Tarskian Set Constraints

Robert Givan, David McAllester, Carl Witty and **Dexter Kozen**

Robert Givan
Dept. of Electrical & Computer Engineering
Purdue University
1285 EE Building
West Lafayette, IN 47907
Phone: (765)494-9068
Email: givan@ecn.purdue.edu
Web: http://www.ece.purdue.edu/~givan/

David McAllester
P. O. Box 971
AT&T Labs Research
180 Park Avenue
Florham Park, NJ, 07932 USA
Phone: (973)-360-8318
Email: dmac@research.att.com
Web: http://www.research.att.com/~dmac/

Carl Witty
Newton Research Labs
4140 Lind Avenue SW
Renton, WA 98055
Email: cwitty@newtonlabs.com
Phone: (425)251-9600

Dexter Kozen
Computer Science Department
Upson Hall
Cornell University
Ithaca, New York 14853-7501
Phone: (607) 255-9209
Email: kozen@cs.cornell.edu
Web: http://www.cs.cornell.edu/kozen/

Abstract

We investigate set constraints over set expressions with Tarskian functional and relational operations. Unlike the Herbrand constructor symbols used in recent set constraint formalisms, the meaning of a Tarskian function symbol is interpreted in an arbitrary first order structure. We show that satisfiability of Tarskian set constraints is decidable in nondeterministic doubly exponential time. We also give complexity results and open problems for various extensions and restrictions of the language.

Keywords: Set constraints, decision procedures, dynamic logic, mu-calculus.

1. Introduction

There has been considerable interest recently in formalisms for describing and reasoning about sets. Here we consider a family of formalisms that have received surprisingly little attention. Consider a set expression of the form $f(C_1, \ldots, C_n)$ where $C_1, \ldots, C_n$ denote sets. In recent work on set constraints, operation symbols are interpreted as Herbrand term constructors so that the set expression $f(C_1, \ldots, C_n)$ denotes the set of terms $f(t_1, \ldots, t_n)$ where $t_1 \in C_1, \ldots, t_n \in C_n$. But an equally natural interpretation takes $f(C_1, \ldots, C_n)$ to be the set of values that can derived by applying the *meaning* of *f* to elements of the sets denoted by $C_1, \ldots, C_n$. For example, if + denotes addition and *O* denotes the set of odd integers then we would expect $+(O, O)$ to denote all the integers that can be expressed as the sum of two odds, i.e., all the even integers. In general we can let the meaning of operations be determined by a first order structure in the standard way, and view any subset assertion between set expressions as constraining both the set variable meanings *and the operation symbol meanings* for the variables and operation symbols that appear in the assertion. We call set expressions under this form of semantics "Tarskian" to distinguish them from the "Herbrand" set expressions that have received considerable recent attention.

Tarskian set constraints seem fundamentally different from Herbrand set constraints. There does not seem to be any simple reduction of Tarskian set constraints to the monadic class. Since Tarskian set constraints are not restricted to Herbrand interpretations, induction principles for Herbrand interpretations do not apply. It turns out that Tarskian set constraints are closely related to modal logics. Before stating our main results on Tarskian constraints we review some work on set calculi. We organize the review around four classes of set calculi — Herbrand set constraints, modal logics, AI concept languages, and Tarskian set constraints.

Herbrand set constraints involve set expressions generated by the grammar

$$C \ ::- \ X \ | \ f(C_1, \ldots, C_n) \ | \ C_1 \cap C_n \ | \ \neg C \ , \qquad (1)$$

where X is any set variable, and f is any Herbrand function symbol. A set expression of the form $f(C_1, \ldots, C_n)$ is taken to denote the set of all terms $f(t_1, \ldots, t_n)$ with $t_i \in C_i$. A set constraint is an expression of the form $C_1 \subseteq C_2$. Herbrand set constraints are largely inspired by applications to the static analysis of computer programs (Heintze and Jaffar, 1990b) (Heintze and Jaffar, 1990a) (Frühwirth, et. al., 1991) (Aiken et. al., 1994). The problem of determining satisfiability of a finite set of Herbrand set constraints is the problem of determining whether there is any interpretation of the set variables appearing in the constraints as sets of terms such that all the constraints are true relative to the interpretation. This problem is known to be complete for nondeterministic exponential time (Aiken et. al., 1993a) (Bachmair, et. al., 1993). The problem remains decidable in nondeterministic exponential time if one adds both negative constraints, *i.e.*, $C_1 \not\subseteq C_2$, (Aiken et. al., 1993b) (Charatonik and Pacholski, 1994a), and projection functions (Charatonik and Pacholski, 1994b).

Modal logics involve formulas which are true or false of possible worlds in Kripke structures. Equivalently, each formula of a modal logic can be taken to denote the set of worlds in which it is true. Since modal formulas denote sets, modal logics can be viewed as set calculi. Propositional dynamic logic (PDL) (Fischer and Ladner, 1979) (Pratt, 1980) and the modal μ-calculus (Kozen, 1983) are particularly significant modal logics. If R is a binary relation symbol and C is a set expression then in both these logics the set expression $[R]C$ denotes the set $\{x : \forall y \ R(x, y) \rightarrow y \in C\}$. The set expression $\langle R \rangle C$ is defined analogously to denote $\{x : \exists y \in C \ R(x, y)\}$. The modal μ-calculus allows for recursively defined set expressions of the form $\mu X.C[X]$ where X is a set variable and $C[X]$ is a set expression in which every occurrence of X in $C[X]$ occurs inside an even number of negation signs. PDL can be seen as a restriction of the modal μ-calculus which has much simpler decision procedures and yet is sufficiently expressive to cover many applications. Satisfiability for both PDL and the modal μ-calculus are known to be complete for deterministic exponential time (Street and Emerson, 1989) (Emerson and Jutla, 1988) (Safra, 1988).

AI concept languages have been developed for knowledge representation in expert systems (Brachman and Schmolze, 1985) (Schmidt-Schaub and Smalka, 1991). The set expressions of concept languages are constructed from set variables and relation variables using a variety of compositional mechanisms. For example, the expression $\forall R.C$ where R is a relation expression and C is a set expression denotes the set $\{x : \forall y\ R(x, y) \Rightarrow y \in C\}$ (and hence is a syntactic variant of $[R]C$). For the most part these languages can be viewed as fragments of PDL (Calvanese et. al., 1994) (Giacomo and Lenzerini, 1994b) (Giacomo and Lenzerini, 1994a). However, many of these languages have satisfiability problems in P, NP, or PSPACE (Donini, et. al., 1991). Also, concept languages often include cardinality primitives which appear not to be expressible in PDL. Furthermore, there is a natural relationship between certain concept languages and Montague grammar for natural language. In particular the set expression $R(\text{every } C)$ is taken to be the set $\{x : \forall y \in C\ R(x, y)\}$. This provides a natural meaning for English verb phrases such as "contains every prime number." One simple but expressive Montagovian concept language has a polynomial time satisfiability problem (McAllester and Givan, 1992).

Tarskian set expressions have been studied by Jònnson and Tarski in the framework of Boolean algebra with operations (Jònnson and Tarski, 1951) (Jònnson and Tarski, 1952). In the work of Jònnson and Tarski the operation f in the expression $f(C_1, \ldots, C_n)$ actually denotes a relation on $n + 1$ arguments. More specifically, $f(C_1, \ldots, C_n)$ denotes $\{y : \exists x_1 \in C_1, \ldots, \exists x_n \in C_n\ \langle x_1, \ldots, x_n, y\rangle \in f\}$. One can think of f as a nondeterministic operation — for any given tuple of inputs there is a set of possible outputs. Jònnson and Tarski's main result is a variant of the Stone representation theorem which can be viewed as a completeness theorem for an algebraic axiomatization. They did not study decision theoretic complexity issues. Representation theorems for subclasses of Boolean algebras with operations have recently been studied in a general setting by Goldblatt (Goldblatt, 1989). Kozen (Kozen, 1993) has recently obtained a Stone duality in the context of Herbrand set constraints between the algebra of set constraints and the topological term automata of (Kozen, et. al., 1993) and (Kozen, et. al., 1994).

Here we consider a superset of the original set expressions studied by Jònnson and Tarski. We make a syntactic distinction between deterministic and nondeterministic operation symbols corresponding to classical function symbols and relation symbols respectively. We use this nonstandard terminology so that we can write set expressions of the form $f(C_1, \ldots, C_n)$ where f is an operation symbol (either deterministic or nondeterministic). We also allow least fixed point expressions. The complete grammar of our Tarskian set expressions is as follows.

$$C \ ::= \ X \mid f(C_1, \ldots, C_n) \mid C_1 \cap C_n \mid \neg C \mid \mu X . C \qquad (2)$$

In the above grammar f can be either deterministic or nondeterministic and may take no arguments, *i.e.*, be a constant symbol. $\mu X . C$ is restricted so that X can only occur inside an even number of negation symbols in C. We consider finite sets of constraints of the form $C_1 \subseteq C_2$ or $C_1 \not\subseteq C_2$.

In spite of the apparent naturality of Tarskian set constraints, their computational properties have not been widely studied. It is shown in (McAllester and Givan, 1993) that satisfiability of nonrecursive Tarskian set constraints not involving Boolean operations is decidable in cubic time (assuming unit time hash table operations). It is shown in (Givan and McAllester, 1992) that satisfiability of constraints on expressions involving meets, joins, and monotone applications in an arbitrary lattice is similarly decidable in cubic time. The results of this paper are summarized in the table below. We categorize Tarskian set constraint satisfiability problems by the presence or absence of recursion (μ-sets), the presence or absence of functions (deterministic operations of arity at least one), and the presence or absence of constants (deterministic operations of arity zero). In all cases we allow nondeterministic operations (of all arities) and both positive and negative set constraints.

	Rec	**Fun**	**Const**	**Lower Bound**		**Upper Bound**	
1.	–	–	–	EXPTIME	Sec. 3.1	EXPTIME	Sec. 4.2
2.	–	–	+	EXPTIME	Sec. 3.1	EXPTIME	Sec. 4.3
3.	–	+	–	NEXPTIME	Sec. 3.2	NEXPTIME	Sec. 4.4
4.	–	+	+	NEXPTIME	Sec. 3.2	2-NEXPTIME	Sec. 4.5
5.	+	–	–	EXPTIME	Sec. 3.1	EXPTIME	Sec. 5
6.	+	–	+	EXPTIME	Sec. 3.1	?	
7.	+	+	–	NEXPTIME	Sec. 3.2	?	
8.	+	+	+	Undecidable	Sec. 3.3	?	

Table 1: Summary of Results with Pointers to Relevant Paper Sections

The results in the first two lines of the table are proved using techniques similar to those used for PDL (Pratt, 1980) (see Sections 3.1, 4.2, and 4.3). The lower bound in line three is proved using techniques similar to those used in proving NEXPTIME hardness for the monadic class (Lewis, 1980) (see Section 3.2). The upper bound in line three is proved by a filtration-like argument (see Section 4.4).

Standard techniques fail for the fourth line upper bound, the case of nonrecursive constraints with arbitrary operations. We show in Section 4.5 that satisfiability for nonrecursive Tarskian set constraints is decidable in nondeterministic doubly exponential time. Our procedure involves a reduction to a natural class of

Diophantine constraints which we call *prequadratic*. We show that satisfiability for prequadratic Diophantine constraints is decidable in nondeterministic exponential time. However, we conjecture that prequadratic Diophantine satisfiability is in NP. If so, then we get a nondeterministic singly exponential procedure for nonrecursive Tarskian constraints.

The fifth line in the table corresponds to recursive constraints with nondeterministic operations. It turns out that constraint set satisfiability in this calculus is linear time equivalent to set expression satisfiability in the modal □-calculus. We show in Section 5 that constraint set satisfiability for this class is polynomial time reducible to closed set expression satisfiability in a calculus we call the Herbrand □-calculus. Closed set expression satisfiability for the Herbrand □-calculus is known to be decidable in exponential time.

Decision procedures for the modal □-calculus can be viewed as consisting of two phases. The first phase can be viewed as a reduction of set expression satisfiability in the modal calculus to set expression satisfiability in the closed Herbrand calculus. The second phase is a decision procedure for the closed Herbrand calculus. The formal justification for the first phase is rather elaborate (Street and Emerson, 1989). Here we give an alternative reduction from the modal □-calculus to the closed Herbrand □-calculus with a simplified correctness proof.

We believe it likely that techniques used in decision procedures for the modal □-calculus can be used to construct decision procedures for lines six and seven, although this has not yet been done.

The undecidability of the eighth line is proved by a reduction of Hilbert's tenth problem. The reduction, given in Section 3.3, uses only intersection and union constraints (no negation) and only a single level of □-quantification.

It is interesting to note that the difficulties in both lines four and eight arise from the ability to express Diophantine constraints. It seems that both constants and functions of arity at least one are necessary for expressing such constraints.

The remainder of this paper is organized as follows:

- Section 2 is a basic concepts section laying out the terminology we use for the various Tarskian set constraints satisfiability problems;
- Section 3 gives proofs of the lower bounds given above in Table 1;
- Section 4 gives proofs of the upper bounds from Table 1 for the nonrecursive variations of the language; and
- Section 5 gives a proof of the EXPTIME upper bound for recursive Tarskian set constraints without function symbols of any arity.

2. Basic Concepts

We assume a countably infinite collection of set variables and for each arity (number of arguments) an infinite number of deterministic and an infinite number of nondeterministic operation symbols of that arity. We will call deterministic operation symbols of arity zero *constant symbols*, and those of nonzero arity *function symbols*. We consider set expressions generated by the following grammar.[1]

$$C \ ::- \ X \mid f(C_1, \ldots, C_n) \mid C_1 \cap C_2 \mid \neg C \mid \mu X . C \tag{3}$$

We also write $C_1 \cup C_2$ as an abbreviation for $\neg(\neg C_1 \cap \neg C_2)$. We take a first order structure M to be a domain set D plus an interpretation, denoted $M(f)$, of each operation symbol f such that if f has arity n then $M(f) \subseteq D^{n+1}$ and such that if f is deterministic then for all $x_1, \ldots, x_n$ in D there exists exactly one y such that $\langle x_1, \ldots, x_n, y \rangle \in M(f)$. A set variable interpretation over a first order structure M is a mapping from set variables to subsets of the domain of M. If ρ is a set variable interpretation then $\rho[X{:=}S]$ is the interpretation identical to ρ except that it interprets the variable X as the set S. For any set expression C, first order structure M with domain D, and set variable interpretation ρ over M we take $M[\![C]\!]\rho$ to be a subset of D defined by the following conditions:

$$\begin{aligned}
M[\![X]\!]\rho &= \rho(X) \\
M[\![f(C_1, \ldots, C_n)]\!]\rho &= \left\{ \begin{array}{c} y : \exists x_1 \in M[\![C_1]\!]\rho, \ldots, \exists x_n \in M[\![C_n]\!]\rho \\ \langle x_1, \ldots, x_n, y \rangle \in M(f) \end{array} \right\} \\
M[\![C_1 \cap C_2]\!]\rho &= M[\![C_1]\!]\rho \cap M[\![C_2]\!]\rho \\
M[\![\neg C]\!]\rho &= D - M[\![C]\!]\rho \\
M[\![\mu X . C]\!]\rho &= S_\kappa \text{ for } \kappa \text{ any cardinal greater than that of domain}(M) \\
\text{where } S_\alpha &= \bigcup_{\beta < \alpha} M[\![C]\!]\rho[X{:=}S_\beta] \text{ for ordinals } \alpha, \beta
\end{aligned}$$

A positive constraint is an expression of the form $C_1 \subseteq C_2$, and a negative constraint of the form $C_1 \not\subseteq C_2$. A pair $\langle M, \rho \rangle$ is called a *model*. We say that a model $\langle M, \rho \rangle$ satisfies the constraint $C_1 \subseteq C_2$ whenever $M[\![C_1]\!]\rho \subseteq M[\![C_2]\!]\rho$. We say $\langle M, \rho \rangle$ satisfies $C_1 \not\subseteq C_2$ if $M[\![C_1]\!]\rho \not\subseteq M[\![C_2]\!]\rho$. We say $\langle M, \rho \rangle$ satisfies a set Σ of constraints if $\langle M, \rho \rangle$ satisfies every member of Σ. We call $\langle M, \rho \rangle$ a *model of* Σ in this case. A constraint set Σ is satisfiable if it is satisfied by some $\langle M, \rho \rangle$. We are interested in determining the satisfiability of finite sets of constraints.

