.

Proof. The alternating Turing machine of the upper bound theorem can be simulated in deterministic time $c^{n/\log n}$. □

The direct construction of a deterministic $c^{n/\log n}$ time decision procedure of Lewis [27] is easier. He starts with a big structure (with 2^m elements, where m is the number of predicate symbols), and eliminates bad elements of this structure, to get either a model or the non-existence of a model.

We have chosen the decision procedure by an alternating Turing machine to get the following result for free.

COROLLARY 2. *The satisfiable formulas of the monadic* $\exists^* \forall \exists$ *class are in* $NSPACE\,(n/\log n)$.

Proof. The universal states of the alternating Turing machine M which decides the monadic $\exists^* \forall \exists^*$ class are not used for the subclass $\exists^* \forall \exists$. If we drop them, we get a nondeterministic Turing machine. □

By combining the proposition with the upper bound theorem we get immediately.

COROLLARY 3. *The satisfiable formulas of the* $\exists^* \forall \exists^*$ *class are in* $DTIME\,(c^{(n/\log n)^2})$ *for some* c. □

COROLLARY 4. *The satisfiable formulas of the* $\exists^* \forall \exists$ *class are in* $NSPACE\,((n/\log n)^2)$. □

Lewis [27] claims the same time bound in Corollary 3 as for the monadic case. But this seems not to work. For example, if $P(x_1, y), \ldots, P(x_p, y)$ and $P(y, x_1), \ldots, P(y, x_p)$ appear in the formula, then p^2 truth values for $P^\alpha(c_i^\alpha, c_j^\alpha)$ $(i, j = 1, \ldots, p)$ have to be guessed.

But these upper bounds are not very good, as e.g. in Corollary 3 the Turing machine could be replaced by one which works a short time $(O((n/\log n)^2)$ steps) nondeterministically and then only $c^{n/\log n}$ steps deterministically.

The $\exists^* \forall$ *class*

Formulas of the $\exists^* \forall$ class are transformed by our procedure in monadic formulas again of the $\exists^* \forall$ class. For these formulas, the procedure of the upper bound theorem works in nondeterministic polynomial time. On the other hand the $\exists^* \forall$ class is certainly more difficult than propositional calculus. Therefore the set of satisfiable formulas of the $\exists^* \forall$ class is NP-complete. (NP-completeness is discussed in [15].)

In fact, as the Herbrand models of the satisfiable formulas of the $\exists^p \forall^q$ class, have only max $(p, 1)$ elements, it is easy to see that the satisfiability problem for all the following classes in NP-complete:

a) $\exists^p \forall^q \quad p + q \geqslant 1$

b) $\exists^* \forall^q \quad q \geqslant 0$

c) $\forall^*$

d) $\exists \forall^*$

But the classes $\exists\exists\forall^*$ and $\exists^* \forall^*$ need $NTIME\ c^{n/\log n}$ resp. c^n.

6. Lower bounds

Definition. A *marked binary number* is a word over the alphabet $\{0, \mathit{0}, 1, \mathit{1}\}$ described by the regular expression $(0 \cup 1)^* \mathit{0}\, \mathit{1}^* \cup \mathit{1}^*$. The *value* of a marked binary number is given by the homomorphism h with $h(0) = h(\mathit{0}) = 0$ and $h(1) = h(\mathit{1}) = 1$, i.e. by disregarding the type of the digits.

Note: The digits in italics are those which will change their value when the marked binary number is increased by one.

Marked binary numbers allow the following local tests:

1. A word over the alphabet $\{0, \mathit{0}, 1, \mathit{1}\}$ is a marked binary number, iff the last digit is in italics and only the following adjacent pairs of digits occur:

   a) 00, 01, 0*0*, 10, 11, 1*0* (0, 1 or *0* behind 0 or 1), and

   b) *01*, *11* (*1* behind *0* or *1*).

2. For two right adjusted marked binary numbers x and y with y below x holds:

   value $(x) + 1 =$ value (y) iff only the following vertically adjacent pairs of digits occur:

   a) 0 or *0* below 0 or *1* and

   b) 1 or *1* below *0* or 1.

