CiAD

Second International Workshop on **Complexity in Automated Deduction**

Georg Gottlob, TU Wien, Vienna, Austria

Miki Hermann, LIX, Ecole Polytechnique, Paris, France

Michaël Rusinowitch, LORIA, Nancy, France

25–26 July 2002

Invited speakers

A. Blumensath and E. Grädel:	Finite Presentations of Infinite Structures: Automata and Interpretations
M. Cadoli:	The expressive power of Binary Linear Programming
R. Pichler:	Algorithms and Complexity of Model Representations

Submitted papers

D. Berwanger and E. Grädel:	Fixed point formulae and solitaire games
A. Durand and M. Hermann:	The Inference Problem of Propositional Circumscription of Affine Formulas is coNP-complete
M. Schmidt-Schauß and J. Stuber:	On the complexity of linear and stratified context matching problems
O. Terán and B. Edmonds:	Computational Complexity of a Constraint Model-based Proof of the Envelope of Tendencies in a MAS-based Simulation Model

Fixed point formulae and solitaire games

Dietmar Berwanger and Erich Grädel
Mathematische Grundlagen der Informatik
RWTH Aachen

Abstract

The model checking games associated with fixed point logics are parity games, and it is currently not known whether the strategy problem for parity games can be solved in polynomial time. We study Solitaire-LFP, a fragment of least fixed point logic, whose model checking games are nested soltaire games. This means that on each strongly connected component of the game, only one player can make non-trivial moves. Winning sets of nested solitaire games can be efficiently computed.

The model checking problem for Solitaire-LFP is PSPACE-complete in general and PTIME-complete for formulae of bounded width. On finite structures (but not on infinite ones), Solitaire-LFP is equivalent to transitive closure logic.

1 Introduction

Fixed point logics play an important role in many areas of logic. LFP, the extension of first-order logic by least and greatest fixed points is of fundamental importance in finite model theory, descriptive complexity, and databases. The modal μ-calculus L_μ is a similar extension of propositional modal logic. It relates to LFP in much the same way as multi-modal logic relates to first-order logic. This logic has also been extensively studied for a number of reasons. In terms of expressive power, it subsumes a variety of logics used in verification, in particular LTL, CTL, CTL*, PDL, and also many logics used in other areas of computer science, for instance game logic and description logics. Both LFP and L_μ have a rich theory, and are well-behaved in model-theoretic and also, to a large extent, in algorithmic terms.

Nevertheless there are still important open problems concerning their complexity. The most prominent one is whether the model checking problem for the modal μ-calculus, or, more generally, for LFP-formulae of bounded width can be solved in polynomial time. This problem is equivalent to an algorithmic problem in the theory of infinite games, namely the question whether winning sets in *parity games* can be computed in polynomial time. Parity games are two-person games on finite or infinite game graphs which admit infinite plays and where each position is assigned a natural number, called its priority. The winner of an infinite play is determined according to whether the least priority seen infinitely often during the play is even or odd. Parity games arise as the natural model checking games for fixed point logics. Priorities of game positions correspond to the alternation level of fixed point formulae. It is open whether winning sets and winning strategies for parity games can be computed in polynomial time. The best algorithms known today are polynomial in the size of the game, but exponential with respect to the number of priorities.

In this paper we have a closer look at a class of parity games that can be solved efficiently, and at the fixed point formulae that are associated with them. *Nested solitaire games* are parity games, where on each strongly connected component, only one player can make non-trivial choices. As we will show, the winning sets of a nested solitaire game can be computed in time that is linear in the product of the number of priorities with the size of the game graph.

We define Solitaire-LFP, a fragment of least fixed point logic, whose model checking games are nested solitaire games, and we analyse the algorithmic properties and the expressive power of this fragment. A corresponding fragment in the modal μ-calculus has already been studied in [6], and has been shown to be equivalent to the logic ECTL*. For Solitaire-LFP, it turns out that the model checking problem is PSPACE-complete in the general case (for formulae of unbounded width) and PTIME-complete for formulae of bounded width. Further we prove that on finite structures, Solitaire-LFP is equivalent to transitive closure logic (TC). To establish this result we exploit the solitaire-structure of the model checking game and the fact that TC-formulae are equivalent to stratified linear Datalog programs. We construct for every formula in Solitaire-LFP a stratified linear Datalog program which defines the winning positions in the associated model checking game.

A further consequence of this proof is that every formula in Solitaire-LFP of width k (of arbitrary alternation level) is equivalent, on finite structures, to an alternation-free fixed-point formula of width at most $2k$.

2 Least fixed point logic

Least fixed point logic, denoted LFP, extends first order logic by least and greatest fixed points of definable relational operators. Every formula $\psi(R, \boldsymbol{x})$, where R is a relation symbol and $\boldsymbol{x}$ is a tuple of variables of the same length as the arity of R, defines, on any structure $\mathfrak{A}$ of appropriate vocabulary, an update operator $F : \mathcal{P}(A^k) \to \mathcal{P}(A^k)$ given by $F : R \mapsto \{\boldsymbol{a} : (\mathfrak{A}, R) \models \psi(R, \boldsymbol{a})\}$. If R occurs only positively in ψ, this operator is monotone and has a *least fixed point*, that can also be constructed inductively, in *stages* $X^0 := \emptyset$, $X^{\alpha+1} := F(X^\alpha)$ for successor ordinals α, and $X^\lambda := \bigcup_{\alpha<\lambda} X^\alpha$ for limit ordinals λ. By monotoncity of F, the sequence of stages increases until it reaches the least fixed point. The *greatest fixed point* is constructed in a dual way.

Formally, LFP is defined by adding to the syntax of first order logic the following *fixed point formation rule:* If $\psi(R, \boldsymbol{x})$ is a formula with a relational variable R occurring only positively and a tuple of first-order variables $\boldsymbol{x}$, and if $\boldsymbol{t}$ is a tuple of terms (such that the lengths of $\boldsymbol{x}$ and $\boldsymbol{t}$ match the arity of R), then

$$[\mathbf{lfp}\, R\boldsymbol{x} \,.\, \psi](\boldsymbol{t}) \quad \text{and} \quad [\mathbf{gfp}\, R\boldsymbol{x} \,.\, \psi](\boldsymbol{t})$$

are also formulae, binding the variables R and $\boldsymbol{x}$.

The semantics of least fixed point formulae in a structure $\mathfrak{A}$, providing interpretations for all free variables in the formula, is the following: $\mathfrak{A} \models [\mathbf{lfp}\, R\boldsymbol{x} \,.\, \psi](\boldsymbol{t})$ if $\boldsymbol{t}^{\mathfrak{A}}$ is contained in the least fixed point of the update operator defined by ψ on $\mathfrak{A}$. Similarly for greatest fixed points.

Note that in formulae $[\mathbf{lfp}\, R\boldsymbol{x} \,.\, \psi](\boldsymbol{t})$ one might permit that ψ may have other free variables besides $\boldsymbol{x}$, which are called *parameters.* However, every LFP-formula can easily be transformed into an equivalent one without parameters, at the expense of increasing the arity of fixed point variables. In this paper, we only consider fixed point formulae without parameters.

The duality between least and greatest fixed point implies that for any formula ψ

$$[\mathbf{gfp}\, R\boldsymbol{x} \,.\, \psi](\boldsymbol{t}) \equiv \neg[\mathbf{lfp}\, R\boldsymbol{x} \,.\, \neg\psi[R/\neg R]](\boldsymbol{t}).$$

Using this duality together with de Morgan's laws, every LFP-formulae can be brought into *negation normal form*, where negation applies to atoms only.