1. The (deterministic or nondeterministic) operation symbol f must have arity n in expressions of the form $f(C_1, \ldots, C_n)$ and all occurrences of X in C in the expression $\mu X . C$ must occur inside an even number of negations.

Because we discuss many variations of the basic Tarskian language, we introduce here a system of abbreviations for the variations considered. We will write “plain Tarskian set constraint” for a constraint with no recursions and no function or constant symbols. We will add a prefix of *R*-, *F*-, and/or *T*- to the word “*Tarskian*” to indicate the possible presence of recursion, function symbols, and/or constant symbols, respectively. So, for example, an *RC-Tarskian* set constraint may contain recursion and/or constant symbols but may not contain function symbols. Except where explicitly mentioned, all languages we consider here allow Boolean operations and nondeterministic operation symbols of arbitrary arity.

3. Lower Bounds

In this section we present the three reductions responsible for all the lower bounds shown in Table 1. First, we show that plain Tarskian constraints have an EXPTIME-hard satisfiability problem by giving a reduction from the acceptance problem for linear-space-bounded alternating Turing machines. Second, we show that the addition of function symbols to the language results in a satisfiability problem that is at least nondeterministic exponential-time hard, by reduction from the satisfiability of “Lewis clauses”. Finally, we show that full *RFC-Tarskian* set constraints have an undecidable satisfiability problem; this is shown by reduction from Hilbert’s tenth problem. All eight lower bounds shown in Table 1 derive directly from these three reductions.

3.1 Lower Bound for Plain Tarskian Constraints

In this section we show that satisfiability of Tarskian set constraints is EXPTIME hard for constraints without recursion or deterministic operation symbols. The results of this section can be contrasted with known results on the satisfiability of *individual* set expressions in these expressively weak languages (Donini, et. al., 1991). In the nonrecursive case satisfiability of individual set expressions is considerably easier than satisfiability of a system of set constraints. We show in Section 5 that when recursive set expressions are allowed but deterministic operations are not, then constraint set satisfiability can be reduced to set expression satisfiability. Without recursion the reduction fails.

It turns out that languages somewhat weaker than Tarskian set constraints without recursion or deterministic operations are still hard for exponential time. We will characterize some weaker languages using the following definitions.

Definition 1. If □ is a set of constraints and □ is a constraint we write □ ⊨ □ to indicate that any model satisfying all the constraints in □ also satisfies □. A *positive entailment problem* is a set □ of *positive* set constraints and a positive set constraint □. The problem is to determine whether □ ⊨ □.

Each positive entailment problem □ ⊨ □ is equivalent to a set constraint satisfiability problem ($\Sigma \cup \{\neg\phi\}$) in which there is exactly one negative constraint. We call a set of constraints with at most one negative constraint a *positive entailment satisfiability problem*. Now consider the following ways in which set expressions can be formed.

- $\cap$ If U and W are set expressions then so is $U \cap W$.
- $\cup$ If U and W are set expressions then so is $\neg(\neg U \cap \neg W)$, which will also be written $U \cup W$.
- $\Diamond$ If C is a set expression and R is a binary operation symbol then $R(C)$, which will also be written $\langle R\rangle C$, is also a set expression.
- $\Box$ If C is a set expression and R is a binary operation symbol then $\neg R(\neg C)$, which will also be written $[R]C$, is also a set expression.

We use the notation $L(F_1, \ldots, F_n)$ to mean the set language with set variables and set formation features $F_1, \ldots, F_n$. For example, $L(\cap, \Diamond)$ is the language whose set expressions are constructed from set variables using only the set formation operations $\cap$ and $\Diamond$ as defined above. All of the languages defined by the above four features are sublanguages of plain Tarskian set constraints. In all of these languages the only occurrences of the set complement operation are the occurrences implicit in set expressions of the form $[R]C$ or $U \cup W$. We show here that the positive entailment satisfiability problem for $L(\Box, \Diamond, \cap)$ and $L(\Box, \Diamond, \cup)$ are both EXPTIME hard. Before doing this we now briefly mention the difficulty of the positive entailment satisfiability problem for other combinations of these features.

The positive entailment satisfiability problems for $L(\cap)$ and $L(\cup)$ can both be reduced in linear time to the satisfiability problem for propositional Horn clauses which is known to be decidable in linear time (Downing and Gallier, 1984), as follows. Any constraint set with no negative constraints is trivially satisfiable by the empty model. If there is one negative constraint, we can focus on a single domain object d witnessing the truth of the negative constraint (*i.e.,* such that $d \in M[\![U \cap \neg W]\!]\rho$ for negative constraint $U \not\subseteq W$) and then treat each set variable P as a proposition symbol whose truth corresponds to $d \in M[\![P]\!]\rho$. Each constraint can then be written as a set of Horn clauses over these proposition symbols so that the resulting set is satisfiable exactly if there exists a model of the positive constraints with at least one witness d to the single negative constraint.

The positive entailment satisfiability problem for $L(\cap, \cup)$ can be shown to be NP complete (we leave this as an exercise for the reader). It is known that satisfiability of Tarskian set constraints not involving Boolean operations or recursion, but with both deterministic and nondeterministic operation symbols of all arities, is decidable in polynomial time (McAllester and Givan, 1993). This implies that the positive entailment satisfiability problem for $L(\Diamond)$ is decidable in polynomial time. By duality arguments given below this implies that the problem for $L(\Box)$ is also decidable in polynomial time. To our knowledge the difficulty of the positive entailment problem for other combinations of these features is open.

It is possible to relax the semantics of Tarskian set expressions so that the “set” expressions denote elements of a lattice and ▯ and ▯ denote least upper bound and greatest lower bound operations respectively. Once we allow an arbitrary lattice (rather than require a complemented distributive lattice), and only require that relations denote monotone operations on lattice elements, then the positive entailment problems for $L(\Diamond, ▯, ▯)$ is decidable in polynomial time (Givan and McAllester, 1992).

Theorem 1: The positive entailment problem for $L(\Box, \Diamond, ▯)$ is EXPTIME hard.

Proof: The proof is by reduction of the acceptance problem for linear space bounded alternating Turing machines. In an alternating Turing machine the states are classified into “universal” and “existential” states and for any given state and input symbol there can be many different next states (as in simple nondeterministic machines). A configuration of the machine consists of a state of the tape, the tape location of the Turing machine head, and the state of the machine. A configuration in which the machine is in a universal (existential, accepting) state is called a universal (existential, directly accepting) configuration. Each configuration has a set of possible next configurations defined by the transition table of the machine in the standard way. The set of accepting configurations is the least set containing all directly accepting configurations (where the machine is in an accept state) and including every universal configuration such that all next configurations are accepting and every existential configuration such that some next configuration is accepting. The linear space alternating Turing machine problem can be phrased as the problem of deciding if a given configuration of a given alternating machine is accepting subject to the restriction that configurations are restricted to ones in which the head occurs on a given set of tape squares (all other configurations are taken to be failing configurations). We can assume without loss of generality that all configurations have exactly two next configurations. We show that this problem is polynomial time reducible to the positive entailment problem for $L(\Box, \Diamond, ▯)$.

In the reduction to from alternating machines to set expressions the set expressions can be viewed as sets of machine configurations, or equivalently each set expression can be viewed as a “proposition” that is true or false of any given configuration. Given any linear space bounded alternating Turing machine we introduce a set variable $X_{n,a}$ for each tape location n and possible tape symbol a. Intuitively $X_{n,a}$ represents the proposition that symbol a is written on square n. We also introduce a set variable H_n for each tape location n representing the proposition that the head is at square n and a proposition Z_s for each machine state s representing the proposition that the machine is in state s. We also have set variables `START` and `ACCEPT` representing, respectively, the propositions “the current configuration is the given initial configuration” and “the current configuration is an accepting configuration.” Finally we have one binary operation symbol N representing the “next configuration” relation. We let Σ be the following set of positive constraints. Each constraint can be viewed as an implication required to hold at all configurations.

1. $\mathtt{START} \Rightarrow H_0$
2. $\mathtt{START} \Rightarrow Z_{s_0}$ where s_0 is the initial state.
3. $\mathtt{START} \Rightarrow X_{n,a}$ where a is the initial symbol on tape square n.
4. A constraint $X_{n,a} \wedge H_n \wedge Z_s \Rightarrow \langle N\rangle(H_{n+1} \wedge Z_w \wedge X_{n,b})$ for each entry in the transition table of the machine which replaces a by b, moves from state s to state w, and moves right. A similar constraint is included for each left-moving entry in the machine table.
5. All constraints of the form $X_{n,a} \wedge H_m \Rightarrow [N]X_{n,a}$ where $n \neq m$.
6. All constraints of the form $Z_s \Rightarrow \mathtt{ACCEPT}$ where s is an accepting state.
7. All constraints of the form

   $$Z_s \wedge \langle N\rangle(Z_u \wedge \mathtt{ACCEPT}) \wedge \langle N\rangle(Z_w \wedge \mathtt{ACCEPT}) \Rightarrow \mathtt{ACCEPT}$$

   where s is a universal state with successor states u and w.
8. All constraints of the form $Z_s \wedge \langle N\rangle\mathtt{ACCEPT} \Rightarrow \mathtt{ACCEPT}$ where s is an existential state.

We now sketch a proof that $\Sigma \models \mathtt{START} \Rightarrow \mathtt{ACCEPT}$ if and only if the given initial configuration is accepting. We first assume that $\Sigma \models \mathtt{START} \Rightarrow \mathtt{ACCEPT}$ and show that the initial configuration is accepting. Suppose not. We can then construct a model $\langle M, \rho\rangle$ of Σ from “reality”, in which the initial configuration is in $M[\![\mathrm{START}]\!]\rho$ but not in $M[\![\mathtt{ACCEPT}]\!]\rho$, contradicting our assumption that $\Sigma \models \mathtt{START} \Rightarrow \mathtt{ACCEPT}$. The domain of M is the set of all configurations reachable from the initial configuration. Each set variable `START`, `ACCEPT`, $X_{n,a}$, H_n, or Z_w is interpreted relative to this domain by ρ according to the intended meanings given above (*e.g.*, H_n is the set of configurations in the domain for which the head is at tape location n). M interprets the binary operation symbol N as true for two configurations γ_1 and γ_2 if and only if γ_2 is reachable in one step from γ_1. It is easy to check that the constraints in Σ are all satisfied by $\langle M, \rho\rangle$ and that the initial configuration is in $M[\![\mathrm{START}]\!]\rho$ but not in $M[\![\mathtt{ACCEPT}]\!]\rho$, as desired, allowing us to conclude by contradiction that $\Sigma \models \mathtt{START} \Rightarrow \mathtt{ACCEPT}$ implies that the initial configuration is accepting.

We now consider the converse direction. We assume that the given initial configuration γ is accepting and prove that $\Sigma \models \mathtt{START} \Rightarrow \mathtt{ACCEPT}$. We say that a configuration γ is n-accepting for natural number n if it satisfies the following conditions:

- if $n = 0$, □ is directly accepting
- if $n > 0$ and the machine is in a universal state in □, then all successors of □ are $(n - 1)$-accepting.
- if $n > 0$ and the machine is in an existential state in □, then some successor of □ is $(n - 1)$-accepting.

We also define the set expression $D(□)$ for each configuration □ to be the expression X_{1, a_1} □ … □ X_{n, a_n} □ H_m □ Z_w where for each i, a_i is the symbol on tape square i in configuration □, the head is at tape square m in configuration □, and the machine is in state w in configuration □.

Constraint types 4 and 5 in □ ensure that $D(□)$ □ □N□$(D(□))$ whenever it is possible for a transition to occur from a configuration □ to a configuration □. Using this fact along with constraint types 6, 7, and 8 we can now show by induction on n that for every n-accepting configuration □ we have □ $\models D(□)$ □ `ACCEPT`, for any n. Since each accepting configuration must be n-accepting for some n, we can conclude that □ $\models D(□)$ □ `ACCEPT` for the given initial configuration □. But the first three types of constraints in □ ensure that □ $\models$ `START` □ $D(□)$, and so by the transitivity of subset we have that □ $\models$ `START` □ `ACCEPT` as desired. ❑

We now establish a general duality principle for all languages defined by subsets of the language features discussed in this section.

Definition 2. If C is a concept expression then we define the dual of C, denoted as C^*, to be the result of simultaneously replacing $<>$ by $\square$, $\square$ by $<>$, □ by □, and □ by □. Note that $(C^*)^* = C$ for any C.

Definition 3. For any variable interpretation □ we define $□^*$ to be the interpretation given by $□^*(X) = D - □(X)$, *i.e.*, $□^*(X)$ is the complement of $□(X)$.

Lemma 1: For any set expression C and model □M, □□, we have $M[\![C^*]\!]$ $□^*$ is equal to $M[\![$ □$C]\!]$ □.

Proof: Push the negation in $\neg C$ down to the set variables using de Morgan's laws and the identities □ □R□$C = [R]$□C and □$[R]C =$ □R□□C. ❑

Definition 4. For any constraint C □ W, we define the dual constraint $(C$ □ $W)^*$ to be W^* □ C^* and for any set □ of constraints we define $□^*$ to be $\{□^* : □ □ □\}$.

The preceding lemma implies that for any constraint □ we have that □M, □□ satisfies □ if and only if □M, $□^*$□ satisfies $□^*$. This yields the following duality lemma.

Lemma 2: (Duality) The entailment relation ▯ $\models$ ▯ holds if and only if the dual relation ▯* $\models$ ▯* holds.

This last lemma allows a direct reduction of the positive entailment problem for $L(\langle\rangle, \square,$ ▯$)$ to the corresponding problem for $L(\langle\rangle, \square,$ ▯$)$ so the latter must also be EXPTIME hard.