THEOREM (Lower bound). *If a language L is accepted by a linear space bounded alternating Turing machine M, with at most q successors for each universal configuration, then L is polynomial time transformable to the set of satisfiable formulas of the monadic $\forall\exists^q$ class via length order $n \log n$.*

Proof. We can assume that M is a one-tape alternating Turing machine accepting L in space $n + 1$ and time $2^m - 1$ for an $m = O(n)$. We describe the case $q = 2$. To each input w of M, we define (using function symbols f_L and f_R) the functional form $F(w)$ of a formula $F'(w)$ of the monadic $\forall\exists^q$ class, such that:

Claim A: $w \in L$ iff $F(w)$ is satisfiable.

Before we define the formula $F(w)$, we show how to construct a structure α from an accepting computation tree, such that α will turn out to be a model of $F(w)$.

If w is accepted by M, then there is an accepting computation tree CT with the properties:

— Every node of the tree with depth less than $2^m - 1$ has exactly two sons, and every node with depth $2^m - 1$ is a leaf. I.e. it is a complete binary tree.

— If the same configuration appears in several nodes, then the corresponding successor configurations are the same.

Therefore, there are functions succ_L and succ_R, which define the instantaneous descriptions of the successor configurations in the tree. Furthermore, we can choose succ_L and succ_R in such a way that they have the following property:

For every pair consisting of a state and a scanned symbol, we consider the possible moves of M to be an ordered set.

If ID is a universal instantaneous description, then succ_L (ID) is the first and succ_R (ID) is the second successor of ID.

If ID is existential, then succ_L (ID) and succ_R (ID) are arbitrary successors of ID (typically succ_L (ID) = succ_R (ID)).

If ID is accepting, then succ_L (ID) = succ_R (ID) = ID.

Given functions succ_L and succ_R and an accepting computation tree CT of depth $2^m - 1$ with the above properties, we define now the structure α, such that:

Claim B: α is a model of $F(W)$.

1. The universe $|\alpha|$ is the set $\{(t, \text{ID}) \mid t$ is an integer with $0 \leqslant t \leqslant 2^m - 1$ and ID is the instantaneous description of a configuration occuring in a branch of the computation tree CT of M with input w at time $t\}$.
2. f_L (resp. f_R) is interpreted by a function mapping (t, ID) for $t < 2^m - 1$ to $(t+1, \text{succ}_L(\text{ID}))$ (resp. $(t+1, \text{succ}_R(\text{ID}))$) and $(2^m - 1, \text{ID})$ to (0, start ID for input w). succ_L (ID) (succ_R (ID)) is defined to be the instantaneous description of the left (right) successor configuration of ID.
3. In (t, ID) the monadic predicates are interpreted as follows:

$$\text{Let } t = \sum_{i=0}^{m-1} b_i 2^i \text{ with } b_i \in \{0, 1\},$$

and let ID $= a_0 \dots a_{k-1} (a_k, q) a_{k+1} \dots a_n$ with $a_j \in \Sigma$ (alphabet) and $q \in Q$ (states).

Then the $O(m)$ monadic predicate symbols $B_j, M_j, Z, L_{j'}, S_p, T_{\sigma j}$ with $j \in \{0, \dots, m-1\}$, $j' \in \{0, \dots, n\}$, $p \in Q$ and $\sigma \in \Sigma$ are interpreted as

B_j^α $((t, \mathrm{ID}))$ is true iff $b_j = 1$
M_j^α $((t, \mathrm{ID}))$ is true iff $b_j = 1$ for all $i < j$, i.e. b_j is marked
Z^α $((t, \mathrm{ID}))$ is true iff $b_i = 0$ for all i
$L_{j'}^\alpha$ $((t, \mathrm{ID}))$ is true iff $j' = k$
S_p^α $((t, \mathrm{ID}))$ is true iff $p = q$
$T_{\sigma j'}^\alpha$ $((t, \mathrm{ID}))$ is true iff $a_{j'} = \sigma$

We now define the formula $F(w)$ and add some remarks about the intended meaning of its subformulas. This makes it obvious that claim B holds. $F(w)$ is the formula

$$\forall_y [F_H(y) \wedge F_V(y, f_L(y)) \wedge F_V(y, f_R(y)) \wedge F_0(y) \wedge F_U(y) \wedge F_W(y) \wedge F_L(y, f_L(y)) \wedge F_R(y, f_R(y)) \wedge F_A(y)]$$

where

a) $F_H(y)$ *is* $\bigwedge_{0 \leq j \leq m-2} [M_{j+1}(y) \leftrightarrow (M_j(y) \wedge B_j(y))] \wedge M_0(y)$