The model checking problem for a logic $\mathcal{L}$ is to establish for a given formula $\psi \in \mathcal{L}$ and an appropriate finite structure $\mathfrak{A}$, whether $\mathfrak{A} \models \psi$. The complexity of model checking problems can be measured in different ways. In general, both the structure and the formula (more precisely, their representations) are considered as inputs and we speak about the *combined complexity* of $\mathcal{L}$. But in many instances it makes sense to fix either the formula or the structure and measure the complexity in terms of the other input, thus obtaining the notions of *expression complexity* and *data complexity*. We say that the data complexity of $\mathcal{L}$ is complete for a complexity class $\mathcal{C}$ if the model checking problem is in $\mathcal{C}$ for *every* fixed formula $\psi \in \mathcal{L}$, and if it is $\mathcal{C}$-hard for *some* fixed formula $\psi \in \mathcal{L}$ (and similarly for the expression complexity).

For general LFP-formulae the model checking complexity is well-known [10, 17].

Theorem 2.1. *The combined and expression complexity of* LFP *is* EXPTIME*-complete, and the data complexity is* PTIME*-complete.*

Note that these complexity bounds take into account only the length of the input formula. It turns out that the critical parameter responsible for the EXPTIME-completeness is actually the *width* of a formula, i.e., the maximal number of free variables in its subformulae. Fortunately, in many applications we only need formula of small width. In particular the modal μ-calculus can be translated to LFP-formulae of width two. For LFP-formulae of bounded width, better complexity bounds apply.

Proposition 2.2. *The model checking problem for* LFP*-formulae of bounded width (and for the modal μ-calculus) is contained in* NP $\cap$ co-NP *and* PTIME*-hard.*

It is open whether this problem can be solved in polynomial time. Positive results have been obtained for fragments of LFP. One such partial result involves the *alternation depth*, i.e., the number of genuine alternations between least and greatest fixed points in a formula. For LFP-formulae of bounded width and bounded alternation depth the model checking problem can be solved in polynomial time.

The fragments of bounded alternation depth in LFP induce a strict semantical hierarchy [3, 15]. On finite structures, this remains true for the modal μ-calculus (since it has the finite model property) but not for LFP. Every LFP-formula is equivalent, over finite structures, to an alternation free one, indeed to a formula with a single application of an **lfp**-operator to a first-order formula [10]. However, this result does not help to improve the model checking complexity, since the proof collapses d nested fixed points to one of width dk, and the complexity of LFP is exponential in the formula width.

Transitive closure logic. A semantic fragment of LFP that is well-behaved in terms of complexity is *transitive closure logic*, TC, which extends first order logic by a constructor for forming the transitive closure of definable relations. Syntactically, if $\varphi(\boldsymbol{x}, \boldsymbol{y})$ is a formula in variables $\boldsymbol{x}$, $\boldsymbol{y}$, and $\boldsymbol{s}$, $\boldsymbol{t}$ are terms, the tuples $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}$, and $\boldsymbol{t}$ being all of the same length, then

$$[\mathbf{tc}_{\boldsymbol{x},\boldsymbol{y}}\, \varphi(\boldsymbol{x}, \boldsymbol{y})](\boldsymbol{s}, \boldsymbol{t})$$

is a also a TC-formula. Its meaning can be expressed in terms of LFP as

$$[\mathbf{lfp}\, T\boldsymbol{xy} \,.\, \varphi(\boldsymbol{x},\boldsymbol{y}) \lor \exists \boldsymbol{z}(T\boldsymbol{xz} \land \varphi(\boldsymbol{z},\boldsymbol{y})](\boldsymbol{s},\boldsymbol{t}).$$

Observe that any TC-formula translates into an alternation free LFP-formula of the same width. The model checking complexity of TC is well-understood [9, 11, 17].

Proposition 2.3. *The model checking problem for* TC *is* PSPACE*-complete in the general case and* PTIME*-complete for formulae of bounded width. The data complexity is* NLOGSPACE*-complete.*

For future use in Section 5, we mention that transitive closure logic can be naturally characterised in terms of the database query language Datalog.

We recall that a *Datalog rule* is an expression of the form $H \leftarrow B_1, \dots, B_r$ where H, the *head* of the rule, is an atomic formula $Ru_1 \cdots u_s$, and $B_1, \dots, B_r$, the *body* of the rule, is a collection of literals (i.e., atoms or negated atoms). The relation symbol R is called the *head predicate* of the rule. A *basic Datalog-program* Π is a finite collection of rules such that none of its head predicates occurs negated in the body of any rule. Given a relational structure $\mathfrak{A}$ over the vocabulary of the input predicates, the program computes, via the usual fixed point semantics, an interpretation for the head predicates.

A *stratified Datalog program* is a sequence $\Pi = (\Pi_0, \dots, \Pi_r)$ of basic Datalog programs, called the *strata* of Π, such that each of the head predicates of Π is a head predicate in precisely one stratum Π_i and is used as an body predicate only in higher strata Π_j for $j > i$. In particular, this means that

(i) if a head predicate of stratum Π_j occurs *positively* in the body of a rule of stratum Π_i, then $j \leq i$, and

(ii) if a head predicate of stratum Π_j occurs *negatively* in the body a rule of stratum Π_i, then $j < i$.

The semantics of a stratified program is defined stratum per stratum. The body predicates of a stratum Π_i are either head predicates in the entire program Π or are head predicates of a lower stratum. Hence, once the lower strata are evaluated, we can compute the interpretation of the head predicates of Π_i as in the case of basic Datalog. For details, please consult [1].

A stratified Datalog program is *linear* if in the body of each rule there is at most one occurrence of a head predicate of the same stratum (but there may be arbitrary many occurrences of head predicates from lower strata). Linear programs suffice to define transitive closures, so it follows by a straightforward induction that TC $\subseteq$ Linear Stratified Datalog. The converse is also true (see [4, 7]).

Proposition 2.4. *Linear Stratified Datalog is equivalent to* TC.

3 Model checking games

Model checking problems, for almost any logic, can be formulated as the problem of computing winning positions in the appropriate evaluation games. For fixed point logics like LFP or L_μ, the evaluation games are *parity games.* A parity game is given by a transition system $\mathcal{G} =$

(V, V_0, E, Ω), where V is a set of positions with a designated subset V_0, $E \subseteq V \times V$ is a transition relation, and $\Omega : V \to \mathbb{N}$ assigns to every position a *priority*. The number of priorities in the range of Ω is called the *index* of $\mathcal{G}$. A *play* of $\mathcal{G}$ is a path $v_0, v_1, \ldots$ formed by the two players starting from a given position v_0. If the current position v belongs to V_0, Player 0 chooses a move $(v, w) \in E$ and the play proceeds from w. Otherwise, her opponent, Player 1, chooses the move. When no moves are available, the player in turn loses. In case this never happens the play goes on infinitely and the winner is established by looking at the sequence $\Omega(v_0), \Omega(v_1), \ldots$ If the least priority appearing infinitely often in this sequence is even, Player 0 wins the play, otherwise Player 1 wins.

Let $V_1 := V \setminus V_0$ be the set of positions where Player 1 moves. A *positional strategy* for Player i in $\mathcal{G}$ is a function $f : V_i \to V$ which indicates a choice $(v, f(v)) \in E$ for every position $v \in V_i$. (It is called positional, because it does not depend on the history of the play, but only on the current position.) A strategy f for Player i is a *winning* strategy if he wins every play in which he moves according to f. The Forgetful Determinacy Theorem for parity games [5] states that these games are always determined (i.e., from each position one of the players has a winning strategy) and in fact, positional strategies always suffice.

Theorem 3.1 (Forgetful Determinacy). *In any parity game the set of positions can be partitioned into two sets W_0 and W_1 such that Player 0 has a positional winning strategy on W_0 and Player 1 has a positional winning strategy on W_1.*

We call W_0 and W_1 the *winning sets* of Player 0 and, respectively, Player 1 and the pair (W_0, W_1) the *winning partition* or *solution* of $\mathcal{G}$. Since positional strategies are small objects and since it can be checked efficiently whether a strategy is winning, the question whether a given position is winning for Player 0 can be decided in NP $\cap$ co-NP. In fact, it is known [12] that the problem is in UP $\cap$ co-UP. The best known deterministic algorithms to compute winning partitions of parity games have running times that are polynomial with respect to the size of the game graph, but exponential with respect to the index of the game [13].