Corollary 1: The positive entailment problem for $L(\langle\rangle, \square,$ ▯$)$ is EXPTIME hard.

3.2 Lower Bound for F-Tarskian Constraints

In this section we give a reduction from a fragment of the monadic class known to be complete for nondeterministic exponential time to nonrecursive Tarskian constraints with function symbols but without constant symbols.

Definition 5. A *first order clause* is a first order sentence of the form ▯$x_1, \ldots, x_n$ (▯$_1$ ▯ … ▯ ▯$_k$) where each ▯$_i$ is a first order literal, *i.e.*, either an application of a predicate symbol to terms or the negation of such an application. Let a be a fixed constant symbol and let f be a fixed monadic function symbol. We define a *Lewis clause* (over a and f) to be one of the following:

- An atomic sentence of the form $P(a)$.
- A clause involving a single variable x where every literal contains an application of a monadic predicate to either x or $f(x)$.
- A clause involving exactly two variables in which every literal contains an application of a monadic predicate to one of the two variables.

The following result is due to Lewis (Lewis, 1980).

Theorem 2: (Lewis) Satisfiability for a set of Lewis clauses is complete for NEXPTIME.

Note that Lewis clauses involve only monadic predicates. Function symbols only arise in clauses of the second type. It is not difficult to show that a set of Lewis clauses can be "inverse Skolemized" to produce an equisatisfiable sentence without any function symbols and involving only monadic predicates. Hence Lewis clauses can be viewed as a fragment of the monadic class. The NEXPTIME lower bound for Herbrand set constraints established in (Bachmair, et. al., 1993) was also proved using a reduction of Lewis clauses.

Theorem 3: Satisfiability for F-Tarskian constraint sets is NEXPTIME hard.

Proof: The proof is by reduction of Lewis clause satisfiability. Let C be a set of Lewis clauses. For each monadic predicate symbol in C we select a corresponding zero arity nondeterministic Tarskian operation symbol — by abuse of notation we will denote the selected Tarskian operation symbol for each monadic predicate (*e.g.* P) by the same name (*i.e.*, P). We define a set $T[C]$ of nonrecursive Tarskian set constraints as follows. Let $P_1(a), \ldots, P_n(a)$ be all clauses in C of the form $P(a)$. We include in $T[C]$ the constraint $P_1 \cap \ldots \cap P_n \not\subseteq$ **F** where **F** is the empty set expression $X \cap \neg X$ for an arbitrary set variable X. For each clause of the form $P_1(x) \vee \ldots \vee P_n(x) \vee Q_1(f(x)) \vee \ldots \vee Q_m(f(x))$ in C, where each P_i and Q_i is either a predicate or its negation, we include the constraint

$$f(\neg P_1 \cap \ldots \cap \neg P_n) \subseteq Q_1 \cup \ldots \cup Q_m \tag{4}$$

in $T[C]$. Finally, consider a clause in C of the form $P_1(x) \vee \ldots \vee P_n(x) \vee Q_1(y) \vee \ldots \vee Q_m(y)$ where each P_i and Q_i is either a predicate or its negation. Let g be a fixed but arbitrary binary Tarskian function symbol. For each such clause in C we include the constraint

$$g(\neg P_1 \cap \ldots \cap \neg P_n, \neg Q_1 \cap \ldots \cap \neg Q_m) \subseteq \mathbf{F} \tag{5}$$

in $T[C]$.

It is easy to show $T[C]$ and C are equisatisfiable, as follows. Suppose that C is satisfied by a first-order structure M. If we extend M by interpreting g as the constant function which maps all pairs of domain elements to the value of a then we get a model of $T[C]$ (since $T[C]$ has no set variables in it the choice of ρ is immaterial). Conversely, let $\langle M, \rho\rangle$ satisfy $T[C]$. It must be possible to extend M by interpreting a in such a way as to satisfy all clauses of the form $P(a)$ in C. This extension must satisfy C. ❑

3.3 Lower Bound for *RFC*-Tarskian Constraints

In this section we show that satisfiability for full Tarskian set constraints (with both recursion and arbitrary arity deterministic and nondeterministic operation symbols) is undecidable. The proof is by a reduction of Hilbert's tenth problem. The proof uses only set variables, constants, monadic functions, set unions and intersections (no complementation), and a single level of $\forall$ quantification.

Theorem 4: Satisfiability for *RFC*-Tarskian constraint sets is undecidable.

Proof: Let Σ be a set of constraints of the form $n = 1$, $n = p + q$, or $n = pq$, where n, p and q range over nonnegative integers. It follows from the undecidability of Hilbert's tenth problem that satisfiability for such systems of constraints is undecidable. We reduce the diophantine constraint set Σ to a set $T(\Sigma)$ of Tarskian set constraints as follows.

For each natural number variable n occurring in □ we introduce a set variable X_n with the intention that the cardinality of X_n represent the value of n. For set expressions C and W we will use $C = W$ as an abbreviation for the two constraints $C \square W$ and $W \square C$. We will also use $|C| \square |W|$ as an abbreviation for $C \square f(W)$ where f is a fresh monadic function symbol used only in this constraint. We will use $|C| = |W|$ as an abbreviation for $|C| \square |W|$ and $|W| \square |C|$. For any monadic function symbol s and set expression C we let $s^*(C)$ be an abbreviation for $\square W . C \square s(W)$, *i.e.*, the set of objects that can be gotten by applying s zero or more times to an element of C. For each variable n in □ we introduce a constant symbol c_n and monadic function symbol s_n and add the following constraints to $T(\square)$:

$$\begin{aligned} X_n &= s_n^*(c_n) \\ c_n &\square\ s_n(s_n^*(c_n)) \end{aligned} \tag{6}$$

The first constraint states that X_n is the set containing c_n and all its transitive successors under s_n. The second constraint states that c_n is the successor under s of some element of $s_n^*(c_n)$ and therefore that the set X_n forms a loop under the successor function s_n. This implies that X_n is a finite set but does not otherwise constrain its cardinality. This corresponds to the Hilbert problem constraint that n is a nonnegative integer. We now need to impose the constraints given in □.

If □ contains the constraint $n = 1$ then $T(\square)$ contains the constraint $X_n = c_n$. If □ contains $n = p + q$ then we add the constraints

$$\begin{aligned} X_n &= U \square W & \qquad |X_p| &= |U| \\ U \square W &\square \mathbf{F} & \qquad |X_q| &= |W| \end{aligned} \tag{7}$$

to $T(\square)$ where C, U, and W are fresh set variables and $\mathbf{F}$ is the set expression $\square X . X$, which always denotes the empty set. It remains only to express product constraints as set constraints.

To handle the product case we use the notation $\square x \square f^*(c)\ x = C[x]$ as an abbreviation for

$$f^*(c) = \square X . (c \square C[c]) \square (f(X) \square C[f(X)]) \tag{8}$$

For example, $\square x \square f^*\ x = g(f(x))$ states that g is the inverse of f on the set $f^*(c)$. More generally, if there is only one occurrence of X in $C[X]$, and $C[X]$ is constructed purely from X and function symbols, then the expression $\square x \square f^*(c)\ x = C[x]$ states that each element x of $f^*(c)$ is a "fixed point" of the set expression $C[\ ]$. Suppose Φ contains $n = pq$. We add the following constraints to $T(\Phi)$:

1. $X_n = f^*(c)$
2. $c \in f(f^*(c))$
3. $X_p = g^*(c)$
4. $c \in g(g^*(c))$
5. $X_q = h^*(c)$
6. $c \in h(h^*(c))$
7. $f^*(c) = \mu X.(c \cup g(X) \cup h(X))$
8. $\forall x \in f^*(c)\ x = g'(g(x))$
9. $\forall x \in f^*(c)\ x = h'(h(x))$
10. $\forall x \in f^*(c)\ x = g'(h'(g(h(x))))$
11. $g^*(c) \cap h^*(c) = c$

where c is a fresh constant and f, g, h, g' and h' are fresh monadic function symbols. Constraints 2, 4, and 6 imply that $f^*(c)$, $g^*(c)$, and $h^*(c)$ are all finite "loops". Constraint 7 implies that g and h are both functions mapping $f^*(c)$ into $f^*(c)$. Constraints 8, and 9 imply that g' and h' are inverses of g and h, respectively, on the set $f^*(c)$. Since both g and h are invertible they must both be bijections from $f^*(c)$ to itself. This implies that the inverses g' and h' are also bijections. Condition 10 implies that g and h commute on $f^*(c)$, *i.e.*, $f(g(x))$ equals $g(f(x))$. Now consider $g^n(h^*(c))$. Since g is bijective, g^n is bijective. Note that $h(g^n(x))$ equals $g^n(h(x))$. So the mapping g^n is a bijection which "preserves h structure". Hence the set $g^n(h^*(c))$ is an h-loop with the same cardinality as $h^*(c)$. Since sets of the form $g^j(h^*(c))$ are h-loops they are either equal or disjoint. Suppose $g^j(h^*(c)) = g^k(h^*(c))$. Applying g'^j to both sides we $h^*(c) = g^{k-j}(h^*(c))$. This implies that $g^{k-j}(c)$ must be in $h^*(c)$ and hence by condition 11 above we have $g^{k-j}(c) = c$. But this implies that k equals j mod $|g^*(c)|$. Hence for $k \neq j$ mod $|g^*(c)|$ we have $g^j(h^*(c))$ is disjoint from $g^k(h^*(c))$. Since all these sets are of size $|h^*(c)|$ we have $|f^*(c)| \geq |g^*(c)||h^*(c)|$. Condition 7 also implies that $f^*(c) \subseteq g^*(h^*(c))$ and with conditions 8 through 10 this gives $|f^*(c)| \leq |g^*(c)||h^*(c)|$ so we can conclude $|f^*(c)| = |g^*(c)||h^*(c)|$ as desired. ❑

4. Upper Bounds for Constraints without Recursion

In this section we present upper bounds for Tarskian set constraint satisfiability problems without recursion. We consider four restrictions on this problem and provide upper bounds for each — the variations are due to the separate prohibition or inclusion of function symbols and constant symbols.

The upper bound proofs build on one another conceptually, as we move from prohibiting function symbols and constants to allowing both. The proofs given establish the upper bound results shown in lines 1 through 4 of Table 1. The upper bounds are tight to the lower bounds proven in Section 3 for each case except the case allowing both function and constant symbols, where there is a gap between our lower and upper bounds.

4.1 Basic Terminology and Summary of Nonrecursive Upper Bound Proofs

Because the following four upper bound proofs all rely on the same basic terminology about nonrecursive set constraints, we collect the relevant definitions here for reference. Let □ be a set of nonrecursive Tarskian set constraints, *i.e.*, set constraints not involving □-sets.

Definition 6. We say that a set expression *C occurs in* □ if either *C* occurs as a top level set expression in a constraint in □ or as a subexpression of such an expression. A □*-type* is a set □ of set expressions satisfying the following conditions:

- Every element of □ is of the form C or $\neg C$ where C occurs in □,
- For any C occurring in □, □ contains exactly one of C or $\neg C$,
- If C_1 □ C_2 occurs in □ then □ contains C_1 □ C_2 if and only if □ contains at least one of C_1 and C_2, and
- If □ contains C □ W and □ contains C, then □ contains W.

Intuitively, the □-types correspond to the "types" of domain elements where two elements have the same type if they are not distinguished by any set expression occurring in □ — *i.e.,* both elements are included or excluded together in the denotation of every set expression occurring in □. The □-types are analogous to truth assignments on the Fischer-Ladner closure used in decision procedures for PDL (Fischer and Ladner, 1979). Now consider a model □M, □□.

Definition 7. A domain element x of M *inhabits* a □-type □ if for every C □ □ we have x □ $M[\![C]\!]$ □. If some domain element inhabits □ then □ is called *inhabited*.

Definition 8. The *type* of any domain element x of M is the unique □-type □ such that for all set expressions U occurring in □, □ contains U if and only if x □ $M[\![U]\!]$ □.

Next we define a notion of a "possible output" for an application of an operation symbol (deterministic or nondeterministic).

Definition 9. A type □ is a *possible output* of operation R applied to types □$_1, \ldots,$ □$_n$, written $R($□$_1, \ldots,$ □$_n) \twoheadrightarrow$ □, provided that for every expression of the form $R(C_1, \ldots, C_n)$ in □ such that each C_i □ □$_i$, □ contains $R(C_1, \ldots, C_n)$ as opposed to □$R(C_1, \ldots, C_n)$.

Intuitively, $R(\tau_1, \ldots, \tau_n) \twoheadrightarrow \tau$ is true provided that the set expressions in τ do not forbid R from mapping objects of type τ_i to an object of type τ. We now consider local consistency properties that must hold for the set of inhabited Σ-types in any model.

Definition 10. A set S of Σ-types is *locally consistent* if:

- For each negative constraint $U \not\subseteq W$ in Σ there is a type in S which contains U but not W,
- If a type τ in S contains $R(C_1, \ldots, C_n)$ for any n-ary operation symbol R, then there exist types $\tau_1, \ldots, \tau_n$ in S such that $R(\tau_1, \ldots, \tau_n) \twoheadrightarrow \tau$ and $C_i \in \tau_i$ for each i,
- For each deterministic operation symbol f and Σ-types $\tau_1, \ldots, \tau_n$ in S, where n is the arity of f, there exists a Σ-type τ in S such that $f(\tau_1, \ldots, \tau_n) \twoheadrightarrow \tau$, and
- Each constant (deterministic operation of arity zero) is contained in exactly one type in S.

If Σ does not contain recursion and either does not contain constants (deterministic operations of arity 0) or does not contain functions (deterministic operations of arity at least one) then Σ is satisfiable if and only if there exists a locally consistent set of Σ-types. The following subsections contain proofs of this fact, which we briefly summarize for each language variation here. In the case where neither constants or functions are present one can start with all Σ-types and iteratively remove those violating the second condition of Definition 10. We then have that Σ is satisfiable if and only if the resulting set of Σ-types satisfies the first condition of Definition 10. This gives a deterministic exponential time decision procedure. If constants are present then we must nondeterministically guess a unique Σ-type for each constant before removing Σ-types to satisfy the second condition of Definition 10. However, this involves only polynomially many nondeterministic choices and hence the space of all possible guesses can be searched in deterministic exponential time. So we again get a deterministic exponential time procedure. When functions are present (but not constants) we can again start with all Σ-types and remove types violating the second and third conditions. However when the third condition is violated we have a choice of removing any one of the types $\tau_1, \ldots, \tau_n$. This gives a nondeterministic exponential time procedure.

When both functions and constants are present, Tarskian set constraints can express a limited form of diophantine constraints on the cardinalities of the sets represented by the set expressions. For this case, we give a nondeterministic reduction to an exponentially large diophantine constraint problem of a form we call *prequadratic*, and then show that such constraint problems can be solved in nondeterministic time exponential in their size, yielding a nondeterministic doubly exponential decision procedure. We conjecture that prequadratic diophantine constraints are in NP — if this conjecture is true, our decision procedure would give a tight upper bound of nondeterministic exponential time for nonrecursive Tarskian constraints.