The intended meaning is:
All binary numbers are correctly marked. (H stands for horizontal condition.)

b) $F_V(y, z)$ is $\bigwedge_{0 \leq j \leq m-1} [B_j(z) \leftrightarrow (M_j(y) \leftrightarrow \neg B_j(y))]$

The intended meaning is:
The level number below level number l is $l + 1$. (V stands for vertical condition.)

c) $F_0(y)$ is $[\bigwedge_{0 \leq j \leq m-1} \neg B_j(y)] \leftrightarrow Z(y)$

The intended meaning is:
The configuration at level 0 is distinguished by Z.

d) $$F_U(y) \text{ is } [\bigwedge_{0 \leq j \leq n} \bigwedge_{\substack{\sigma, \sigma' \in \Sigma \\ \sigma \neq \sigma'}} \neg (T_{\sigma j}(y) \wedge T_{\sigma' j}(y))] \wedge [\bigwedge_{\substack{q, q' \in Q \\ q \neq q'}} \neg (S_q(y) \wedge S_{q'}(y))]$$

The intended meaning is:

For every configuration there is at most one symbol in every tape cell, and the Turing machine is in at most one state.

e) $F_W(y)$ is $Z(y) \rightarrow [\bigwedge_{0 \leq j \leq n} T_{\sigma_j j}(y) \wedge L_0(y) \wedge \bigwedge_{1 \leq j \leq n} \neg L_j(y) \wedge S_{q_0}(y)]$

where $\sigma_0 \sigma_1 \ldots \sigma_n$ is wb (the input w extended by a blank endmarker B), and q_0 is the start state of M. This is the only subformula of $F(w)$ depending not only on $n = |w|$, but also on w. Its intended meaning is:
The distinguished configuration at level 0 is the start configuration.

f) Exactly as for nondeterministic Turing machines, it is possible to check if ID_1 is a successor of ID_0 by writing ID_1 below ID_0 and checking all 6-tuples seen through a window of length 3 and height 2 which is pushed over the two words, and by checking that no head of the Turing machine walks in or out of the tape portion represented by the instantaneous descriptions. In this way, we check

— for universal ID_0, if the left son is labeled with $\mathrm{succ}_L(\mathrm{ID}_0)$ and the right son is labeled with $\mathrm{succ}_R(\mathrm{ID}_0)$;

— for existential ID_0, just if both sons are labeled with any successors;

— for accepting ID_0, if both sons are labeled with ID_0.

It is easy to construct a formula $P_j^L(y, z)$ $(P_j^R(y, z))$ expressing the window condition at the positions $j, j+1, j+2$ for the ID's in node y and in its left (right) son z.

$P_j^L(y, z)$ and $P_j^R(y, z)$ are built from the atomic formulas

$$S_p(y), S_p(z) \quad \text{for } p \in Q$$

and $$L_{j'}(y), L_{j'}(z) \quad \text{for } j' = j, j+1, j+2$$

and $$T_{\sigma j'}(y), T_{\sigma j'}(z) \quad \text{for } j' = j, j+1, j+2 \quad \text{and} \quad \sigma \in \Sigma.$$

The length of $P_j^L(y, z)$ and $P_j^R(y, z)$ are bounded by a constant times the maximal length of the atomic formulas.

For $D = L$ and $D = R$,

$$F_D(y, z) \text{ is } Z(z) \vee [\bigwedge_{0 \leq j \leq n-2} P_j^D(y, z)$$

$$\wedge (L_0(z) \rightarrow (L_0(y) \vee L_1(y))) \wedge (L_n(z) \rightarrow (L_{n-1}(y) \vee L_n(y)))].$$

g) $$F_A(y) \text{ is } [B_{m-1}(y) \wedge M_{m-1}(y)] \rightarrow \bigvee_{q \in Q_a} S_q(y)$$

Q_a is the set of accepting states of M. The intended meaning of $F_A(y)$ is:
At the deepest level $2^m - 1$, all branches of the computation tree accept.