Theorem 3.2. *The winning partition of a parity game $\mathcal{G} = (V, V_0, E, \Omega)$ of index d can be computed in space $O(d \cdot |E|)$ and time*

$$O\Big(d \cdot |E| \cdot \Big(\frac{|V|}{\lfloor d/2 \rfloor}\Big)^{\lfloor d/2 \rfloor}\Big).$$

Consider a structure $\mathfrak{A}$ and a LFP-sentence ψ which we may assume to be in negation normal form, without parameters, and *well-named*, in the sense that every fixed point variable is bound only once.

The model checking game $\mathcal{G}(\mathcal{A}, \psi)$ is a parity game whose positions are formulae $\varphi(\boldsymbol{a})$ such that $\varphi(\boldsymbol{x})$ is a subformula of ψ, and $\boldsymbol{a}$ is a tuple of elements of $\mathfrak{A}$, interpreting the free variables of φ. The initial position is ψ.

Player 0 (Verifier) moves at positions associated to disjunctions and to formulae starting with an existential quantifier. From a position $\varphi \vee \vartheta$ she moves to either φ or ϑ and from a position $\exists y\, \varphi(\boldsymbol{a}, y)$ Verifier can move to any position $\varphi(\boldsymbol{a}, b)$ for $b \in \mathfrak{A}$. In addition, Verifier is supposed to move at atomic false positions, i.e., at positions φ of form $a = a'$, $a \neq a'$, $R\boldsymbol{a}$, or $\neg R\boldsymbol{a}$ (where R is *not* a fixed point variable) such that $\mathfrak{A} \models \neg\varphi$. However, positions associated with literals do not have successors, so Verifier loses at atomic false positions. Dually, Player 1 (Falsifier) moves at conjunctions and universal quantifications, and loses at atomic true positions. In addition, there are positions associated with fixed point formulae and with fixed points atoms. At these positions there is a unique move (by Falsifier, say) to the formula defining the fixed point. For a more formal definition, recall that as ψ is well-named, for any fixed point variable T in ψ there is a unique subformula $[\mathbf{fp}\, T\boldsymbol{x}\,.\,\varphi(T,\boldsymbol{x})](\boldsymbol{a})$. From position $[\mathbf{fp}\, T\boldsymbol{x}\,.\,\varphi(T,\boldsymbol{x})](\boldsymbol{a})$ Falsifier moves to $\varphi(T,\boldsymbol{a})$, and from $T\boldsymbol{b}$ she moves to $\varphi(T,\boldsymbol{b})$.

The priority labelling assigns even priorities to **gfp**-atoms and odd priorities to **lfp**-atoms. Further, if T, T' are fixed point variables of different kind with T' depending on T (which means that T occurs free in the formula defining T'), then T-atoms get lower priority than T'-atoms. All remaining positions, not associated with fixed point variables, receive highest priority. As a result, the number of priorities in the model checking games corresponds to the alternation depth of the fixed point formula. For more details and explanations, and for the proof that the construction is correct, see e.g. [8, 16].

Proposition 3.3. *Let ψ be an* LFP*-sentence and $\mathfrak{A}$ a relational structure. $\mathfrak{A} \models \psi$ if and only if Player 0 has a winning strategy for the parity game $\mathcal{G}(\mathfrak{A},\psi)$.*

For sentences ψ of width k, the game $\mathcal{G}(\mathfrak{A},\psi)$ can be constructed in linear time with regard to its size $O(|A|^k \cdot \psi)$. According to Theorem 3.2, we obtain the following complexity bounds for model checking LFP via the associated parity game.

Theorem 3.4. *For a finite structure $\mathfrak{A}$ and an* LFP*-sentence ψ of width k and alternation depth d, the model checking problem can be solved in space $O(d \cdot |A|^k \cdot |\psi|)$ and time*

$$O\left(d^2 \cdot \left(\frac{|A|^k \cdot |\psi|}{\lfloor (d+1)/2 \rfloor}\right)^{\lfloor (d+3)/2 \rfloor}\right).$$

Note that if both the alternation depth and the width of the formulae are bounded, the algorithm runs in polynomial time. As mentioned above, the model checking problem is EXPTIME-complete for formulae of unbounded width, even if there is only one application of an LFP-operator. The important unresolved case concerns LFP-formulae with bounded width, but unbounded alternation depth. This includes the μ-calculus, since every formula of L_μ can be translated into an equivalent LFP-formula of width two. In fact the following three problems are algorithmically equivalent, in the sense that if one of them admits a polynomial-time algorithm, then all of them do.

(1) Computing winning sets in parity games.

(2) The model checking problem for LFP-formulae of width at most k, for any $k \geq 2$.

(3) The model checking problem for the modal μ-calculus.

4 Solitaire games

The so far unresolved question whether fixed point logics admit efficient model checking algorithms, and the correspondence between parity games and fixed point logics suggests that one may identify algorithmically simple fragments of fixed point logics by studying games of restricted shape that can be solved efficiently. A promising example for the effectivity of this direction is the correspondence between alternation-free formulae and dull games. To define these games we call a cycle in a game graph even (or odd) if the least priority occurring on it is so.

Definition 4.1. A parity game is *dull* if even and odd cycles are disjoint. A game is called *weak* if priorities cannot decrease along transitions.

Weak games and dull games are closely related notions. Observe that every weak game is also dull. Conversely, any dull game can be transformed in linear time into an equivalent weak game, by changing only the priorities, not the game graph. Kupferman, Vardi, and Wolper [14] established (using different terminology) that dull games can be solved in linear time and that they emerge as model checking games for alternation free L_μ-formulae. Actually, dull games correspond in general to the alternation free fragment of LFP (see [2]). As a consequence, the problem of checking a model $\mathfrak{A}$ against an alternation free LFP-formula ψ of width k can be solved in time $O(|\psi| \cdot |A|^k)$. If ψ is a formula of the modal μ-calculus or guarded fixed point logic, the complexity is $O(|\psi| \cdot \|\mathfrak{A}\|)$.

Instead of restricting reachable priorities, we can take a different approach to render games easy, namely by restricting the interaction between the players. The simplest case is given by solitaire games.

Definition 4.2. A parity game is called *solitaire* if all nontrivial moves are performed by the same player.

In a soltaire game where only Player 0 makes nontrivial moves, his winning set consists of those positions from which a terminal position in V_1 or an even cycle is reachable. Consequently, each strongly connected component of a solitaire game is completely included in the winning set of one of the players.

The positions from which terminal positions in V_1 are reachable, can be computed in linear time using depth-first search. Hence, we may restrict our attention to games $\mathcal{G}$ without terminal positions. To establish the components from which an even cycle is reachable in such a game we distinguish two cases.

In the simplest setting, when only priorities 0 and 1 occur in $\mathcal{G}$, the winning set of Player 0 is the set of nodes from which a nontrivial strongly connected component containing at least one position of priority 0 is reachable. By partitioning the game graph into its strongly connected components the solution of $\mathcal{G}$ can be computed in linear time.

Games of higher index can be solved by reduction to several instances of games of the above kind. For every even priority i occurring in the game $\mathcal{G}$, let $\mathcal{G}_i$ be the restriction of $\mathcal{G}$ to positions of priority $j \geq i$ where the priority i is replaced by 0 and all positions of priority $j > i$ receive priority 1. Note that, if Player 0 wins from a position v in some game $\mathcal{G}_i$, he also wins from v in the original game $\mathcal{G}$. Conversely, Player 1 wins in $\mathcal{G}$ only if he can reach a winning position v in some $\mathcal{G}_i$. Hence, we can solve $\mathcal{G}$ by first computing the winning positions of Player 0 for each $\mathcal{G}_i$. The winning set of Player 0 in $\mathcal{G}$ comprises all strongly connected components from which one of these winning positions is reachable.