In the next four subsections we give these four upper bound proofs in detail.

4.2 Upper Bound for Plain Tarskian Constraints

This section gives a simple (deterministic) exponential time procedure for determining the satisfiability of a system of nonrecursive Tarskian set constraints without function or constant symbols. The results in this section are subsumed by those of Section 5 where we show that satisfiability for recursive Tarskian set constraints without functions is also decidable in deterministic exponential time. However, the procedure in Section 5 is based on the known decision procedure for the propositional μ-calculus which in turn is based on tree automaton techniques and is probably unusable in practice ()**CITATION?**. Here we sketch a much simpler exponential time procedure that is somewhat analogous to the exponential time decision procedure for PDL constructed by Pratt (Pratt, 1980).

Theorem 5: Satisfiability of a system of plain Tarskian constraint sets is decidable in exponential time.

Proof: We define a simple procedure. Let Σ be a system of nonrecursive Tarskian set constraints. Initialize S to be the set of all Σ-types. Now repeatedly remove from S any type τ such that τ contains $R(C_1, \ldots, C_n)$ but there are no types $\tau_1, \ldots, \tau_n$ in S such that each τ_i contains C_i and $R(\tau_1, \ldots, \tau_n) \twoheadrightarrow \tau$.

The final set S of Σ-types can be computed in time that is exponential in the number of set expressions occurring in Σ. We now show that Σ is satisfiable if and only if for all negative constraints $C \not\subseteq W$ in Σ there exists some type τ in S such that τ contains C but not W. First suppose there is some negative constraint $C \not\subseteq W$ in Σ such that every type in S that contains C also contains W. It is not difficult to show that every Σ-type removed by the above procedure must be uninhabited in any model of Σ, implying that every model of Σ must satisfy $C \subseteq W$ (because any domain element x in the denotation of C has C in its Σ-type and must then have W in its Σ-type and be in the denotation of W). Since Σ contains $C \not\subseteq W$ there can be no such models and so Σ is unsatisfiable.

Now, to prove the converse, suppose that for each negative constraint $C \not\subseteq W$ in Σ there exists a type in S containing C but not W. Let $\langle M, \rho\rangle$ be the model such that the domain of M is the set S, $\rho(X)$ is the set of types in S containing the variable X and $M(R)$ for operation symbol R is the relation containing those tuples $\langle\tau_1, \ldots, \tau_n, \tau\rangle$ over S such that $R(\tau_1, \ldots, \tau_n) \twoheadrightarrow \tau$. We prove by structural induction on set expressions that for any set expression C occurring in Σ we have that $M[\![C]\!]\rho$ is exactly the set of types in S containing C. It is not difficult to show using the properties of S listed in definitions 6 and 10 that this implies that $\langle M, \rho\rangle$ satisfies Σ. The case of set variables is true by definition.

The case of union set expressions that occur in □ is straightforward given the third property of □-types from Definition 6. If $\neg C$ occurs in □ then by the induction hypothesis $M[\![C]\!]$ □ is the set of types in S containing C. Since by definition every □-type contains exactly one of C and $\neg C$, we have that $M[\![□C]\!]$ □, which equals $S - M[\![C]\!]$ □, is precisely those types in S which contain $\neg C$, as desired. Now consider an application expression $R(C_1, \ldots, C_n)$. Let □ be a type in S containing this application expression. Since □ was not removed in the process of constructing S there must exist types $□_1, \ldots, □_n$ such that $R(□_1, \ldots, □_n) \twoheadrightarrow □$ and where C_i □ $□_i$ for each C_i. It follows from the definition of $M(R)$ that □ is a member of $M[\![R(C_1, \ldots, C_n)]\!]$ □. Finally suppose that □ is a type in $M[\![R(C_1, \ldots, C_n)]\!]$. By the definition of $M(R)$ there must exist types $□_1, \ldots, □_n$ in $M[\![C_1]\!]$ □, …, $M[\![C_n]\!]$ □ respectively such that $R(□_1, \ldots, □_n) \twoheadrightarrow □$. By the induction hypothesis we must have C_i □ $□_i$ for each i. Now by the definition of $\twoheadrightarrow$ we must also have that □ contains $R(C_1, \ldots, C_n)$.
❏

The proof of the above theorem yields a finite model property for plain Tarskian set constraints (every satisfiable set of constraints is satisfiable by a finite model). It is also possible, although we will not do it here, to give a simple set of inference rules and show that the steps of the above procedure can be simulated by inferences in that system and hence that the rules are sound and complete for this case.

4.3 Upper Bound for C-Tarskian Constraints

In this section we sketch a proof that satisfiability for Tarskian set constraints without recursion and without function symbols of arity greater than zero is decidable in exponential time. The decision procedure is very similar to the procedure of the preceding section except that, due to the inclusion of constant symbols in the language, rather than deterministically construct a set of □-types we are forced to guess a set of □-types, making only polynomially many guesses. The guessing is needed because we must ensure that we select a single unique □-type for each constant, even though there may be many ways to make these choices to build a satisfying model.

Theorem 6: Satisfiability for C-Tarskian constraint sets is decidable in exponential time.

Proof: We give a nondeterministic exponential time procedure involving only polynomially many nondeterministic choices. Any such procedure can be run in deterministic exponential time. First we guess a □-type for each constant symbol that appears in □. Note that this can be done with quadratically many nondeterministic choices (for each constant in □, and each set expression in □, we need to decide whether that set expression is in the □-type associated with the constant). Now initialize S to be all □-types not containing any constants (*i.e.*, types containing $\neg c$ for each constant symbol c occurring in □) plus the selected

□-types for the constants. Then, as in the previous section, repeatedly remove from S any type □ such that □ contains $R(C_1, \ldots, C_n)$ but there are no types $\square_1, \ldots, \square_n$ in S such that each $\square_i$ contains C_i and $R(\square_1, \ldots, \square_n) \twoheadrightarrow \square$. Accept □ as satisfiable if the removal process does not remove any of the types containing constants and the final set S contains a type for each negative constraint $C \not\square W$ in □ that contains C but not W.

The proof of correctness of this procedure is a straightforward modification of the proof of the preceding section. Every type eliminated by the procedure must be uninhabited in any model corresponding to the choice of types for constants. Therefore, if the procedure does not accept □ then □ must be unsatisfiable. On the other hand if the procedure does accept □ then the final set S of types constructed by the procedure can serve as the domain of a model of □. ❑

4.4 Upper Bound for *F*-Tarskian Constraints

In this section we show that satisfiability for nonrecursive Tarskian set constraints without constant symbols but including function symbols can be decided in nondeterministic exponential time. We start by proving the following lemma which states that when no constant symbols are involved it suffices to consider nonstandard models in which function symbols are allowed to denote “total” relations.

Definition 11. Let S be a subset of D^{n+1}. The relation S is called *total* if for every tuple $\square x_1, \ldots, x_n \square\square D^n$ there exists some $y \square D$ such that $\square x_1, \ldots, x_n, y \square\square S$. A *nonstandard model* is a model $\square M, \square\square$ such that for each function symbol f we have that $M(f)$ is a total (but not necessarily functional) relation.

The meaning of a set expression in a nonstandard model is defined in an identical manner to its meaning in standard models. We can now state the following lemma for Tarskian set constraints without constant symbols. Note that the following lemma applies to recursive as well as nonrecursive constraints.

Lemma 3: If □ is a set of *RF*-Tarskian constraints then □ is satisfiable if and only if □ is satisfied by a nonstandard model.

Proof: Since every model is a nonstandard model the only if direction is trivial. Now suppose that there is a nonstandard model $\square M, \square\square$ which satisfies □. Let D be the domain of M. By the Löwenheim-Skolem theorem for first order logic we can assume without loss of generality that D is countable. We will construct a standard model satisfying □ whose domain is $D \square \square$, *i.e.*, the set of pairs $\square x, i \square$ for $x \square D$ and i a natural number. We define a *target constraint* to be a tuple of the form $\square f, x_1, \ldots, x_n, \square y, k \square\square$ where f is a function symbol occurring in □, each x_i and y is in D, k is a natural number, and $\square x_1, \ldots, x_n, y \square\square M(f)$. Intuitively, a target constraint states that f should map the tuple $\square x_1, \ldots, x_n \square$ to the pair $\langle y, k\rangle$. We define a *domain constraint* to be a tuple of the form $\langle f, \langle x_1, j_1\rangle, \ldots, \langle x_n, j_n\rangle\rangle$. Intuitively a domain constraint expresses the constraint that f must be defined on the tuple $\langle\langle x_1, j_1\rangle, \ldots, \langle x_n, j_n\rangle\rangle$. The set of all possible target and domain constraints is itself countable. Let $E_1, E_2, \ldots$ be an enumeration of all target and domain constraints (each target and domain constraint appears somewhere in this enumeration). We now define a series of models $\langle M_1, \rho_1\rangle$, $\langle M_2, \rho_2\rangle$, $\langle M_3, \rho_3\rangle$, … , all of which have domain $D \times \mathbb{N}$. The model $\langle M_1, \rho_1\rangle$ is defined by the following conditions:

- For any set variable X, $\rho_1(X)$ is the set of pairs of the form $\langle x, i\rangle$ where $x \in \rho(X)$,
- For any nondeterministic operation symbol R occurring in Σ and any values of $j_1, \ldots, j_n$, and k, $\langle\langle x_1, j_1\rangle, \ldots, \langle x_n, j_n\rangle, \langle y, k\rangle\rangle \in M_1(R)$ if and only if $\langle x_1, \ldots, x_n, y\rangle \in M(R)$,
- For any function symbol f occurring in Σ, $M_1(f)$ is empty.

Each model M_{i+1} is defined in terms of M_i and the constraint E_i. If E_i is a target constraint $\langle f, x_1, \ldots, x_n, \langle y, k\rangle\rangle$ then we let M_{i+1} be identical to M_i except that $M_{i+1}(f)$ contains exactly one more tuple than $M_i(f)$ and that tuple is $\langle\langle x_1, j_1\rangle, \ldots, \langle x_n, j_n\rangle, \langle y, k\rangle\rangle$ where $j_1, \ldots, j_n$ are the lexicographically least sequence of numbers such that $M_i(f)$ does not contain any tuple of the form $\langle\langle x_1, j_1\rangle, \ldots, \langle x_n, j_n\rangle, z\rangle$. If E_i is a domain constraint of the form $\langle f, \langle x_1, j_1\rangle, \ldots, \langle x_n, j_n\rangle\rangle$ then if $M_i(f)$ contains a tuple of the form $\langle\langle x_1, j_1\rangle, \ldots, \langle x_n, j_n\rangle, z\rangle$ then M_{i+1} equals M_i, otherwise $M_{i+1}(f)$ contains one more tuple than $M_i(f)$ and that tuple is $\langle\langle x_1, j_1\rangle, \ldots, \langle x_n, j_n\rangle, \langle y, 0\rangle\rangle$ where $\langle x_1, \ldots, x_n, y\rangle \in M(f)$. Such a y must exist because $M(f)$ is total.

We let $\langle M', \rho'\rangle$ be the model such that $M'(R)$ is the union of all relations of the form $M_i(R)$ for any (deterministic or nondeterministic) operation symbol R. It is not difficult to verify that for any function symbol f, $M'(f)$ is functional. Hence M' is a standard model. Furthermore, one can also verify by structural induction on set expressions that for any nonrecursive set expression C not involving constants we have that $M'[\![C]\!]\rho'$ is the set of pairs $\langle x, i\rangle$ such that $x \in M[\![C]\!]\rho$. This implies that $\langle M', \rho'\rangle$ satisfies Σ. ❑

We now show that a set Σ of nonrecursive Tarskian set constraints without constants is satisfiable if and only if there exists a set of Σ-types satisfying the easily checked conditions of local consistency (see Definition 10). This gives a simple nondeterministic exponential time procedure which simply guesses an appropriate set of Σ-types.

Lemma 4: Let Σ be a set of F-Tarskian set constraints. Σ is satisfiable if and only if there exists a locally consistent set of Σ-types.

Proof: (sketch) The proof is very similar to the proofs in the preceding two sections. If Σ is satisfiable then let $\langle M, \rho\rangle$ be a model of Σ and take S to be the set of types τ which are inhabited in $\langle M, \rho\rangle$. It is not difficult to show that S is locally consistent. Conversely, let S be a set of types satisfying the above conditions. Let $\langle M, \rho\rangle$ be the nonstandard model whose domain is the set S of types and where the set variables and operation symbols are interpreted as follows. As in the earlier proofs interpret each set variable and monadic relation symbol as the set of types in S that contain that symbol. We interpret each operation symbol R so that $M(R)$ is the relation containing those tuples $\langle\tau_1, \ldots, \tau_n, \sigma\rangle$ such that $R(\tau_1, \ldots, \tau_n) \twoheadrightarrow \sigma$. The argument given in Section 4.2 can be used to show that for each type $\tau \in S$ and each set expression C occurring in Σ we have that $\tau \in M[\![C]\!]\rho$ if and only if $C \in \tau$. This can be used to show that $\langle M, \rho\rangle$ is a model of Σ. ❑

This lemma leads to a procedure that is quite similar to the procedures of Section 4.2 and Section 4.3; however, this procedure requires a potentially exponential degree of nondeterminism. We can construct the set S of types by initializing it to be all types and then eliminating those types which are provably empty in any model. A problem arises however when attempting to satisfy the third condition in the definition of locally consistent (Definition 10): that any function symbol and tuple of types from S have a corresponding possible output type also in S. Suppose that there is a function symbol f and Σ-types $\tau_1, \ldots, \tau_n$ in S, where n is the arity of f, such that there is no $\sigma \in S$ such that $f(\tau_1, \ldots, \tau_n) \twoheadrightarrow \sigma$. In this case one of the τ_i must be removed from S but we are left with a choice of which τ_i to remove. We must nondeterministically explore the possible choices because some may lead to a locally consistent S while others fail to do so. This can lead to exponentially many choice points.

4.5 Upper Bound for *FC*-Tarskian Constraints

We now consider the problem of determining the satisfiability of nonrecursive Tarskian set constraints allowing both functions and constants. The procedure presented in this section nondeterministically reduces set constraint satisfiability to a system of diophantine constraints. The first step of the procedure guesses a locally consistent set of inhabited Σ-types as defined in Section 4.1. The procedure then constructs a system of diophantine inequalities describing cardinality constraints. In order to describe the cardinality constraints we need some additional terminology.

When both functions and constants are present, it is not sufficient to find a locally consistent set of types. Consider the constraints $c_1 \cup c_2 \subseteq f(c_3)$ and $c_1 \neq c_2$ where c_1, c_2, c_3, and f are all deterministic. This constraint set has a locally consistent set of types but it is not satisfiable because $f(c_3)$ must be a singleton set while $c_1 \cup c_2$ must contain two elements. In addition to finding a locally consistent set of types, we must also check that cardinality constraints on the selected set of types can be met.