Now the formula $F(w)$ is defined, and for $w \in L$ it should be clear that $F(w)$ is satisfiable and has the model α.

We still have to show the other direction of claim A. If $F(w)$ is satisfiable, then $w \in L$. Let α be a model of $F(w)$. In α the formula

$$\forall y\,[F_H(y) \wedge F_V(y, f_L(y)) \wedge F_V(y, f_R(y))]$$

is valid. Hence for all $b \in |\alpha|$ a level number $l(b)$ is defined by the interpretation of the predicate symbols B_j in α. The level numbers have the property

$$l(f_L^\alpha(b)) = l(f_R^\alpha(b)) = l(b) + 1 \bmod 2^m .$$

Therefore (as $|\alpha|$ is non-empty), there are elements of all levels mod 2^m, in particular, there is an element b_0 of level 0.
Because $\forall y\,[F_0(y) \wedge F_w(y)]$ is valid in α, the truth values of the predicates L_j^α, S_p^α and $T_{\sigma j}^\alpha$ in b_0 encode the start configuration of the alternating Turing machine M with input w.
Let $|\alpha|'$ be the subset of $|\alpha|$ which is accessible from b_0 by several applications of f_L^α and f_R^α. Then the validity of

$$\forall y\,[F_U(y) \wedge F_L(y, f_L(y)) \wedge F_R(y, f_R(y))]$$

in α ensures that the predicates L_j^α, S_p^α and $T_{\sigma j}^\alpha$ define for all $b \in |\alpha|'$ a unique instantaneous description ID (b) such that ID $(f_L(b))$ is a left successor of ID (b), and ID $(f_R(b))$ is a right successor of ID (b).
Finally, the validity of $\forall y\, F_A(y)$ guarantees that the computation tree is accepting.

It is easy to check that $F(w)$ contains only $O(n)$ atomic formulas, each of length $O(\log n)$. Therefore $|F(w)| = O(n \log n)$. It is also obvious that the formula $F'(w)$ and its functional form $F(w)$ can be computed from w in logarithmic space by a Turing machine. Note that most parts of $F(w)$ depend only on $n = |w|$. □

COROLLARY 1. *There is a $c > 1$ such that no deterministic Turing machine accepts the satisfiable formulas of the monadic $\forall\exists\exists$ class in time $O(c^{n/\log n})$.*

Proof. By standard diagonalization arguments, there is a language L in $DTIME\ (c_2^n)$ which is not in $DTIME\ (c_1^n)$ for $c_1 < c_2$ [19].

L is then in $ASPACE\ (n)$. Assume Corollary 1 is not true. Then by first transforming L according to the lower bound theorem to the monadic $\forall\exists\exists$ class, and then accepting this language fast, L could be accepted in deterministic time c_1^n. □

COROLLARY 2. *For every nondeterministic Turing machine M which accepts the satisfiable formulas of the monadic $\forall\exists$ class, there exists a constant c, such that M uses space $cn/\log n$ for infinitely many inputs.*

Proof. We use the hierarchy result for *NSPACE* [35] and the fact that an alternating Turing machine with only one successor configuration for each universal configuration, is a nondeterministic Turing machine. □

CONCLUSIONS

Alternating Turing machines are a powerful tool in the few areas where applications have been found so far. They can make connections visible, which are not seen otherwise. It seems impossible to find the lower bound for the Ackermann case of the decision problem, without knowing alternating Turing machines. Even knowing the result, a direct description of the computation of a deterministic exponential time bounded Turing machine M by a $\exists^* \forall\exists^*$ formula, without obviously copying the simulation of M by an alternating Turing machine, seems impossible.

We are used to think that nondeterministic machines correspond to existential quantifiers (e.g. satisfiability in propositional calculus), and that alternating machines correspond to a sequence of alternating quantifiers (e.g. quantified boolean formulas, i.e. the theory of $\{0, 1\}$ with equality). This paper shows that this needs not always to be the case.

Examples

1. Not only the satisfiability problem of the $\exists^*$ class, but also of the $\forall^*$ class is *NP*-complete (not co-*NP*-complete).
2. Adding an existential quantifier to the $\forall$ prefix class, means moving from a time to a space complexity class.