To summarise, our method involves two reachability tests, one at the beginning, to handle terminal positions, and one at the end; between these, for every even priority, the solution of a solitaire game with only two priorities is computed. Each of these steps requires only linear time with respect to the size of the game graph.

Theorem 4.3. *The solution of a solitaire game $\mathcal{G} = (V, V_0, E, \Omega)$ of index d can be computed in time $O\big(d \cdot (|V| + |E|)\big)$.*

A significant feature of parity games in general is that their main complexity resides in strongly connected components. Indeed, as pointed out in [2], the partial solutions of subgames induced in a game by strongly connected components can be propagated to obtain the global solution with only linear overhead.

Definition 4.4. A parity game is called *nested solitaire* if each strongly connected component induces a solitaire game.

Observe that in a nested solitaire game both players may perform nontrivial moves.

To solve a nested solitaire game $\mathcal{G}$ we can proceed as follows. First, we decompose the game graph into its strongly connected components. Note that in any terminal component C, that is, a strongly connected component with no outgoing edges, a position is winning in $\mathcal{G}$ iff it is also winning in the subgame induced by C. As a solitaire game, this subgame can be solved efficiently, providing a partial solution for the winning sets W_0 and W_1 in $\mathcal{G}$.

Next, we extend the obtained partial solution by assigning to W_0 the positions $v \in V_0$ with some successor already in W_0, and $v \in V_1$ with all successors already in W_0; the partial solution for W_1 propagates dually. Let $\mathcal{G}'$ be the subgame induced by the positions that remained unassigned after the propagation process. In case there are no such positions, we are done. Otherwise, we reiterate the procedure for $\mathcal{G}'$, which is again a nested solitaire game.

Since the overhead caused by partitioning the game into its strongly connected components and by the propagation of solutions is only linear, nested solitaire games are as easy to solve as non-nested ones.

Theorem 4.5. *A nested solitaire game $\mathcal{G} = (V, V_0, E, \Omega)$ of index d can be solved in time $O\big(d \cdot (|V| + |E|)\big)$.*

Notice that any positional strategy can be presented as a solitaire game. In the automata-theoretic view a solitaire game corresponds to a *deterministic* parity tree automaton whose emptiness problem is linear time reducible to the nonemptiness problem of a one-letter nondeterministic parity word automaton.

5 Solitaire formulae in least fixed point logic

Given that nested solitaire games can be treated efficiently, the question arises whether these games correspond to a natural fragment of fixed point logic. Note that in a model checking game, Player 0 makes choices at positions corresponding to disjunctions or existential quantifications, whereas Player 1 makes nontrivial choices at conjunctions and universal quantifications. Hence we obtain solitaire model checking games for formulae where either $\wedge$ and $\forall$ (or, equivalently, $\vee$ and $\exists$) do not appear, and negations are only applied to atomic formulae. However, these formulae are of very limited expressive power.

In order to understand which formulae lead to nested solitaire games, observe that all cycles in model checking games arise by regeneration of fixed point variables. Thus, to guarantee that a nontrivial move $\varphi \to \varphi'$ leaves the current strongly connected component, we have to ensure that φ and φ' do not depend on a common fixed point variable. But according to the rules of the game, whenever a fixed point variable is free in φ' it will also be free in φ. In contrast, if φ' has no free fixed point variables then the position φ will not be reachable from φ'.

Recall that a LFP-formula is called closed if it does not contain free fixed point variables.

Definition 5.1. The *solitaire fragment* of LFP, denoted Solitaire-LFP, consists of those formulae where negation and universal quantification apply to closed formulae only, and conjunctions to pairs of formulae of which at least one is closed.

Note that the above definition is not closed under transformation of formulae into negation normal form. However, when speaking of a solitaire formula ψ we will tacitly assume that ψ is the presentation in negation normal form of a formula complying with the above definition. Under this proviso, a straightforword induction over closed subformulae shows that Solitaire-LFP corresponds indeed to nested solitaire games.

Proposition 5.2. *The model checking games associated with formulae in* Solitaire-LFP *are nested solitaire games.*

We remark that the solitaire fragment of L_μ has been studied under the name L_2 in [6].

5.1 Complexity

As the model checking games of Solitaire-LFP are nested solitaire, we obtain the following deterministic complexity bound as a direct consequence of Theorem 4.5.

Proposition 5.3. *The model checking problem for a structure $\mathfrak{A}$ and a solitaire* LFP*-sentence ψ of width k and alternation depth d can be solved in time $O\left(d \cdot |\psi| \cdot |A|^k\right)$.*

In terms of major complexity classes, the following results can be established.

Theorem 5.4. *The model checking problem for* Solitaire-LFP *is* PSPACE*-complete with respect to expression and combined complexity and* NLOGSPACE*-complete with respect to data complexity.*

Proof. The lower bounds follow by the efficient translation of TC into Solitaire-LFP. For the upper bounds, we present a recursive nondeterministic procedure Eval($\mathfrak{A}, \psi$) which, given a structure $\mathfrak{A}$, and a closed (instantiated) Solitaire-LFP-formula ψ, decides whether $\mathfrak{A} \models \psi$. For convenience, let us assume that in a conjunction the first formula is always closed. Further, let $G(\psi)$ denote the set of **gfp**-variables in ψ.

```
function Eval($\mathfrak{A}, \psi$)
guess  a formula witness $\in \{R\boldsymbol{a} : R \in G(\psi)\} \cup \{\bot\}$
seen_witness := $false$
repeat
  if $\psi$ is an atom then
    return $\mathfrak{A} \models \psi$
  if $\psi = \neg\vartheta$ then
    return $\neg$Eval($\mathfrak{A}, \vartheta$)    (* $\vartheta$ is a closed formula *)
  if $\psi = \varphi_1 \vee \varphi_2$ then
    guess  $i \in \{1, 2\}$; $\psi := \varphi_i$
  if $\psi = \vartheta \wedge \varphi$ then
    if $\neg$Eval($\mathfrak{A}, \vartheta$) then  return $false$ else  $\psi := \varphi$
  if $\psi = \exists x\varphi$ then
    guess  $b \in A$; $\psi := \varphi[x \mapsto b]$
  if $\psi = \forall x\varphi$ then
    $r := true$
    forall  $b \in A$ do $r := r \wedge$ Eval($\mathfrak{A}, \varphi[x \mapsto b]$)
    return $r$
```

```
if $\psi = [\mathbf{fp}\, T\boldsymbol{x}.\varphi](\boldsymbol{c})$ then
  $\psi := \varphi(\boldsymbol{c})$
if $\psi = T\boldsymbol{a}$ (a fixed point atom) then
  if ($\neg$seen_witness $\wedge\, T\boldsymbol{a} =$ witness) then seen_witness := true
  if (seen_witness $\wedge\, T\boldsymbol{a} =$ witness) then return true    (* even cycle detected *)
  if (seen_witness $\wedge\, T$ does not depend on witness) then return false
$\psi := \varphi_T(\boldsymbol{c})$ (where $\varphi_T$ is the formula defining $T$)
```

Up to the handling of fixed points, this algorithm is a variant of a common method for first order evaluation. The correctness is proved by induction; the only interesting cases are fixed point variables.

We can argue in terms of the model checking game. Assume that Verifier has a strategy to prove $\mathfrak{A} \models \psi$. Then, at the starting position of the game $\mathcal{G}(\mathfrak{A}, \psi)$, he can ensure that the play either reaches an even cycle, or it descends into another component. The latter case is covered by the induction hypothesis. For the former, let $R\boldsymbol{a}$ be a position of lowest priority on the cycle. Guessing this position as a witness at the first step of the algorithm will lead to acceptance (even cycle detected). For the other direction, if $\mathfrak{A} \not\models \psi$, every cycle containing a (cheating) witness $R\boldsymbol{a}$ also containes positions of lower priority, i.e., associated to variables that do not depend on R. Thus, R cannot be regenerated whithout regenerating one of these first, at which point the procedure rejects.