Consider an element x which inhabits a □-type □ in a model $\square M, \square\square$. For each application expression $R(C_1, \ldots, C_n)$ in □ there must exist $y_1, \ldots, y_n$ in $M[\![C_1]\!]\square, \ldots, M[\![C_n]\!]\square$, respectively, such that $\square y_1, \ldots, y_n, x\square\square M(R)$ and such that $y_i \square M[\![C_i]\!]\square$ for each i. The values $y_1, \ldots, y_n$ can be viewed as "predecessors" of x which "justify" the fact that x is in the set $R(C_1, \ldots, C_n)$. Suppose the predecessors $y_1, \ldots, y_n$ each have □-types $\square_1, \ldots, \square_n$. The element x can be viewed as inhabiting the image of $\square_1, \ldots, \square_n$ under f. These observations lead to the following definitions.

Definition 12. Let S be a set of □-types. A *range expression* is an expression of the form $f(\square_1, \ldots, \square_n)$, where $\square_1, \ldots, \square_n$ are □-types in S and f is a function symbol appearing in □. We say that a domain element d of a model $\square M, \square\square$ *inhabits* a range expression $f(\square_1, \ldots, \square_n)$ if there are some domain elements $d_1, \ldots, d_n$ inhabiting □-types $\square_1, \ldots, \square_n$ in $\square M, \square\square$, respectively, such that $\square d_1, \ldots, d_n, d\square\square M(f)$. We will say that a range expression is *inhabited* in $\square M, \square\square$ when some element of the domain of M inhabits that expression.

Simply writing and solving inequality constraints on the cardinalities of the sets of inhabitants of the □-types and the range expressions is still not enough to force the existence of a model. To understand why consider the constraints $c_4 \not\square c_5$, $c_4 \square c_5 \square f(c_1 \square c_2)$, and $c_4 \square c_5 \square f(c_1 \square c_3)$. These constraints are satisfiable but in any model we will have that $f(c_2) = f(c_3)$. Adding the constraint $c_4 \square c_5 \square f(c_2 \square c_3)$ makes the constraint set unsatisfiable, even though the natural local cardinality constraints on the range expressions are all satisfied. There exist locally consistent sets of □-types for all four constraints. The □-types in these sets appear consistent even if cardinality constraints on □-types and range expressions are considered (the natural local cardinality constraints, similar to those given below, are satisfiable). Furthermore, each individual constraint of the form $U \square V$ appears consistent with the cardinalities. This problem forces us to explicitly allocate predecessors for each □-type, as follows.

Definition 13. A *predecessor justification* for a function application expression $f(C_1, \ldots, C_n)$ is a range expression $f(\square_1, \ldots, \square_n)$ such that each $C_i \square \square_i$. A □-*predecessor-type* is a pair $\square\square, \square\square$ where □ is a □-type and □ is a mapping from function applications appearing in the type □ to range expressions such that for any function application U in □ we have that $\square(U)$ is a predecessor justification of U and $\square(U) \twoheadrightarrow \square$. An object x inhabits a □-predecessor-type $\square\square, \square\square$ in a model M if x inhabits □ and, for each application expression U in □, x inhabits $\square(U)$.

An application expression $f(C_1, \ldots, C_n)$ can often be justified in more than one way, *i.e.*, many different predecessors of many different □-types can simultaneously explain the presence of x in the set $M[\![f(C_1, \ldots, C_n)]\!]\square$. However, for each element x in a model of □ there will be at least one □-predecessor-type □□, □□ inhabited by x. We will assume that some choice function is provided with each model □M, □□ so that for each element x of the domain of M we can choose a unique □-predecessor-type for x.

Note that the number of □-types is at most $2^{|□|}$. The number of justification functions □ is no greater than the number of functions from expressions in □ to range expressions. We can assume without loss of generality that no application expressions involve more than two arguments (larger arities can be eliminated with the use of a pairing function). Under this assumption there are at most $|□|2^{2|□|}$ range expressions. Hence the number of justification functions □ is no more than $(|□|2^{2|□|})^{|□|}$, which is $2^{2|□|^2 + |□|\log|□|}$ and hence is exponential in $|□|$. This implies that the number of □-predecessor-types is also exponential in $|□|$.

Definition 14. Let S be a set of □-types. For each type □ □ S let $z_□$ be a variable representing the number of inhabitants of □. For each range expression $f(□_1, \ldots, □_n)$ with each $□_i$ □ S and f appearing in □, let $z_{f(□_1, \ldots, □_n)}$ be a variable representing the number of domain members inhabiting $f(□_1, \ldots, □_n)$. For each □-predecessor-type □□, □□ let $z_{□□, □□}$ be a variable representing the number of individuals whose selected predecessor type is □□, □□. We define $D(S)$ to be the system of diophantine constraints including the following constraints:

Habitation: $z_□$ □ 1

Constants: $z_□ = 1$ when c □ □ for constant c

Type: $z_□ = □_{□□, □□} z_{□□, □□}$

Range: $z_{f(□_1, \ldots, □_n)}$ □ $□_{□□, □□ \text{ such that } (□U □ □ . □(U) = f(□_1, \ldots, □_n))} z_{□□, □□}$

Predecessor: $z_{f(□_1, \ldots, □_n)}$ □ $□_i z_{□_i}$ for deterministic f only

We now come to the first main theorem of this section.

Theorem 7: A set □ of FC-Tarskian constraints is satisfiable if and only if there exists a locally consistent set S of □-types such that the constraint set $D(S)$ is satisfiable over the positive integers plus □.

Proof: First suppose that □ is satisfied by a model □M, □□. Let S be the set of □-types inhabited by M. It is easy to check that S is locally consistent. Now select for each element x in the domain of M a □-predecessor-type which it inhabits. We interpret the cardinality variable $z_{□□, □□}$ to be the cardinality of the set of elements whose selected □-predecessor-type is □□, □□. If this cardinality is infinite then we assign $z_{\square\square, \square\square}$ the special value □ and ignore the order of infinity of the actual cardinality. Likewise, we interpret $z_{\square}$ as the number of inhabitants of the type □, for each □ □ S (again using □ for any infinite type), and $z_{f(\square_1, \ldots, \square_n)}$ as the number of inhabitants for the range expression $f(\square_1, \ldots, \square_n)$. It is not difficult to check that all the constraints in $D(S)$ are satisfied under this interpretation.

Now, to show the converse direction, suppose that there exists a locally consistent set S of □-types such that $D(S)$ is satisfiable over positive integers plus □. Consider a particular assignment of natural numbers and □ to the variables such that $D(S)$ is satisfied. We will build a model □M, □□ of □ based on this assignment. The domain of our model will be the union over all inhabited types □ of sets $\{ \square\square, i\square : 1 \square i \square z_{\square} \}$ where for each type □ and each index i we assume that $\square\square, i\square$ is a distinct object. If $z_{\square}$ is □ then we include in the domain the object $\square\square, i\square$ for each natural number $i \square 1$, so that there will be a countably infinite number of distinct elements of the form $\square\square, i\square$. We call □ the *base type* of $\square\square, i\square$.

We define □ on each set variable X to be the set of all domain elements $\square\square, i\square$ whose base type □ contains X. We define $M(c)$ for each constant symbol c to be $\square\square, 1\square$ for the unique □ □ S containing c. We define $M(R)$ for each nondeterministic operation symbol R be the set of all tuples $\square\square\square_1, i_1\square, \ldots, \square\square_n, i_n\square, \square\square, j\square\square$ such that $R(\square_1, \ldots, \square_n) \twoheadrightarrow \square$.

Finally, we define M on the function symbols. We have from the constraints $D(S)$ that for every □-type □, $z_{\square}$ is the sum of all $z_{\square\square, \square\square}$. Using this fact, we partition the values $\square\square, i\square$ into sets $\square_{\square}$, one for every □-type □ and every □ such that $z_{\square\square, \square\square}$ is nonzero such that each $\square_{\square}$ has cardinality $z_{\square\square, \square\square}$. To define M on the n-ary function symbol f consider an n-tuple $\square\square_1, \ldots, \square_n\square$ of □-types in S. For each such n-tuple, we define f on all domain tuples of the form $\square\square\square_1, i_1\square, \ldots, \square\square_n, i_n\square\square$ as follows. Let *Dom* be the set of all such domain tuples (for a fixed tuple $\square\square_1, \ldots, \square_n\square$) and let *Range* be the set of all domain elements that are members of any set $\square_{\square}$ such that $\square(U) = f(\square_1, \ldots, \square_n)$ for some $U \square \square$. Let □ be some □-type in S such that $f(\square_1, \ldots, \square_n) \twoheadrightarrow \square$; we call □ the "default value" for f on $\square_1, \ldots, \square_n$ (□ exists due to the local consistency of S). Both *Dom* and *Range* are countable sets so both can be enumerated. Now define f to map each element of *Dom* to the corresponding element of *Range* under the given enumeration. The type and range constraints in $D(S)$ ensure that the cardinality of *Range* is no larger than the cardinality of *Dom*, so every element of *Range* will be the image of some element of *Dom*. If for some tuple in *Dom* there is no corresponding element of *Range* (because *Dom* has larger cardinality than *Range*), then f maps that tuple to $\square\square, 1\square$.

To prove □M, □□ satisfies □, we first show, by structural induction on a set expression E, that if E occurs in □ then $M[\![E]\!]\square$ is exactly the set of domain elements of the form $\square\square, i\square$ such that □ contains E. If E is a set variable or a constant symbol then this follows directly from the definition of $\square M, \square\square$ and from the "constants" constraint in $D(S)$. Suppose for induction that E is the negation of an expression E_1 for which the result holds. We show that the result holds for E. Our induction hypothesis tells us that $M\llbracket E_1 \rrbracket \square$ is the set of domain elements $\square\square, i\square$ whose base $\square$-type $\square$ contains E_1 — then of course $M\llbracket E \rrbracket \square$ is by definition the complement of this set, *i.e.*, the set of domain elements whose base types does not contain E_1. The definition of $\square$-type ensures that every $\square$-type contains exactly one of E_1 or $\neg E_1$, so the complement of the domain elements whose type contains E_1 is exactly the set of domain elements whose type contains $\neg E_1$, as desired.

Now we suppose for induction that E is the union of two expressions E_1 and E_2 for which the result holds, and show that the result holds for E. Our induction hypothesis guarantees that $M\llbracket E_1 \rrbracket \square$ (and, respectively, $M\llbracket E_2 \rrbracket \square$) is just the set of all domain members $\square\square, i\square$ whose base type $\square$ contains E_1 (respectively, E_2). So $M\llbracket E_1 \square E_2 \rrbracket \square$ is the set of all domain members whose base type contains either E_1 or E_2. The definition of a $\square$-type ensures that this is exactly the set of all domain members whose base type contains $E_1 \square E_2$, as desired.

Now suppose for induction that E is the application $R(E_1, \ldots, E_n)$ for n-ary nondeterministic operation R and expressions $E_1, \ldots E_n$ for which the result holds. We first show the forward direction — that every domain element whose base type contains $R(E_1, \ldots, E_n)$ is also in $M\llbracket R(E_1, \ldots, E_n) \rrbracket$. Consider a domain element $\square\square, k\square$ where $\square$ contains $R(E_1, \ldots, E_n)$. Since the set S is locally consistent there must exist $\square$-types $\square_1, \ldots, \square_n$ in S containing the $E_1, \ldots, E_n$, respectively, such that $R(\square_1, \ldots, \square_n) \twoheadrightarrow \square$. But then by the definition of $\square M, \square\square$, $M(R)$ maps $\square\square\square_1, 1\square \ldots, \square\square_n, 1\square\square$ to $\square\square, k\square$. But by our induction hypothesis, each element $\square\square_i, 1\square$ must be in the corresponding $M\llbracket E_i \rrbracket \square$, so $\square\square, k\square$ must be in $M\llbracket R(E_1, \ldots, E_n) \rrbracket \square$, as desired. For the reverse direction, suppose that a domain element $\square\square, k\square$ is in $M\llbracket R(E_1, \ldots, E_n) \rrbracket \square$. This implies that there must be domain elements $\square\square_1, i_1\square, \ldots, \square\square_n, i_n\square$ in $M\llbracket E_1 \rrbracket \square, \ldots, M\llbracket E_n \rrbracket \square$, respectively, such that $R(\square_1, \ldots, \square_n) \twoheadrightarrow \square$. But, by our induction hypothesis, since each $\square\square_j, i_j\square$ is in $M\llbracket E_j \rrbracket \square$, we have that each $\square_j$ contains E_j. Since $R(\square_1, \ldots, \square_n) \twoheadrightarrow \square$, the type $\square$ then must contain $R(E_1, \ldots, E_n)$, as desired.

The final case to consider is when E is the application $f(E_1, \ldots, E_n)$ of an n-ary function symbol f to n expressions $E_1, \ldots, E_n$ for which the result holds. Again, we first consider one direction — we suppose $\square\square, k\square$ is a domain element in $M\llbracket f(E_1, \ldots, E_n) \rrbracket \square$ and show that $\square$ contains $f(E_1, \ldots, E_n)$. Since $\square\square, k\square$ is in $M\llbracket f(E_1, \ldots, E_n) \rrbracket \square$, $\square\square, k\square$ must be the image under $M(f)$ of some tuple of domain elements $\square\square\square_1, i_1\square \ldots, \square\square_n, i_n\square\square$ which are members, respectively, of $M\llbracket E_1 \rrbracket \square, \ldots, M\llbracket E_n \rrbracket \square$. Our induction hypothesis then implies that for each j, E_j is a member of $\square_j$. When $M(f)$ was defined for tuples of the form $\square\square\square_1, k_1\square \ldots, \square\square_n, k_n\square\square$, the type $\square\square, k\square$ must have been in the range enumeration (or □□, k□ would not be the image under $M(f)$ of □□□$_1$, i_1□ …, □□$_n$, i_n□□). There are two ways that this can happen. First, □□, k□ could be an element of □$_□$ for some □ containing f(□$_1$, …, □$_n$) in its range. Second, □ could be the "default value" for f on □$_1$, … , □$_n$. In either case we have f(□$_1$, …, □$_n$) $\twoheadrightarrow$ □ . But then, since each E_i is in the corresponding □$_i$, $f(E_1, …, E_n)$ must be in □ as desired (by the definition of $\twoheadrightarrow$).

It remains to show the reverse direction: we take an arbitrary domain element □□, k□ such that □ contains $f(E_1, …, E_n)$ and show that □□, k□ is a member of $M[\![f(E_1, …, E_n)]\!]$ □ . Since □ □ S the habitation constraint in $D(S)$ guarantees that $z_□$ is nonzero. So, by the type constraint in $D(S)$, some $z_{□□, □□}$ must also be nonzero, and so for some □ the set □$_□$ is nonempty. Since □ contains $f(E_1, …, E_n)$, □($f(E_1, …, E_n)$) must be some range expression f(□$_1$, …, □$_n$) such that each □$_i$ contains E_i and f(□$_1$, …, □$_n$) $\twoheadrightarrow$ □ . But then, for each 1 □ i □ n, by our inductive hypothesis about E_i every domain element of base type □$_i$ must belong to $M[\![E_i]\!]$ □. Therefore, for every tuple □$x_1, …, x_n$□ of domain elements in the domain enumeration for f on □$_1$, … , □$_n$, we have x_i □ $M[\![E_i]\!]$ □ for each i; this together with the definition of $M(f)$ implies that the entire range enumeration for the function f on □$_1$, … , □$_n$ must be contained in $M[\![f(E_1, …, E_n)]\!]$ □ . But this range enumeration must include □□, k□ since it is a member of □$_□$ and □ contains f(□$_1$, …, □$_n$) in its range.