3. Adding another existential quantifier to the $\forall\exists$ prefix class means moving from a nondeterministic (space) to a deterministic (time) complexity class.

One possible continuation of this work, is to investigate the complexity of the decision problem for formulas with simple quantifier patterns in decidable theories. For most of the decidable theories, huge lower bounds are known, because a class of formulas with so many quantifier alternations, that they hardly appear in practice, is shown to be difficult to decide.

Acknowledgment

The deterministic lower time bound $c^{n \setminus \log n}$ for the $\exists^*\forall\exists^*$ case has been obtained independently by Harry R. Lewis (Complexity Results for Classes of Quantificational Formulas. *J. of Computer and System Sciences 21*, No. 3, Dec. 1980, pp. 317-353). His method is quite different and uses alternating pushdown automata.

REFERENCES

[1] Ackermann, Wilhelm. Über die Erfüllbarkeit gewisser Zählausdrücke. *Mathematische Annalen 100* (1928), pp. 638-649.

[2] —— *Solvable Cases of the Decision Problem.* North Holland, Amsterdam, 1954.

[3] Aho, A. V., J. E. Hopcroft and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[4] Asser, G. Das Repräsentantenproblem im Präditatenkalkül der ersten Stufe mit Identität. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik 1* (1955), pp. 252-263.

[5] Bennett, J. On spectra. Doctoral Dissertation, Princeton University, N.J., 1962.

[6] Berman, L. Precise bounds for Presburger arithmetic and the reals with addition. *Proceedings 18th Symposium on Foundations of Computer Science*, 1977, pp. 95-99.

[7] Bernays, Paul und Moses Schönfinkel. Zum Entscheidungsproblem der mathematischen Logik. *Mathematische Annalen 99* (1928), pp. 342-372.

[8] Büchi, J. R. Turing machines and the Entscheidungsproblem. *Mathematische Annalen 148* (1962), pp. 201-213.

[9] Chandra, A. K., D. C. Kozen and L. J. Stockmeyer. Alternation. *Journal of the ACM 28* (1981), pp. 114-133.

[10] Chandra, A. K. and L. J. Stockmeyer. Alternation. *Proc. 17th Symposium on Foundations of Computer Science*, 1976, pp. 98-108.

[11] Christen, C. Spektren und Klassen elementarer Funktionen. Doctoral Dissertation, ETH Zürich, 1974.

[12] Church, Alonzo. A note on the Entscheidungsproblem. *Journal of Symbol. Logic 1* (1936), pp. 40-41; Correction. *ibid.*, pp. 101-102.

[13] COOK, S. A. Characterization of push-down machines in terms of time-bounded computers. *Journal of the Association for Computing Machinery 18* (1971), pp. 4-18.

[14] DREBEN, B. and W. D. GOLDFARB. *The Decision Problem: Solvable Classes of Quantificational Formulas.* Addison-Wesley, Advanced Book Program, Reading, Massachusetts, 1979.

[15] GAREY, M. R. and D. S. JOHNSON. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco, 1979.

[16] GÖDEL, Kurt. Ein Spezialfall des Entscheidungsproblems der theoretischen Logik. *Ergebnisse eines mathematischen Kolloquiums 2*, 1932, pp. 27-28.

[17] —— Zum Entscheidungsproblem des logischen Funktionenkalküls. *Monatsch. Math. Phys. 40* (1933), pp. 433-443.

[18] GOLDFARB, W. D. To appear in: *The Journal of Symbolic Logic.*

[19] HARTMANIS, J. and R. E. STEARNS. On the computational complexity of algorithms. *Transactions of the American Mathematical Society 117* (1965), pp. 285-306.

[20] IBARRA, O. H. Characterization of some tape and time complexity classes of Turing machines in terms of multihead and auxiliary stack automata. *J. Comput. System Sci. 5* (1971), pp. 88-117.

[21] JONES, N. G. and A. L. SELMAN. Turing machines and the spectra of first-order formulas. *Journal of Symbolic Logic 39* (1974), pp. 139-150.

[22] KAHR, A. S., E. F. MOORE und Hao WANG. Entscheidungsproblem Reduced to the AEA Case. *Proc. Nat. Acad. Sci. USA 48* (1962), pp. 365-377.