Now, let us consider the space requirement. During the evaluation of a formula ψ of width k, the recursion depth is bounded by $|\psi|$ and at each level a pointer to ψ together with an assignment consisting of k pointers to elements of A is stored. Hence, the algorithm requires space $O(|\psi| \cdot k \log |A|)$. □

5.2 Expressive power

Observe that the translation from transitive closure logic into LFP given in Section 2 involves only solitaire formulae. Consequently, Solitaire-LFP subsumes TC.

Lemma 5.5. TC $\subseteq$ Solitaire-LFP.

It follows from well-known results that the converse is not true in general. A simple example is the solitaire formula $[\mathbf{gfp}\, Tx \,.\, \exists y(Exy \wedge Ty)](x)$ expressing that there is an infinite path from x. It is known that this query is not even expressible in the infinitary logic $L_{\infty\omega}$ (otherwise, well-founded linear orders were axiomatizable in $L_{\infty\omega}$). Even restricted to countable structures this query is not expressible in TC. However, on finite structures the converse does hold.

Theorem 5.6. *On finite structures,* Solitaire-LFP $\equiv$ TC.

To prove this, we exploit the solitaire-structure of the model checking game and the fact that TC-formulae are equivalent to stratified linear Datalog programs. For every formula in $\psi \in$ Solitaire-LFP we construct a stratified linear Datalog program Π_ψ which defines the winning positions in the associated model checking game.

More precisely, the construction proceeds by induction along the following lines:

(a) For every subformula $\varphi(\boldsymbol{x})$, we introduce a head predicate W_φ. On any finite structure $\mathfrak{A}$, the program evaluates the atom $W_\varphi(\boldsymbol{a})$ to true if, and only if, Verifier has a winning strategy from position $\varphi(\boldsymbol{a})$.

(b) Further, the program contains auxiliary head predicates $R_{\varphi T}$, for each **gfp**-formula $[\mathbf{gfp}\ T\boldsymbol{x}\,.\,\vartheta](\boldsymbol{x})$ in ψ and every subformula φ depending on T. On any finite $\mathfrak{A}$, the program evaluates $R_{\varphi T}(\boldsymbol{a},\boldsymbol{b})$ to true if, and only if, in the game $\mathcal{G}(\mathfrak{A},\psi)$ Verifier has a strategy from position $\varphi(\boldsymbol{a})$ to either win the game without seeing position $\varphi(\boldsymbol{a})$ again, or to reach the position $\vartheta(\boldsymbol{b})$ without passing through any position of priority less than $\Omega(T)$. Note that $R_{\vartheta T}(\boldsymbol{a},\boldsymbol{a})$ implies that Verifier has a strategy to reach a cycle of minimal priority $\Omega(T)$ (or, to win by other means).

We remark that the following construction is standard up to the treatment of the **gfp**-formulae via the reachability predicates $R_{\varphi T}$.

(1) If $\varphi(\boldsymbol{x})$ is a literal $(\neg)P\boldsymbol{x}$, the program Π_φ consists of the single rule

$$W_\varphi(\boldsymbol{x}) \leftarrow \varphi(\boldsymbol{x})$$

(2) For $\varphi = \neg\eta$, Π_φ is obtained by adding to Π_ϑ a new stratum with the rule

$$W_\varphi(\boldsymbol{x}) \leftarrow \neg W_\eta(\boldsymbol{x}).$$

(Note that this is well-defined since η does not contain free fixed-point variables.)

(3) For $\varphi = \eta \vee \vartheta$, the program Π_φ consists of $\Pi_\eta \cup \Pi_\vartheta$ together with the rules

$$W_\varphi(\boldsymbol{x}) \leftarrow W_\eta(\boldsymbol{x}); \quad W_\varphi(\boldsymbol{x}) \leftarrow W_\vartheta(\boldsymbol{x})$$

and, if applicable,

$$R_{\varphi T}(\boldsymbol{x},\boldsymbol{y}) \leftarrow W_\varphi(\boldsymbol{x}); \quad R_{\varphi T}(\boldsymbol{x},\boldsymbol{y}) \leftarrow R_{\eta T}(\boldsymbol{x},\boldsymbol{y}); \quad R_{\varphi T}(\boldsymbol{x},\boldsymbol{y}) \leftarrow R_{\vartheta T}(\boldsymbol{x},\boldsymbol{y}).$$

(4) For $\varphi = \eta \wedge \vartheta$, we can assume that ϑ does not contain free fixed point variables. Now Π_φ is $\Pi_\eta \cup \Pi_\vartheta$ augmented with the rules

$$W_\varphi(\boldsymbol{x}) \leftarrow W_\eta(\boldsymbol{x}) \wedge W_\vartheta(\boldsymbol{x})$$

and, if applicable,

$$R_{\varphi T} \leftarrow W_\varphi(\boldsymbol{x}); \quad R_{\varphi T}(\boldsymbol{x},\boldsymbol{y}) \leftarrow R_{\eta T}(\boldsymbol{x},\boldsymbol{y}) \wedge W_\vartheta(\boldsymbol{x}).$$

Note that W_ϑ does not depend on W_φ, so the program is indeed linear.

(5) For $\varphi = \exists z\eta(\boldsymbol{x},z)$, we obtain Π_φ by adding to Π_η the rules

$$W_\varphi(\boldsymbol{x}) \leftarrow W_\eta(\boldsymbol{x},z); \quad R_{\varphi T}(\boldsymbol{x},\boldsymbol{y}) \leftarrow R_{\eta T}(\boldsymbol{x}z,\boldsymbol{y})$$

for appropriate **gfp**-variables T.

(6) For $\varphi = \forall z\eta(\boldsymbol{x},z)$, the subformula η does not contain free fixed point variables. We construct Π_φ by adding to Π_η the rules

$$L_\varphi(\boldsymbol{x}) \leftarrow \neg W_\eta(\boldsymbol{x},z); \quad W_\varphi(\boldsymbol{x}) \leftarrow \neg L_\varphi(\boldsymbol{x}).$$

(7) For $\varphi = [\mathbf{lfp}\ T\boldsymbol{x} \,.\, \vartheta](\boldsymbol{x})$, we construct Π_φ by adding to Π_ϑ the rules

$$W_\varphi(\boldsymbol{x}) \leftarrow W_\vartheta(\boldsymbol{x}); \quad T(\boldsymbol{x}) \leftarrow W_\vartheta(\boldsymbol{x})$$

and for, all **gfp**-variables T' that depend on T (and, hence, have priority $\Omega(T') > \Omega(T)$),

$$R_{\varphi T'}(\boldsymbol{x}, \boldsymbol{y}) \leftarrow R_{\vartheta T'}(\boldsymbol{x}, \boldsymbol{y}); \quad R_{TT'}(\boldsymbol{x}, \boldsymbol{y}) \leftarrow R_{\vartheta T'}(\boldsymbol{x}, \boldsymbol{y})$$

(8) Finally, for $\varphi = [\mathbf{gfp}\ T\boldsymbol{x} \,.\, \vartheta](\boldsymbol{x})$, we construct Π_φ by adding to Π_ϑ the rules

$$W_\varphi(\boldsymbol{x}) \leftarrow W_\vartheta(\boldsymbol{x}); \quad W_\varphi(\boldsymbol{x}) \leftarrow R_{\vartheta T}(\boldsymbol{x}, \boldsymbol{x})$$

and, for all **gfp**-variables T' that depend on T, including T itself, (hence, $\Omega(T') \geq \Omega(T)$),

$$R_{TT}(\boldsymbol{x}, \boldsymbol{x}); \quad R_{\varphi T'}(\boldsymbol{x}, \boldsymbol{y}) \leftarrow R_{\vartheta T'}(\boldsymbol{x}, \boldsymbol{y}); \quad R_{TT'}(\boldsymbol{x}, \boldsymbol{y}) \leftarrow R_{\vartheta T'}(\boldsymbol{x}, \boldsymbol{y})$$

It is readily seen that the solitaire structure of ψ implies that Π_ψ is indeed a linear stratified program. It remains to prove the following.