We have now shown the property that for each E occurring in □, $M[\![E]\!]$ □ is exactly the set { □□, i□| 1 □ i □ $z_□$, □ □ S, E □ □} . It is easy to show from this that □M, □□ satisfies □, as follows. Given a positive constraint U □ W in □ and an element □□, i□ of $M[\![U]\!]$ □, □□, i□ must also be in $M[\![W]\!]$ □ due to the above property along with the fact that U □ □ implies W □ □ by the definition of □-type. Finally, given a negative constraint U □ W in □, by the definition of locally consistent (Definition 10) there must be some □ in S such that U is in □ and W is not in □. It follows that □□, 1□□ $M[\![U]\!]$ □ and □□, 1□□ $M[\![W]\!]$ □ and so □M, □□ satisfies U □ W as desired. ❑

The above theorem shows that satisfiability for FC-Tarskian constraints can be reduced in nondeterministic exponential time to the satisfiability of an exponentially larger system of diophantine constraints where the variables range over positive integers plus □. We can eliminate □ by nondeterministically guessing which variables are infinite and folding this guess into the constraints. More specifically, break any equality statements into two inequalities, and then we substitute □ in for the variables we guess to be infinite and check that in any inequality with an infinity in the lesser side there is also an infinity in the greater side. We then remove all inequalities involving infinity. This produces a set of diophantine constraints over positive integers. These constraints have the following form.

Definition 15. A set of Diophantine inequalities:

$$\{p_i(x_1, …, x_n) \,□\, q_i(x_1, …, x_n) : 1 \,□\, i \,□\, m\}$$

between polynomials p_i and q_i over nonnegative integer variables $x_1, \ldots, x_n$ is *prequadratic* if every p_i is linear, and every q_i is either linear or is a product of variables.

The size of the prequadratic system of diophantine constraints $D(S)$ generated by a set □ of nonrecursive Tarskian constraints for any set S of □-types is exponential in $|□|$. The following theorem shows that satisfiability of prequadratic diophantine constraints can be determined in nondeterministic exponential time. We conjecture that satisfiability of prequadratic diophantine constraints is actually in NP.

Theorem 8: (Prequadratic Decidability Theorem) The problem of determining the satisfiability of a prequadratic set of Diophantine inequalities is solvable in nondeterministic exponential time.

Proof: Consider a prequadratic set of m Diophantine inequalities over n variables where the largest constant that appears has b bits. Each inequality is either linear or has a right hand side that is the product of two variables.

Without affecting the satisfiability of the entire set, we can replace each linear inequality $p_i(x_1, \ldots, x_n)$ □ $q_i(x_1, \ldots, x_n)$ with the equation $p_i(x_1, \ldots, x_n) + y_i = q_i(x_1, \ldots, x_n)$ where the new variables y_i introduced play the role of "slack" variables. Renaming the variables into a vector $x = x_1, \ldots, x_k$ we can then write the resulting problem in matrix form as $Ax = B$.

Call a variable x_i *bounded* in $Ax = B$ if there exists a finite upper bound on the value of x_i over all *rational* solutions to $Ax = B$. We can use linear programming (over the rationals) to determine which variables are bounded using the fact that a variable is bounded if and only if there is a solution to $Ax = 0$ over the rationals where that variable is nonzero. An analysis of the maximum possible upper bound that can be imposed by a system of linear constraints shows that the binary representation of the value of a bounded variable can contain at most $O(bn\log n)$ bits ()**CITATION?**. Our nondeterministic procedure can now guess the values of the bounded variables. We can then replace each bounded variable by the guessed value giving a simplified problem. In substituting in the guesses, some of the nonlinear constraints become linear and must be added to the resulting linear sub-problem, yielding new linear and nonlinear subproblems in fewer variables[2]. We repeat this process until either all variables have been guessed or all the variables in the resulting linear problem are unbounded. If at some point the guesses lead to a linear problem which is unsolvable over the rationals then all variables are bounded and values for them are guessed (of course in practice the procedure would simply terminate when such an inconsistency is found). Once all variables in the remaining linear prob-

2. New slack variables must be added in this case to convert the nonlinear inequalities to equations, but over the entire process at most m slack variables are introduced, one per original inequality.

lem are unbounded we determine the solvability of $Ax = B$ over the nonnegative integers. At this point we call the remaining linear problem the *residual linear problem*, and the remaining nonlinear problem the *residual nonlinear problem*. If the residual linear problem $Ax = B$ is solvable over the nonnegative integers then we accept the original prequadratic problem as solvable. Otherwise we fail.

We must show that this procedure is correct and that it terminates in nondeterministic exponential time. First we consider the running time analysis. The number of bits in the bounded variables can grow at most exponentially in the number of iterations of the procedure. One can show that after k variables are guessed the largest constant in the residual linear problem has at most $O((cn\log n)^{k+1})$ bits for some constant c. Since the number of variables guessed is bounded by n, we get an exponential upper bound on the size of the numbers appearing in the sequence of linear problems examined by the procedure.

Since all the linear programming operations required over the rationals can be completed in polynomial time relative to their input size, and their input is at most exponential in size relative to the original prequadratic problem size, the linear programming involved in the above procedure can be completed in exponential time. Moreover, the above procedure must guess values for at most linearly many variables (those in the original problem after slack variables are added), and the largest value guessed involves exponentially many bits; therefore, there are at most exponentially many bits guessed by the procedure. Finally, the residual linear problem (over the unbounded variables) can be solved over the integers in NP. Combining the complexities of these parts, we get a nondeterministic exponential running time for the procedure.

We now show the correctness of the procedure; *i.e.,* that if the procedure accepts a prequadratic system of constraints then that constraint set is solvable (the converse is straightforward). If the procedure accepts then there exists a residual linear problem $Ax = B$ solvable over the nonnegative integers, and where each variable is unbounded, plus a residual set of nonlinear constraints. It suffices to show that this residual prequadratic problem is solvable over the nonnegative integers. Since the procedure accepts, there exists a nonnegative integer solution □ to $Ax = B$. It is a fact of linear programming that if all variables are unbounded then there must be a nonnegative rational solution □ to $Ax = 0$ such that all components of □ are nonzero. We can assume without loss of generality that □ is integral because any nonintegral □ can be made integral by multiplying by an appropriate constant. The vector $□ + c□$ is a solution to $Ax = B$ for any c. For sufficiently large c this vector also satisfies all nonlinear constraints because the nonlinear expressions must eventually grow faster than the linear expressions. Thus if the procedure accepts then the residual prequadratic problem is satisfiable and so is the original problem. ❑

Finally, we can combine the above results to get the following.

Theorem 9: Satisfiability of FC-Tarskian constraint sets is decidable in nondeterministic doubly exponential time.

Note that if our conjecture holds, that prequadratic Diophantine constraint satisfiability is in NP, then we would get a tight upper bound here of nondeterministic exponential time. Without this conjecture, there is an exponential gap between our lower and upper bounds for this problem.

5. Upper Bound for *R*-Tarskian Constraints

In this section we consider Tarskian set constraints with recursive set expressions but without deterministic operation symbols of any arity. Constraint set satisfiability in this calculus turns out to be linear time equivalent to set expression satisfiability in the modal □-calculus. Here we give a linear time reduction from Tarskian constraint set satisfiability without determinism to set expression satisfiability in a calculus we call the Herbrand □-calculus. The Herbrand □-calculus is known to be decidable in exponential time. ()

To assist in our proofs about recursive expressions, we introduce syntactically indexed □-expressions representing the partial iterates involved in computing the fixed-point value. These are defined as follows:

Definition 16. For each ordinal □, the indexed □-expression $\square_\square X . C$ has the following meaning in any given model □M, □□:

$$M[\![\square_\square X . C]\!] \square = \square_{\square < \square} M[\![C]\!] \square[X{:=}M[\![\square_\square X . C[X]]\!] \square] \tag{9}$$

Note that by definition, for any cardinal □ greater than the cardinality of the domain of M, $M[\![\square X . C]\!] \square$ is equal to the union of all the sets $M[\![\square_\square X . C]\!] \square$ for □ less than any □.

5.1 Reducing Constraint Set Satisfiability to Set Expression Satisfiability

We begin by reducing satisfiability of sets of R-Tarskian set constraints to satisfiability for single R-Tarskian set expressions. We say that a Tarskian set expression C is *satisfiable* if there exists □M, □□ such that $M[\![C]\!] \square$ is nonempty.

Definition 17. For any set □ of Tarskian set constraints and set F of operation symbols we define □[□, F] to be the following set expression:

$$\mu X . \left[\begin{array}{c} (\neg W_1 \cap U_1) \cup \ldots \cup (\neg W_n \cap U_n) \cup \\ \bigcup_{i,j} f_i(\mathbf{T}, \ldots, \mathbf{T}, X, \mathbf{T}, \ldots, \mathbf{T}) \end{array} \right] . \tag{10}$$

Here X is a set variable not occurring in Σ, $U_1 \subseteq W_1, \ldots, U_n \subseteq W_n$ are the positive set constraints in Σ, $\mathbf{T}$ is the set expression $Z \cup \neg Z$ for some arbitrary set variable Z, and the expression $f_i(\mathbf{T}, \ldots, \mathbf{T}, X, \mathbf{T}, \ldots, \mathbf{T})$ ranges over all set expressions where f_i is an operation appearing in F and X occurs as the jth argument.

Intuitively, we have $x \in C[\Sigma, F]$ if there exists a y reachable by inverse operations in F from x such that y violates a positive constraint in Σ. If $x \in \neg C[\Sigma, F]$ then the positive constraints in Σ are satisfied at all points reachable by inverse operations in F from x. If $\langle M, \rho\rangle$ satisfies Σ then $M[\![C[\Sigma, F]]\!]\rho$ is the empty set for any F. To formalize these properties, we introduce the following definitions, in which we take $\langle M, \rho\rangle$ to be a model, x a domain element of M, and F a set of operation symbols:

Definition 18. The one step predecessors of x in M relative to F, written Pred(x, M, F), are the domain elements y of M such that for some operation f in F there is some tuple $\langle z_1, \ldots, z_n, x\rangle$ in $M(f)$ where y is equal to z_i for some i.

Definition 19. Let $\langle M, \rho\rangle$ be a model, x a domain element of M, and F a set of operation symbols. For each natural number n, define the n-step inverse closure of x relative to M and F, written $IC_n(x, M, F)$, as follows:

$$IC_0(x, M, F) = \{x\}$$

$$IC_{n+1}(x, M, F) = \{x\} \cup \bigcup_{y \in \mathrm{Pred}(x, M, F)} IC_n(y, M, F)$$

The *inverse closure* of x in M under F, written $IC(x, M, F)$, is the union over all natural numbers n of $IC_n(x, M, F)$. The *inverse closure substructure* of M generated by x and F, written $M_{x,F}$ is the structure whose domain is $IC(x, M, F)$ and such that for each nondeterministic operation $f \in F$ we have that $M_{x,F}(f)$ is the restriction of the relation $M(f)$ to $IC(x, M, F)$. For any variable interpretation ρ, we the inverse image restriction of ρ with respect to M, x, and F, written $\rho_{x,M,F}$, interprets each variable X as $\rho(X) \cap IC(x, M, F)$.

A set expression C can be thought of as a "predicate" about domain objects that only "looks at" the inverse closure substructure of its given argument object over the function symbols appearing in C. This view of set expressions leads to the following lemma about the expression $C[\Sigma,F]$ defined above:

Lemma 5: For any set Σ of R-Tarskian set constraints and set F of operation symbols, the expression $C[\Sigma,F]$ denotes in any model $\langle M, \rho\rangle$ the set of all domain elements x such that some y □ $IC(x, M, F)$ is a counter-example to some positive constraint in □. In other words, for some U □ W in □, y is in the denotation of U but not in the denotation of W in □M, □□.

Proof: We say that a domain element y of M "fails" a constraint U □ W in □ if y □ $M[\![U]\!]$ □ and y □ $M[\![W]\!]$ □. We denote the indexed versions of the □-expression □[□,F] as □$_□$[□,F] for index □. We first observe that regardless of the cardinality of the domain of M, the expressions □[□,F] and □$_□$[□,F] denote the same set in □M, □□ — that is, □ reaches a fixed-point after a countable number of iterations. We can then show by induction on natural numbers i that the following holds for all i:

$$\begin{array}{c} x \;\square\; \square_{i+1}(\square, F) \\ \text{iff} \\ \text{some } y \text{ in } IC_i(x, M, F) \text{ fails some } U \;\square\; W \text{ in } \square \end{array} . \tag{11}$$

The lemma follows from this fact and that □ reaches a fixed-point at □$_□$. ❑

Lemma 6: Let □M, □□ be a model, with x an element of the domain of M. Let F be a set of operation symbols. For any set expression C involving only operation symbols from F the following statement holds:

$$M[\![C]\!] \square \square\; IC(x, M, F) = M_{x, F}[\![C]\!] \square_{x, M, F} . \tag{12}$$

Proof: Fix a structure M, set of operations F, and element x of the domain of M. It is straightforward to show by structural induction on set expressions C involving only operations in F that for all variable interpretations □, Equation (12) holds. We need the quantification over variable interpretations in the induction hypothesis in order to handle the case of □ expressions. ❑

It is easy to determine whether □ is satisfied by the empty model, where all set expressions denote the empty set. To determine whether □ is satisfied by a non-empty model, we use the following lemma.

Lemma 7: Suppose □ is a set of Tarskian constraints not involving deterministic operations. Then □ is satisfiable by a nonempty model if and only if the set expression

$$f(U_1 \square \square W_1, \ldots, U_n \square \square W_n) \square \square\square[\square, F] \tag{13}$$

is satisfiable, where f is a fresh operation symbol, $U_1 \not\square\; W_1, \ldots, U_n \not\square\; W_n$ are all the negative constraints in □, and F is the set of all operation symbols occurring in □ together with f.

Proof: First suppose $\langle M, \rho\rangle$ satisfies Σ. We show that the set expression above (13) is satisfiable. Note that by Lemma 5, $M[\![\Phi[\Sigma, F]]\!]\rho$ is empty. For each negative constraint $U_i \not\subseteq W_i$ in Σ select a y_i such that $y_i \in U_i \cap \neg W_i$. Now extend M to M' by interpreting f as the operation containing the single tuple $\langle y_1, \ldots, y_n, x\rangle$ where x is an arbitrary domain element of M. Now x is the desired element of the above set expression in the extended model $\langle M', \rho\rangle$.