[23] KALMÁR, László. Über die Erfüllbarkeit derjenigen Zählausdrücke, welche in der Normalform zwei benachbarte Allzeichen enthalten. *Mathematische Annalen 108* (1933), pp. 466-484.

[24] KOZEN, D. On parallelism in Turing machines. *Proc. 17th Symp. on Foundations of Computer Science*, 1976, pp. 89-97.

[25] —— First order predicate logic without negation is *NP*-complete. Report No. 77-307, Dept. of Computer Science, Cornell University, Ithaca, NY, 1977.

[26] LADNER, R. E., R. J. LIPTON and L. J. STOCKMEYER. Alternating Pushdown Automata. *Proc. 19th Symp. on Foundations of Computer Science*, 1978, pp. 92-106.

[27] LEWIS, Harry R. Complexity of solvable cases of the Decision Problem for Predicate Calculus. *Proc. 19th Symp. on Foundations of Computer Science*, 1978, pp. 35-47.

[28] —— *Unsolvable Classes of Quantificational Formulas.* Addison-Wesley, Advanced Book Program, Reading, Massachusetts, 1979.

[29] LÖWENHEIM, Leopold. Über Möglichkeiten im Relativkalkül. *Mathematische Annalen 76* (1915), pp. 447-470; English translation in [42], pp. 228-251.

[30] MOSTOWSKI, A. Concerning a problem of H. Scholz. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik 2* (1956), pp. 210-214.

[31] RACKOFF, C. The complexity of theories of the monadic predicate calculus. Technical Report, IRIA, 1975.

[32] RUZZO, L. Tree-Size Bounded Alternation. *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*, 1979, pp. 352-359.

[33] SCHÜTTE, Kurt. Untersuchungen zum Entscheidungsproblem der mathematischen Logik. *Mathematische Annalen 109* (1934), pp. 572-603.

[34] —— Über die Erfüllbarkeit einer Klasse von logischen Formeln. *Mathematische Annalen 110* (1934), pp. 161-194.

[35] SEIFERAS, J. I., M. J. FISCHER and A. R. MEYER. Refinements of the nondeterministic time and space hierarchies. *Proceedings of 14th IEEE Symposium on Switching and Automata Theory*, 1973, pp. 130-137.

[36] SCHOENFIELD, J. R. *Mathematical Logic.* Addison-Wesley, Series in Logic, 1967.

[37] SKOLEM, Thoralf. Logisch-kombinatorische Untersuchungen über die Erfüllbarkeit oder Beweisbarkeit mathematischer Sätze nebst einem Theorem über dichte Mengen. *Videnskapsselskapets skrifter, I. Matematisk-naturvidenskabelig klasse, No. 4* (1920), 36 pp; English translation in [42], pp. 252-263.

[38] —— Einige Bemerkungen zur axiomatischen Begründung der Mengenlehre. *Matematikerkongressen i Helsingfors den 4-7 Juli 1922, Den femte skandinaviska matematikerkongressen, Redogörelse* (Akademiska Bokhandeln, Helsinki, 1923), pp. 217-232; English translation in [42], pp. 290-301.

[39] —— Über die mathematische Logik. *Norsk matematisk tidsskrift 10* (1928), pp. 125-142; English translation in [42], pp. 508-524.

[40] SURÁNYI, János. Zur Reduktion des Entscheidungsproblems des logischen Funktionenkalküls. *Mat. es Fizikai Lapok 50* (1943), pp. 51-74.

[41] TURING, Alan Mathison. On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc. 42* (1936/37), pp. 230-265. A correction, *ibid. 43* (1937), pp. 544-546.

[42] VAN HEIJENOORT, Jean. *From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931.* Harvard University Press, Cambridge, Massachusetts, 1971.

[43] WANG, Hao. Proving Theorems by Pattern Recognition II. *The Bell System Technical Journal 40* (1961), pp. 1-41.

[44] —— Dominoes and the AEA case of the decision problem. *Proceedings of a Symposium on the Mathematical Theory of Automata,* Polytechnic Institute of Brooklyn, New York, 1962, pp. 23-55.

(Reçu le 28 septembre 1980)

Martin Fürer

Computer Science Department
University of Edinburgh
Edinburgh EH9 3JZ
Scotland