Lemma 5.7. *For every solitaire formulae $\psi(\boldsymbol{x})$ and every finite structure $\mathfrak{A}$ we have that $\mathfrak{A} \models \psi(\boldsymbol{a})$ iff Π_ψ evaluates on $\mathfrak{A}$ the atom $W_\psi(\boldsymbol{a})$ to true.*

Proof. We show that the truth values for $W_\varphi(\boldsymbol{a})$ and $R_{\varphi T}(\boldsymbol{a}, \boldsymbol{b})$ defined by Π_ψ on $\mathfrak{A}$ indeed have the game theoretic meaning described by items (a) and (b) above.

A winning play for Verifier in $\mathcal{G}(\psi, \mathfrak{A})$ from a position $\varphi(\boldsymbol{a})$ must either lead in finitely many steps to a literal $(\neg)P\boldsymbol{b}$ that is true in $\mathfrak{A}$, or it must lead to a **gfp**-atom $T\boldsymbol{b}$ from which it cycles without hitting any priority smaller than $p(T)$. It is not difficult to see that the rules of Π_ψ ensure a strategy for precisely this.

The rules for cases (1) to (6) reduce the winning conditions W_φ and the reachability conditions $R_{\varphi T}$ in the obvious way to the immediate subformulae of φ (i.e. to the positions after the next move).

For $\varphi = [\mathbf{lfp}\, T\boldsymbol{x}, ., \vartheta](\boldsymbol{x})$ the rules do the same, and in addition, they take into account the moves from $T\boldsymbol{a}$ back to $\vartheta(\boldsymbol{a})$. To win from $T\boldsymbol{a}$, Verifier must win from $\vartheta(\boldsymbol{a})$. If T is regenerated infinitely often the play is lost (unless a **gfp**-variable of smaller priority is also regenerated infinitely often).

This leaves the case of formulae $\varphi(\boldsymbol{x}) = [\mathbf{gfp}\, T\boldsymbol{x} \,.\, \vartheta](\boldsymbol{x})$. Besides reducing the winning condition W_φ and the reachability conditions $R_{\varphi T'}$ to W_ϑ and $R_{\vartheta T'}$, respectively, the rules of Π_φ take care of the back-moves from $T\boldsymbol{a}$ to $\vartheta(\boldsymbol{a})$, and, most importantly, the possibility to win by forcing an appropriate cycle. By the rule $W_\varphi(\boldsymbol{x}) \leftarrow R_{\vartheta T}(\boldsymbol{x}, \boldsymbol{x})$ it follows that $W_\varphi(\boldsymbol{b})$ is evaluated to true if Verifier can force a cycle that contains $\vartheta(\boldsymbol{b})$ and on which no priority is smaller than $\Omega(T)$. Together with the other rules this further implies that Verifier also wins from positions where she can force a play that eventually hits such a cycle. $\square$

This completes the proof of Theorem 5.6.

References

[1] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*, Addison-Wesley, 1995.

[2] D. Berwanger and E. Grädel, *Games and model checking for guarded logics*, in Proceedings of LPAR 2001, Lecture Notes in Computer Science Nr. 2250, Springer, 2001, pp. 70–84.

[3] J. Bradfield, *The modal μ-calculus alternation hierarchy is strict*, Theoretical Computer Science, 195 (1998), pp. 133–153.

[4] H.-D. Ebbinghaus and J. Flum, *Finite Model Theory*, Springer, 2nd edition ed., 1999.

[5] A. Emerson and C. Jutla, *Tree automata, mu-calculus and determinacy*, in Proc. 32nd IEEE Symp. on Foundations of Computer Science, 1991, pp. 368–377.

[6] A. Emerson, C. Jutla, and P. Sistla, *On model checking for the μ-calculus and its fragments*, Theoretical Computer Science, 258 (2001), pp. 491–522.

[7] E. Grädel, *On transitive closure logic*, in Proceedings of 5th Workshop on Computer Science Logic CSL 91, Bern 1991, vol. 626 of LNCS, Springer-Verlag, 1991, pp. 149–163.

[8] E. Grädel, *Model checking games*, in Proceedings of WOLLIC 02, Electronic Notes in Theoretical Computer Science, Elsevier, 2002.

[9] E. Grädel and M. Otto, *On logics with two variables*, Theoretical Computer Science, 224 (1999), pp. 73–113.

[10] N. Immerman, *Relational queries computable in polynomial time*, Information and Control, 68 (1986), pp. 86–104.

[11] N. Immerman, *Languages that capture complexity classes*, SIAM Journal on Computing, 16 (1987), pp. 760–778.

[12] M. Jurdziński, *Deciding the winner in parity games is in UP $\cap$ Co-UP*, Information Processing Letters, 68 (1998), pp. 119–124.

[13] M. Jurdziński, *Small progress measures for solving parity games*, in STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Proceedings, vol. 1770 of Lecture Notes in Computer Science, Springer, 2000, pp. 290–301.

[14] O. Kupferman, M. Vardi, and P. Wolper, *An automata-theoretic approach to branching-time model checking*, Journal of the ACM, 47 (2000), pp. 312–360.

[15] Y. Moschovakis, *Elementary induction on abstract structures*, North Holland, 1974.

[16] C. Stirling, *Bisimulation, model checking and other games.* Notes for the Mathfit instructional meeting on games and computation. Edinburgh, 1997.

[17] M. Vardi, *The complexity of relational query languages*, in Proceedings of the 14th ACM Symposium on the Theory of Computing, 1982, pp. 137–146.

Finite Presentations of Infinite Structures: Automata and Interpretations

Achim Blumensath[*] Erich Grädel[*]

Abstract

We study definability problems and algorithmic issues for infinite structures that are finitely presented. In particular we focus on structures presented by automata or by model-theoretic interpretations.

1 Computational Model Theory

The relationship between logical definability and computational complexity is an important issue in a number of different fields including finite model theory, databases, knowledge representation, and computer-aided verification. So far most of the research has been devoted to finite structures where the relationship between definability and complexity is by now fairly well understood (see e.g. [14, 26]) and has many applications in particular to database theory [1]. However, in many cases the limitation to finite structures is too restrictive. Therefore in most of the fields mentioned above, there have been considerable efforts to extend the methodology from finite structures to suitable classes of infinite ones. In particular, this is the case for databases and computer-aided verification where infinite structures (like constraint databases or systems with infinite state spaces) are of increasing importance.

Computational model theory extends the research programme, the general approach and the methods of finite model theory to interesting domains of infinite structures. From a general theoretical point of view, one may ask what domains of infinite structures are suitable for such an extension. More specifically, what conditions must be satisfied by a domain $\mathcal{D}$ of not necessarily finite structures such that the approach and methods of finite model theory make sense. There are two obvious and fundamental conditions:

Finite representations. Every structure $\mathfrak{A} \in \mathcal{D}$ should be representable in a finite way (e.g. by a binary string, by an algorithm, by a collection of automata, by an axiomatisation in some logic, by an interpretation ...).

Effective semantics. For the relevant logics to be considered (e.g. first-order logic), the model checking problem on $\mathcal{D}$ should be decidable. That is, given a sentence $\psi \in L$ and a representation of a structure $\mathfrak{A} \in \mathcal{D}$, it should be decidable whether $\mathfrak{A} \models \psi$.