Conversely suppose that x is in the denotation of set expression (13) in some model $\langle M, \rho\rangle$. Lemmas 5 and 6 imply that the inverse image substructure $\langle M_{x, F}, \rho_{x, M, F}\rangle$ is a model of Σ, as follows. Since $x \in M[\![\neg\Phi[\Sigma, F]]\!]\rho$ by inspection of (13), and $x \in IC(x, M, F)$ by definition, we can conclude by Lemma 6 that $x \in M_{x, F}[\![\neg\Phi[\Sigma, F]]\!]\rho_{x, M, F}$, and thus by Lemma 5 that there are no counterexamples to positive constraints in Σ relative to $\langle M_{x, F}, \rho_{x, M, F}\rangle$ in $IC(x, M_{x, F}, F)$. But $IC(x, M_{x, F}, F)$ is the entire domain of $M_{x, F}$, and so $\langle M_{x, F}, \rho_{x, M, F}\rangle$ satisfies all the positive constraints in Σ. $\langle M_{x, F}, \rho_{x, M, F}\rangle$ satisfies the negative constraints because by inspection of (13) and our choice of x, x is in $M[\![f(U_1 \cap \neg W_1, \ldots, U_n \cap \neg W_n)]\!]\rho$ and by the definition of IC, x is in $IC(x, M, F)$ — so we have x in $M_{x, F}[\![f(U_1 \cap \neg W_1, \ldots, U_n \cap \neg W_n)]\!]\rho_{x, M, F}$ by Lemma 6. This implies the existence of some domain element y_i in the set $M_{x, F}[\![U_i \cap \neg W_i]\!]\rho_{x, M, F}$ for each i, implying that $\langle M_{x, F}, \rho_{x, M, F}\rangle$ satisfies each negative constraint $U_i \not\subseteq W_i$ in Σ as desired. ❑

Lemma 7 fails if we allow deterministic operations. For example, let Σ consist of the constraint set $\mathbf{T} \not\subseteq \mathbf{F}$ and $f(\mathbf{T}) \subseteq \mathbf{F}$, where f is deterministic and $\mathbf{T}$ and $\mathbf{F}$ denote the universal and empty sets, respectively. Σ is not satisfiable but the set expression $g(\mathbf{T} \cap \neg\mathbf{F}) \cap \neg\Phi[\Sigma, F]$ is satisfiable. The proof fails because we cannot simply restrict the meaning of a deterministic operation f to the smaller domain of $IC(x, M, F)$ for some x as we did for nondeterministic operations — the relation resulting from this restriction may not be a suitable meaning for a deterministic operation because it may not be total.

5.2 The Herbrand μ-calculus

Set satisfiability in both the modal μ-calculus and the Tarskian μ-calculus are polynomial time reducible to set satisfiability in a language we call the Herbrand μ-calculus. All of these calculi include set variables, Boolean operations on sets, and least fixed point expressions of the form $\mu X . C[X]$ where X occurs positively in $C[X]$. The modal μ-calculus has no application expressions but instead has set expressions of the form $\langle R\rangle C$ where R is a binary symbol. The set expression $\langle R\rangle C$ denotes the set $\{x : \exists y \in C\ R(x, y)\}$. The Tarskian μ-calculus consists of the Tarskian set expressions defined here but without deterministic operations. The Herbrand μ-calculus has same syntax as the Tarskian μ-calculus but with only deterministic operations which are interpreted over the fixed universe of (possibly infinite) Herbrand terms. The set expression $f(C_1, \ldots, C_n)$ denotes the set of (possibly infinite) terms of the form $f(t_1, \ldots, t_n)$ with each $t_i \in C_i$. In the Herbrand calculus we only consider the satisfiability problem for closed set expressions (those not containing free set variables).

The closed Herbrand μ-calculus seems most natural for understanding the exponential time satisfiability algorithms for set expressions in these calculi (Street and Emerson, 1989) (Emerson and Jutla, 1988) (Safra, 1988). The Herbrand calculus is based on the Herbrand universe of possibly infinite terms over a given set of function symbols. This would seem to indicate a relationship between the Herbrand calculus and Herbrand set constraints. However, in traditional Herbrand set constraint problems we are concerned with the existence of certain *sets* of Herbrand terms while here we are concerned with the existence of a single (possibly infinite) term satisfying given constraints (because here we are concerned with satisfiability of set expressions rather than satisfiability of sets of subset constraints).

There are many interesting examples of term sets definable in the Herbrand μ-calculus. The expression $\mu X . a \cup f(X)$ is the set of all finite terms which are some number of applications of f to a. We let $\nu X . C[X]$, a greatest fixed point expression, be an abbreviation for $\neg\mu X . \neg C[\neg X]$. The expression $\nu X . f(X)$ denotes a singleton set containing the infinite term $f(f(f(\ldots)))$. We will abbreviate this expression as f^{ω}. Another interesting example is $\mu X . g^{\omega} \cup f(X) \cup g(X)$. This is the set of infinite terms constructed from monadic function symbols f and g that have only finitely many occurrences of f. One can similarly define the set of infinite terms constructed from f and g that have only finitely many occurrences of g. Any satisfiability testing procedure must be capable of determining that the intersection of these two term sets is empty. It is known that the Herbrand μ-calculus defines exactly those term sets definable by Rabin tree automaton, or alternatively by formulas of SnS (the second order theory of n successors) (Emerson and Jutla, 1991).

The following theorem is shown in () and is cited here because in the next subsection we reduce Tarskian set expression satisfiability to Herbrand μ-calculus set expression satisfiability.

Theorem 10: ()**CITATION??** Set expression satisfiability in the Herbrand μ-calculus is decidable in exponential-time.

5.3 Reducing Tarskian Set Satisfiability to Herbrand μ-Calculus Satisfiability

Here we provide a reduction from R-Tarskian set expressions to Herbrand μ-calculus set expressions, preserving expression satisfiability. In addition to providing an exponential-time procedure for R-Tarskian expression satisfiability, this reduction also provides an alternative to the known reduction from the modal μ-calculus to the Herbrand μ-calculus — our alternative reduction passes through the R-Tarskian calculus and has a simplified correctness proof compared to the known reduction given in ()**CITATION??**. Note that there is a trivial satisfiability preserving reduction from the modal μ-calculus to the Tarskian μ-calculus where $\langle R\rangle C$ is translated to $R(C)$.

The reduction from the Tarskian calculus to the Herbrand calculus is almost as simple syntactically but quite a bit more difficult to prove correct.

Definition 20. For any expression C of the Tarskian calculus we define $T(C)$ by the following equations:

$$\begin{aligned} T(Y) &= Y, \text{ for variables } Y \\ T(\neg C) &= \neg T(C) \\ T(C_1 \cup C_2) &= T(C_1) \cup T(C_2) \\ T(\mu X . C) &= \mu X . T(C) \\ T(f(C_1, \ldots, C_n)) &= \mu X . (f(T(C_1), \ldots, T(C_n)) \cup g(\mathbf{T}, X) \cup g(X, \mathbf{T})) \end{aligned}$$

where X is a fresh set variable and g a binary operation symbol not in C.

We will show that if C is a closed Tarskian set expression then C is satisfiable if and only if $T(C)$ is satisfiable. Since free set variables can be replaced with set constants (nondeterministic operations of no arguments) it suffices to consider closed expressions. For an expression C of the Herbrand μ-calculus we define $[\![C]\!]\rho$ by analogy with $M[\![C]\!]\rho$ — in the Herbrand calculus no structure is required. If C is closed then we write $[\![C]\!]$ to denote the meaning of C independent of any variable interpretation.

The fresh function symbol g in Definition 20 is used to represent the many possible output values of the Tarskian relation f for particular arguments — in the Herbrand calculus f can have only one output for each domain tuple. This intuition is captured with the following definition and the Tarskian model we define using it below.

Definition 21. Given a set S of Herbrand terms, we define the set of terms *g-accessible* from S, written $g\text{-}acc(S)$ as follows:

$$\begin{aligned} g\text{-}acc_0(S) &= S \\ g\text{-}acc_{i+1}(S) &= g\text{-}acc_i(S) \cup \{g(y, z) : y \in g\text{-}acc_i(S) \vee z \in g\text{-}acc_i(S)\} \\ g\text{-}acc(S) &= \bigcup_i g\text{-}acc_i(S) \end{aligned}$$

We further define M_g to be the Tarskian structure whose domain is the set of all (possibly infinite) Herbrand terms and such that $M_g(f)$ is the infinite set of tuples $\langle y_1, \ldots, y_n, x\rangle$ such that x is g-accessible from $\{f(y_1, \ldots, y_n)\}$.

Using this definition, we can show that if $T(C)$ is satisfiable in the Herbrand calculus, then so is C in the Tarskian calculus, as follows.

Theorem 11: For any set expression C, for any variable environment □ mapping variables to sets of (possibly infinite) Herbrand terms we have $M_g[\![C]\!]\square$ equals $[\![T(C)]\!]\square$.

Proof: (sketch) The main proof is by induction on the structure of the set expression C, noting that the quantification over □ is in the inductive hypothesis (*i.e.,* the induction hypothesis gives us the theorem for all small class expressions for all variable interpretations □). We discuss only the key case here, when C is an application expression $f(C_1, \ldots, C_n)$ for which for all □ we have $M_g[\![C_i]\!]\square$ equal to $[\![T(C)]\!]\square$ for each i. Note that the translation $T(C)$ is a □-expression, and as such has a denotation defined as the union over an infinite collection of indexed □-expressions. We write $T_{\square}(C)$ for the expression $T(C)$ with the □-expression indexed by □, and observe that $T_{\square}(C)$ is equal to $T_{\square+1}(C)$. We can now show by induction on the natural number i that

$$g\text{-}acc_i(\{f(y_1, \ldots, y_n) : y_j \,\square\, M_g[\![C_j]\!]\square\}) = [\![T_{i+1}(C)]\!]\square. \qquad (14)$$

Observing that $M_g[\![C]\!]\square$ is by the definition of M_g equal to the set of Herbrand terms $g\text{-}acc(\{f(y_1, \ldots, y_n) : y_j \,\square\, M_g[\![C_j]\!]\square\})$, it now follows that $M_g[\![C]\!]\square$ is equal to $[\![T(C)]\!]\square$ as desired. ❑

Corollary 2: If $T(C)$ is satisfiable in the Herbrand calculus then so is C in the Tarskian calculus.

Proof: When $T(C)$ is satisfiable, the set $[\![T(C)]\!]\square$ is nonempty and then so is the set $M_g[\![C]\!]\square$, hence C is satisfiable. ❑

Now we prove the converse. This proof is essentially a simplification of the proof given in (Street and Emerson, 1989) that any satisfiable set expression of the modal □-calculus can be satisfied by a model with bounded branching. First we simplify the problem by converting every expression to a purely positive form. This is done by introducing conjunctions, greatest fixed points $\square X.C$ and "disapplications" $[f](C_1, \ldots, C_n)$. We define $M[\![\square X.C]\!]\square$ to be the greatest subset S of the domain of M such that $S = M[\![C]\!]\square[X := S]$. We define the meaning of disapplications by $[f](C_1, \ldots, C_n) = \square f(\square C_1, \ldots, \square C_n)$. In the Tarskian calculus we have $x \,\square\, M[\![[f](C_1, \ldots, C_n)]\!]\square$ if and only if for every tuple $\square y_1, \ldots, y_n \square$ such that $\square y_1, \ldots, y_n, x \square\square\, M(f)$ we have that $y_i \,\square\, M[\![C_i]\!]\square$ for at least one i. We can now eliminate negation from any closed expression using de Morgan's laws and the following rules to push negations down:

$$\begin{aligned}
\square\square X.C[X] &= \square X.\square C[\square X] \\
\square\square X.C[X] &= \square X.\square C[\square X] \\
\square f(C_1, \ldots, C_n) &= [f](\square C_1, \ldots, \square C_n) \\
\square[f](C_1, \ldots, C_n) &= f(\square C_1, \ldots, \square C_n)
\end{aligned}$$

Since all recursion must be monotone, variables can not appear in negative contexts in closed expressions and negation disappears entirely.[3] For any set expression of either the Tarskian or Herbrand μ-calculus we let $\mathrm{pos}(C)$ be the positive form of C achieved by pushing negations down using these rules. We can extend our Tarskian-to-Herbrand translation T to handle greatest fixed points and disapplications by adding:

$$\begin{aligned} T(\nu X.C) &= \nu X.T(C) \\ T([f](C_1, \ldots, C_n)) &= \mu X.[f](T(C_1), \ldots, T(C_n)) \cup [g](\mathbf{F}, X) \cup [g](X, \mathbf{F}) \end{aligned}$$

We now have that $T(\mathrm{pos}(C))$ is semantically equivalent to $T(C)$ and that $\mathrm{pos}(C)$ is semantically equivalent to C. (more detail on this??) So to prove that T preserves satisfiability we need only consider positive expressions.

As with μ-expressions, we add syntactically indexed fixed point expressions for ν-expressions of the form $\nu_\alpha X.C$ where α is any ordinal. The semantics of these expressions are defined by the following equation.[4]

$$M[\![\nu_\alpha X.C[X]]\!]\rho = \bigcap_{\beta<\alpha} M[\![C[\nu_\beta X.C[X]]]\!]\rho$$

As with μ-expressions, we have that $M[\![\nu X.C]\!]\rho = M[\![\nu_\kappa X.C]\!]\rho$ where κ is any cardinal larger than the cardinality of M. The same statement holds for greatest fixed-point expressions. An unindexed fixed point expression $\mu X.C$ can be viewed as a syntactic variant of $\mu_\Omega X.C$ where Ω is the class of all ordinals. Intuitively, Ω plays the role of a "largest ordinal". So we can assume that all fixed point expressions are indexed. An expression in which all fixed point expressions are indexed with Ω (*i.e.*, unindexed) will be called a *maximally indexed expression*.

The following definitions lead to a definition of the term "execution tree". An execution tree can be viewed as an "explanation" of why a given Tarskian set expression is satisfiable. By showing how to encode execution trees as Herbrand terms we show how to construct a Herbrand term satisfying $T(C)$ whenever we have an execution tree "explanation" of the satisfiability of C.

Definition 22. We define a *type* to be a set τ of positive closed R-Tarskian set expressions satisfying the following conditions:

- If $C \cup W \in \tau$ then either $C \in \tau$ or $W \in \tau$.
- If $\mu_\alpha X.C[X] \in \tau$ then $C[\mu_\beta X.C[X]] \in \tau$ for some $\beta < \alpha$.
- If $\nu_\alpha X.C[X] \in \tau$ then $C[\nu_\beta X.C[X]] \in \tau$.

3. If P is a zero-ary nondeterministic operation of the Tarskian calculus then we can think of $[P]$ as a syntactic variant of $\neg P$.

4. In these equation α can be empty, in which case the empty union denotes the empty set and the empty intersection the entire domain of M. α can be either a limit or successor ordinal.