These are just minimal requirements, that may need to be refined according to the context and the questions to be considered. We may for instance also require:

[*]Aachen University of Technology, Mathematical Foundations of Computer Science, D-52065 Aachen, {blume,graedel}@informatik.rwth-aachen.de, www-mgi.informatik.rwth-aachen.de

Closure. For every structure $\mathfrak{A} \in \mathcal{D}$ and every formula $\psi(\bar{x})$, also $(\mathfrak{A}, \psi^{\mathfrak{A}})$, the expansion of $\mathfrak{A}$ with the relation defined by ψ, belongs to $\mathcal{D}$.

Effective query evaluation. Suppose that we have fixed a way of representing structures. Given a representation of $\mathfrak{A} \in \mathcal{D}$ and a formula $\psi(\bar{x})$ we should be able to compute a representation of $\psi^{\mathfrak{A}}$ (or of the expanded structure $(\mathfrak{A}, \psi^{\mathfrak{A}})$).

Note that contrary to the case of finite structures, query evaluation does not necessarily reduce to model checking. Further, instead of just effectiveness of these tasks, it may be required that they can be performed within some complexity bounds.

After giving a brief survey on different classes of finitely presented structures in the next section, we will focus on domains where structures are presented by two closely related methods, namely by finite *automata* or by model-theoretic *interpretations.* While *automatic groups* have been studied rather intensively in computational group theory (see [15, 17]) a general notion of *automatic structures* has only been defined in [28], and their theory has been developed in [5, 7]. These structures will be defined in Section 3. Informally, a relational structure $\mathfrak{A} = (A, R_1, \dots, R_m)$ is automatic if its universe and its relations can be recognised by finite automata reading their inputs synchronously. We believe that automatic structures are very promising for the approach of computational model theory. Not only do automatic structures admit finite presentations, there also are numerous interesting examples and a large body of methods that has been developed in five decades of automata theory. Further, automatic structures admit effective evaluation of all first-order queries and possess many other pleasant algorithmic properties.

Automatic structures can also be defined via interpretations. As we show in Section 4 a structure is automatic if, and only if, it is first-order interpretable in an appropriate expansion of Presburger arithmetic or, equivalently, in the infinite binary tree with prefix order and equal length predicate. Similar results hold for ω-automatic structures and appropriate expansions of the real ordered group.

Such results suggest a very general way for obtaining other interesting classes of infinite structures suitable for the approach of computational model theory: Fix a structure $\mathfrak{A}$ (or a class of such structures) with 'nice' algorithmic and/or model-theoretic properties, and consider the class of all structures that are interpretable in $\mathfrak{A}$, for instance via first-order or monadic second-order logic. Obviously each structure in this class is finitely presentable (by an interpretation). Further, since many 'nice' properties are preserved under interpretations, every structure in the class inherits them from $\mathfrak{A}$. In particular, every class of queries that is effective on $\mathfrak{A}$ and closed under first-order operations is effective on the interpretation-closure of $\mathfrak{A}$.

In Section 5 we turn to decidability and complexity issues. It is shown that the model checking problem for FO($\exists^\omega$), first-order logic extended by the quantifier "there are infinitely many", is decidable for automatic and ω-automatic structures, and the complexity for various fragments of first-order logic is investigated. On the other hand, we prove that several properties not expressible in FO, such as isomorphism of automatic structures, are undecidable.

In the final section, Feferman-Vaught like products are introduced, and it is shown that every domain which can be characterised via interpretations of a certain kind is closed under such products.

2 Finitely presentable structures

We briefly survey some domains of infinite, but finitely presentable structures which may be relevant for computational model theory.

Recursive structures are countable structures whose functions and relations are computable and therefore finitely presentable. They have been studied quite intensively in model theory since the 1960s (see e.g. [2, 16]). Although recursive model theory is very different from finite model theory, there have been some papers studying classical issues of finite model theory on recursive structures and recursive databases [21, 24, 25, 35]. However, for most applications, the domain of recursive structures is far too large. In general, only quantifier-free formulae admit effective evaluation algorithms.

Constraint databases are a modern database model admitting infinite relations that are finitely presented by quantifier-free formulae (constraints) over some fixed background structure. For example, to store geometrical data, it is useful to have not just a finite set as the universe of the database, but to include all real numbers 'in the background'. Also the presence of interpreted functions, like addition and multiplication, is desirable. The constraint database framework introduced by Kanellakis, Kuper and Revesz [27] meets both requirements. Formally, a constraint database consists of a *context structure* $\mathfrak{A}$, like $(\mathbb{R}, <, +, \cdot)$, and a set $\{\varphi_1, \ldots, \varphi_m\}$ of quantifier-free formulae defining the database relations. Constraint databases are treated in detail in [29].

Metafinite structures are two-sorted structures consisting of a finite structure $\mathfrak{A}$, a background structure $\mathfrak{R}$ (which is usually infinite, but fixed) and a class of weight functions from the finite part to the infinite one. Simple examples are finite graphs whose edges are weighted by real numbers. For any fixed infinite structure $\mathfrak{R}$, the metafinite structures with background $\mathfrak{R}$ are finitely presentable and admit effective evaluation of logics that make use of arithmetic operations on $\mathfrak{R}$, but do not admit full quantification over its elements. Metafinite model theory has been developed in [20] and has been put to use for studying issues in database theory, optimisation and descriptive complexity. In particular metafinite structures have provided the basis for logical characterisations of complexity classes over the real numbers [22].

Automatic structures are structures whose functions and relations are represented by finite automata. Informally, a relational structure $\mathfrak{A} = (A, R_1, \ldots, R_m)$ is automatic if we can find a regular language $L_\delta \subseteq \Sigma^*$ (which provides names for the elements of $\mathfrak{A}$) and a function $\nu : L_\delta \to A$ mapping every word $w \in L_\delta$ to the element of $\mathfrak{A}$ that it represents. The function ν must be surjective (every element of $\mathfrak{A}$ must be named) but need not be injective (elements can have more than one name). In addition it must be recognisable by finite automata (reading their input words synchronously) whether two words in L_δ name the same elements, and, for each relation R_i of $\mathfrak{A}$, whether a given tuple of words in L_δ names a tuple in R_i. Automatic structures provide many examples of high relevance for computer science. There are also interesting connections to computational group theory, where *automatic groups* have already been studied quite intensively [15, 17]. The general notion of structures presentable by automata has been proposed in [28] and their theory has been developed in [5, 7].

The notion of an automatic structure can be modified and generalised in many directions. By using automata over infinite words, we obtain the notion of **ω-automatic structures** (which, contrary to automatic structures, may have uncountable cardinality). Contrary to the class of recursive structures, automatic and ω-automatic structures admit effective (in fact, automatic) evaluation of all first-order queries.

Theorem 2.1. *The model checking problem for,* $\mathrm{FO}(\exists^\omega)$*, first-order logic extended by the quantifier "there are infinitely many", is decidable on the domain of ω-automatic structures.*

Tree-automatic structures, which are defined by automata on finite or infinite trees, are further natural generalisations of automatic structures. They also admit effective evaluation of first-order formulae. The theory of tree-automatic structures has been developed in [5]. On the other side, first-order logic is *not* effective on another popular extension of automatic graphs, the so-called **rational graphs** [31], which are defined by *asynchronous* multihead automata.

Tree-interpretable structures are structures that are interpretable in the infinite binary tree $\mathcal{T}^2 = (\{0,1\}^*, \sigma_0, \sigma_1)$ via a one-dimensional monadic second-order interpretation (see Section 4 for details on interpretations). By Rabin's Theorem, monadic second-order logic (MSO) can be effectively evaluated on $\mathcal{T}^2$, and since MSO is closed under one-dimensional interpretations, the same holds for all tree-interpretable structures. Tree-interpretable structures generalise various notions of infinite graphs that have been studied in logic, automata theory and verification. Examples are the **context-free graphs** [32, 33], which are the configuration graphs of pushdown automata, the **HR-equational and VR-equational graphs** [11], which are defined via graph grammars, and the **prefix-recognisable graphs** [10] which can for instance be defined as graphs of form $(V, (E_a)_{a\in A})$ where V is a regular language and each edge relation E_a is a finite union of sets $X(Y \times Z) := \{\, (xy, xz) \mid z \in Z,\ y \in Y,\ z \in Z \,\}$, for regular languages X, Y, Z. It has been established in a series of papers that some of these classes coincide with the tree-interpretable graphs (see [3, 6, 10]).