Definition 23. We define an *execution tree* to be a pair $\langle \tau, \Sigma \rangle$ such that τ is a type and Σ is a set of expressions of the form $f(\gamma_1, \ldots, \gamma_n)$ where each γ_i is (recursively) an execution tree. We will be interested in infinite execution trees. We write $C \in \gamma$ if γ is a tree of the form $\langle \tau, \Sigma \rangle$ with $C \in \tau$. A tree γ' is called a *subtree* of a tree $\gamma = \langle \tau, \Sigma \rangle$ if either $\gamma = \gamma'$ or there is some $f(\gamma_1, \ldots, \gamma_n)$ in Σ such that γ' is (recursively) a subtree of γ_i for some i.

Definition 24. An execution tree is called *locally consistent* if for every subtree $\langle \tau, \Sigma \rangle$ we have that both τ and Σ are countable sets such that:

- for every $f(\gamma_1, \ldots, \gamma_n) \in \Sigma$ and $[f](W_1, \ldots, W_n) \in \tau$ there is some i such that $W_i \in \gamma_i$, and
- for every $f(W_1, \ldots, W_n) \in \tau$ there is some $f(\gamma_1, \ldots, \gamma_n) \in \Sigma$ such that for all i we have $W_i \in \gamma_i$.

We are now ready for a key lemma stating that any satisfiable C can be "explained" by an execution tree.

Lemma 8: If C is a closed satisfiable positive R-Tarskian set expression then there exists a locally consistent execution tree γ such that $C \in \gamma$.

Proof: Consider an arbitrary model $\langle M, \rho \rangle$. We say that a set of expressions Φ is true at x (in $\langle M, \rho \rangle$) if $x \in M[\![W]\!]\rho$ for all $W \in \Phi$. For any countable set Φ of expressions true at a point x in $\langle M, \rho \rangle$ we describe how to construct a locally consistent execution tree $E_{M,\rho}(\Phi, x)$ whose root type contains Φ. Let τ be a countable type containing Φ and true at x in $\langle M, \rho \rangle$. For each expression $f(C_1, \ldots, C_n)$ in τ construct an element of Σ as follows. Select points $\langle y_1, \ldots, y_n \rangle$ such that $\langle y_1, \ldots, y_n, x \rangle \in M(f)$ and $y_i \in M[\![C_i]\!]\rho$. For each $[f](W_1, \ldots, W_n) \in \tau$ select a W_i such that $y_i \in M[\![W_i]\!]\rho$. Let Φ_i consist of C_i and all selected W_i. Now add $f(E_{M,\rho}(\Phi_1, y_1), \ldots, E_{M,\rho}(\Phi_n, y_n))$ to Σ. Finally return the pair $\langle \tau, \Sigma \rangle$ as $E_{M,\rho}(\Phi, x)$. It is straightforward to prove by induction on the structure of $E_{M,\rho}(\Phi, x)$ that $E_{M,\rho}(\Phi, x)$ is a locally consistent execution tree with a root containing Φ.

Since C is satisfiable, there must be some $\langle M, \rho \rangle$ and domain element x such that $\{C\}$ is true at x in $\langle M, \rho \rangle$, and therefore $E_{M,\rho}(\{C\}, x)$ is the desired locally consistent execution tree containing C. ❑

We now show how to construct a closed Herbrand term $t(\gamma)$ from any execution tree γ such that $t(\gamma) \in [\![T(C)]\!]$ whenever γ is a locally consistent tree with $C \in \gamma$.

Definition 25. For any execution tree γ, for a a constant not appearing in γ, we define the Herbrand term $t(\gamma)$ by the following rules, recursively on γ:

$$t(\square\square, \{\ \}\square) = a$$
$$t(\square\square, \{f(\square_1, \ldots, \square_n)\} \square \square_2\square) = g(f(t(\square_1), \ldots, t(\square_n)), t(\square\square, \square_2\square)) \qquad (15)$$

The second rule is applied “fairly” so that if □is $\square\square$, $\square\square$ and $f(\square_1, \ldots, \square_n) \square \square$ then $t(\square)$ is g-accessible from $\{f(t(\square_1), \ldots, t(\square_n))\}$.

In order to prove the desired property for $t(\square)$, we need to define an unusual well-founded order for use in an induction proof.

Lemma 9: There exists a well founded ordering < on closed syntactic expressions such that:

- $W < C$ for any W and C such that W is a closed proper subexpression of C,
- $C[\square_\square X.C[X]] < \square_\square X.C[X]$ when $\square < \square$, and
- $C[\square_\square X.C[X]] < \square_\square X.C[X]$ when $\square < \square$.

Proof: We define the Fisher-Ladner closure (Fischer and Ladner, 1979) of an expression C to be the least set FL(C) of maximally indexed expressions such that:

- $C_\square \square \mathrm{FL}(C)$ for $C_\square$ the result of maximally indexing all fixed-points in C,
- any closed subexpression of an element of FL(C) is an element of FL(C),
- if $\square_\square X.C[X] \square \mathrm{FL}(C)$ then $C[\square_\square X.C[X]] \square \mathrm{FL}(C)$, and
- if $\square_\square X.C[X] \square \mathrm{FL}(C)$ then $C[\square_\square X.C[X]] \square \mathrm{FL}(C)$.

The set FL(C) is finite — it has one member for each (possibly open) subexpression of C. We define the *rank* of an expression E to be the level of nesting of recursion of *closed* subexpressions of E — the rank of E is zero if it has no □ or □ subexpressions, the rank of any *closed* □ or □ expression is one more than the rank of its body, and the rank of any other expression is equal to the largest rank of any proper subexpression. We define the *signature* of an expression C to be the tuple $\square_1, \ldots, \square_n$ where n is the largest rank of any expression in FL(C) and each $\square_i$ is the maximum index of all closed recursion subexpressions of C of rank i, or zero if there is no such subexpression.

We order signatures first by length and then lexicographically within signatures of the same length. We can now define the order < to order expressions first by signature then by syntactic depth (breaking any remaining ties randomly).

To see the first property of the ordering claimed in the lemma observe that the following hold whenever C is a closed subexpression of W:

- $\mathrm{FL}(C) \square \mathrm{FL}(W)$,
- the signature of C is shorter or equal in length to that of W, and

- every indexed □- or □-subexpression of C is also counted in the signature of W.

These properties allow us to conclude that the signature of C is always less than or equal to that of W. Because C is a subexpression of W, the syntactic depth of C is always less than or equal to that of W as well, allowing us to conclude that C is ordered ahead of W.

To see that the second and third properties claimed in the lemma hold, note first that the closure of a □-expression $\mathrm{FL}(\square_\square X.C[X])$ is equal to the closure of the unrolling $\mathrm{FL}(C[\square_\square X.C[X]])$, and therefore both the □-expression and its unrolling have the same signature length. Observe then that the signature of any closed □-expression $\square_\square X.C$ is of the form $\square\square_1, \ldots, \square_{j-1}, \square, 0, \ldots, 0\square$ where j is the rank of $\square_\square X.C$. The signature of an unrolling $C[\square_\square X.C[X]]$ of $\square_\square X.C$ with $\square < \square$ is then given by $\square\square_1, \ldots, \square_{j-1}, \square, \square_1, \ldots, \square_k\square$. The second signature is lexicographically smaller than the first (given $\square < \square$) and hence unrolling strictly reduces signature (the same holds for □-expressions). We can conclude that unrolling reduces the ordering we have defined and thus that the ordering satisfies all the desired properties.❑

Lemma 10: If □ is a locally consistent execution and $C \square \square$ then $t(\square) \square [\![T(C)]\!]$.

Proof: We define a □-reindexing of an expression C to be any expression $C\square$ identical to C except for the indices of □-expressions. We prove by transfinite induction on expressions using the ordering of Lemma 9 that if C is any □-reindexing of an expression $C\square\square\ \square$ then $t(\square) \square [\![T(C)]\!]$. To show the need for □-reindexing we will explicitly give the proof for □-expressions. Consider an expression $\square_\square X.C[X]$ which is a □-reindexing of an expression $\square_\square X.C\square[X] \square \square$. We have $C\square[\square_\square X.C\square[X]] \square \square$ from the closure properties of types given in Definition 22. Now consider any ordinal $\square < \square$. By the induction hypothesis we have that $t(\square) \square [\![T(C[\square_\square X.C[X]])]\!]$. But we have that $[\![T(\square_\square X.C[X])]\!]$ is the intersection of all such sets so we have $t(\square) \square [\![T(\square_\square X.C[X])]\!]$. The other cases of the induction are straightforward given the above properties of the well-founded ordering on expressions. ❑

Theorem 12: $T(C)$ is satisfiable if and only if C is satisfiable.

6. Conclusions

A wide variety of set calculi have been studied in the logic and computer science literature. Tarskian set expressions yield a natural set calculus that has received surprisingly little attention. We have answered a variety of questions concerning the computational complexity of Tarskian set constraints but several problems remain open. It seems likely that Tarskian set constraints without recursion (but with deterministic operations) can be solved in nondeterministic singly exponential time. This would follow from a demonstration that the satisfiability of prequadratic Diophantine equations is in NP. The decidability of Tarskian set constraints with recursion and deterministic operations of arity at least one, or with arity just zero, remains open. It seems likely that techniques used in decision procedures for the modal □-calculus can also be used to construct decision procedures for these cases, although this has not yet been done.

7. References

[1] A. Aiken, D. Kozen, M. Vardi, and E. Wimmers. The complexity of set constraints. In E. Börger, Y. Gurevich, and K. Meinke, editors, *Proceedings of the 1993 Conference on Computer Science Logic (CSL'93)*, volume 832 of *Lecture Notes in Comput. Science*, pages 1–17. European Association for Computer Science Logic, Springer, September 1993.

[2] A. Aiken, D. Kozen, and E. Wimmers. Decidability of systems of set constraints with negative constraints. *Information and Computation*, 122(1):30-44, 1995. Also Cornell University Technical Report 93-1362, June, 1993.

[3] A. Aiken, E. Wimmers, and T. Lakshman. Soft typing with conditional types. In *ACM Symposium on Principles of Programming Languges*, pages 163–173. Association for Computing Machinery, 1994.

[4] R. Brachman and J. Schmolze. An overview of the KL-one knowledge representation system. *Computational Intelligence*, 9(2):171–216, 1985.

[5] D. Calvanese, M. Lenzerini, and D. Nardi. A unified framework for class-based representation formalisms. In *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning*, pages 109–119, May 1994.

[6] W. Charatonik and L. Pacholski. Negative set constraints with equality. In *Proceedings, Ninth Annual Symposium on Logic in Computer Science*, pages 128–136, Paris, France, 4 July 1994. IEEE Computer Society Press.

[7] W. Charatonik and L. Pacholski. Set constraints with projections are in NEXPTIME. In *35th Annual Symposium on Foundations of Computer Science*, pages 642–653, Santa Fe, New Mexico, 20–22 November 1994. IEEE.

[8] W. Downing and J. H. Gallier. Linear Time Algorithms for testing the Satisfiability of Propositional Horn formulae. *Journal of Logic Programming,* 1(3):267-284, 1984.

[9] E. Emerson and C. S. Jutla. The complexity of tree automata and modal logics of programs (extended abstract). In *29th Annual Symposium on Foundations of Computer Science*, pages 328–337, White Plains, New York, 24–26 October 1988. IEEE.

[10] E. A. Emerson and C. S. Jutla. Tree automata, □-calculus, and determinancy. In *32nd Annual Symposium on Foundations of Computer Science*, pages 368–377, San Juan, Puerto Rico, 1–4 October 1991. IEEE.

[11] M. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, April 1979.

[12] L. Bachmair, H. Ganzinger, and U. Waldmann. Set constraints are the monadic class. In *Proceedings, Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 75–83, Montreal, Canada, 19–23 June 1993. IEEE Computer Society Press.

[13] G. D. Giacomo and M. Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics. In *Proceedings of the 12th National Conference on Artificial Intelligence*, Volume 1, pages 205–212, Seattle, Washington, July, 1994. AAAI Press.

[14] G. D. Giacomo and M. Lenzerini. Concept languages with number restrictions and fixed points and its relationship with mu-calculus. In *11th European Conf. on Artificial Intelligence*, pages 356–360, August, 1994.

[15] R. Givan and D. McAllester. New results on local inference relations. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, pages 403–412. Morgan Kaufman Press, October 1992. Also available as internet file http://www.ece.purdue.edu/~givan/kr92.ps.

[16] R. Goldblatt. Varieties of complex algebras. *Annals of Pure and Applied Logic*, 44:173–242, 23 October 1989.

[17] N. Heintze and J. Jaffar. A finite presentation theorem for approximating logic programs. In *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pages 197–209, San Francisco, California, January 1990.

[18] N. Heintze and J. Jaffar. A decision procedure for a class of set constraints. In *Proceedings, Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 42–51, Philadelphia, Pennsylvania, 4–7 June 1990. IEEE Computer Society Press.

[19] B. Jònnson and A. Tarski. Boolean algebras with operators. part i. *Amer. J. Math*, 73:891–939, 1951.

[20] B. Jònsson and A. Tarski. Boolean algebras with operators. part ii. *Amer. J. Math*, 74:127–167, 1952.

[21] D. Kozen. Results on the propositional □-calculus. *Theoretical Computer Science*, 27:333–354, December 1983.

[22] D. Kozen. Logical aspects of set constraints. In *Proceedings of the 1993 Conference on Computer Science Logic*. European Association for Computer Science Logic, September 1993.

[23] D. Kozen, J. Palsberg, and M. I. Schwartzbach. Efficient recursive subtyping. In *Conference Record of the Twentieth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Language*s, pages 419–428, Charleston, South Carolina, January 1993.

[24] D. Kozen, J. Palsberg, and M. I. Schwartzbach. Efficient inference of partial types. *Journal of Computer and System Sciences*, 49(2):306–324, October 1994.

[25] H. R. Lewis. Complexity results for classes of quantificational formulas. *Journal of Computer and System Sciences*, 21:317–353, December 1980.

[26] D. McAllester and R. Givan. Natural language syntax and first order inference. *Artificial Intelligence*, 56:1–20, 1992. Internet file http://www.ece.purdue.edu/~givan/papers/aij1.ps.

[27] D. McAllester and R. Givan. Taxonomic syntax for first order inference. *Journal of the ACM*, 40(2):246–283, April 1993. Internet file http://www.ece.purdue.edu/~givan/papers/jacm1.ps.

[28] F. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference*, pages 151–162. Morgan Kaufmann Publishers, 1991.

[29] V. R. Pratt. A near-optimal method for reasoning about actions. *Journal of Computer and System Sciences*, 20(2):231–254, April 1980.

[30] S. Safra. On the complexity of □-automaton. In *29th Annual Symposium on Foundations of Computer Science*, pages 319–327, White Plains, New York, 24–26 October 1988. IEEE.

[31] M. Schmidt-Schaub and G. Smalka. Attributive concept descriptions with complements. *Artificial Intelligence*, 47:1–26, 1991.

[32] R. S. Streett and E. A. Emerson. An automaton theoretic decision procedure for the propositional □-calculus. *Information and Computation*, 81(3):249–264, June 1989.

[33] T. Frühwirth, E. Shapiro, M. Vardi, and E. Yardeni. Logic programs as types for logic programs. In *Proceedings, Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 300-309, Amsterdam, The Netherlands, 15-18 July 1991. IEEE Computer Society Press.