Theorem 2.2. *For any graph $G = (V, (E_a)_{a\in A})$ the following are equivalent:*

(i) *G is tree-interpretable.*

(ii) *G is VR-equational.*

(iii) *G is prefix-recognisable.*

(iv) *G is the restriction to a regular set of the configuration graph of a pushdown automaton with ε-transitions.*

On the other hand the classes of context-free graphs and of HR-equational graphs are strictly contained in the class of tree-interpretable graphs.

The question arises whether there are even more powerful domains than the tree-interpretable structures on which monadic-second order logic is effective. An interesting way to obtain such domains are **tree constructions** that associate with any structure a kind of tree unravelling. A simple variant is the **unfolding** of a labelled graph G from a given node v to the tree $\mathcal{T}(G, v)$. Courcelle and Walukiewicz [12, 13] show that the MSO-theory of $\mathcal{T}(G, v)$ can be effectively computed from the MSO-theory of (G, v). A more general operation, applicable to relational structures of any kind, has been invented by Muchnik. Given a relational structure $\mathfrak{A} = (A, R_1, \dots, R_m)$, let its **iteration** $\mathfrak{A}^* = (A^*, R_1^*, \dots, R_m^*, son, clone)$ be the structure with universe A^*, relations $R_i^* = \{ (wa_1, \dots, wa_r) \mid w \in A^*, (a_1, \dots, a_r) \in R_i \}$, the successor relation $son = \{ (w, wa) \mid w \in A^*, a \in A \}$ and the predicate *clone* consisting of all elements of form waa. It is not difficult to see that unfoldings of graphs are first-order interpretable in their iterations. Muchnik's Theorem states that the monadic theory of $\mathfrak{A}^*$ is decidable if the monadic theory of $\mathfrak{A}$ is so (for proofs, see [4, 37]). Define the domain of **tree-constructible structures** to be be the closure of the domain of finite structures under (one-dimensional) MSO-interpretations and iterations. By Muchnik's Theorem, and since effective MSO model checking is preserved under interpretations, the tree constructible structures are finitely presentable and admit effective evaluation of MSO-formulae. By results of Courcelle [12] every algebraic tree is tree-constructible. Since not all algebraic trees are tree-interpretable it follows that the domain of tree-constructible structures forms a proper extension of the tree-interpretable ones.

Ground tree rewriting graphs are defined by tree rewriting [30]. Vertices are represented by finite trees and edges are generated by ground rewriting rules. In this way one can obtain graphs that are not tree-interpretable (for instance the infinite two-dimensional grid), but for which, in addition to the first-order theory, also the reachability problem remains decidable. While universal reachability and universal recurrence (and hence general MSO formulae) are undecidable on ground tree rewriting graphs, Löding exhibits a fragment of CTL (permitting EF and EGF-operations, but not EG, EFG or until operations) that can be effectively evaluated on this class.

3 Automatic structures and automatic groups

As usual in logic, we consider *structures* $\mathfrak{A} = (A, R_1, R_2, \dots, f_1, f_2, \dots)$ where A is a non-empty set, called the *universe* of $\mathfrak{A}$, where each $R_i \subseteq A^{r_i}$ is a relation on A, and every $f_j : A^{s_j} \to A$ is a function on A. The names of the relations and functions of $\mathfrak{A}$, together with their arities, form the *vocabulary* of $\mathfrak{A}$. We consider constants as functions of arity 0. A *relational* structure is a structure without functions. We can associate with every structure $\mathfrak{A}$ its *relational variant* which is obtained by replacing each function $f : A^s \to A$ by its graph $G_f := \{ (\bar{a}, b) \in A^{s+1} \mid f(\bar{a}) = b \}$.

For a structure $\mathfrak{A}$ and a formula $\varphi(\bar{x})$, let $\varphi^{\mathfrak{A}} := \{ \bar{a} \mid \mathfrak{A} \models \varphi(\bar{a}) \}$ be the relation (or query) defined by φ on $\mathfrak{A}$.

We assume that the reader is familiar with the basic notions of automata theory and regular languages. One slightly nonstandard aspect is that, in order to present a structure by a list of finite automata, we need a notion of regularity not just for languages $L \subseteq \Sigma^*$ but also k-ary relations of words, for $k > 1$. Instead of introducing synchronous multihead automata that take tuples $\bar{w} = (w_1, \dots, w_k)$ of words as inputs and work synchronously on all k components of $\bar{w}$, we reduce the case of higher arities to the unary one by encoding tuples $\bar{w} \in (\Sigma^*)^k$ by a single word $w_1 \otimes \dots \otimes w_k$ over the alphabet $(\Sigma \cup \{\Box\})^k$, called the *convolution* of $w_1, \dots, w_k$. Here $\Box$ is a padding symbol not belonging to Σ. It is appended to some of the words w_i to make sure that all components have the same length. More formally, for $w_1, \dots, w_k \in \Sigma^*$, with $w_i = w_{i1} \cdots w_{i\ell_i}$ and $\ell = \max\{|w_1|, \dots, |w_k|\}$,

$$w_1 \otimes \dots \otimes w_k := \begin{bmatrix} w'_{11} \\ \vdots \\ w'_{k1} \end{bmatrix} \dots \begin{bmatrix} w'_{1\ell} \\ \vdots \\ w'_{k\ell} \end{bmatrix} \in \big((\Sigma \cup \{\Box\})^k\big)^*$$

where $w'_{ij} = w_{ij}$ for $j \leq |w_i|$ and $w'_{ij} = \Box$ otherwise. Now, a relation $R \subseteq (\Sigma^*)^k$ is called *regular*, if $\{ w_1 \otimes \cdots \otimes w_k \mid (w_1, \ldots, w_r) \in R \}$ is a regular language. In the sequel we do not distinguish between a relation on words and its encoding as a language.

Definition 3.1. A relational structure $\mathfrak{A}$ is *automatic* if there exist a regular language $L_\delta \subseteq \Sigma^*$ and a surjective function $\nu : L_\delta \to A$ such that the relation

$$L_\varepsilon := \{ (w, w') \in L_\delta \times L_\delta \mid \nu w = \nu w' \} \subseteq \Sigma^* \times \Sigma^*$$

and, for all predicates $R \subseteq A^r$ of $\mathfrak{A}$, the relations

$$L_R := \{ \bar{w} \in (L_\delta)^r \mid (\nu w_1, \ldots, \nu w_r) \in R \} \subseteq (\Sigma^*)^r$$

are regular. An arbitrary (not necessarily relational) structure is automatic if and only if its relational variant is.

We write AutStr$[\tau]$ for the class of all automatic structures of vocabulary τ. Each structure $\mathfrak{A} \in$ AutStr$[\tau]$ can be represented, up to isomorphism, by a list $\mathfrak{d} = \langle M_\delta,\ M_\varepsilon,\ (M_R)_{R \in \tau} \rangle$ of finite automata that recognise L_δ, L_ε, and L_R for all relations R of $\mathfrak{A}$. When speaking of an *automatic presentation* of $\mathfrak{A}$ we either mean the function $\nu : L_\delta \to A$ or such a list $\mathfrak{d}$. An automatic presentation $\mathfrak{d}$ is called *deterministic* if all its automata are, and it is called *injective* if $L_\varepsilon = \{ (u, u) \mid u \in L_\delta \}$ (which implies that $\nu : L_\delta \to A$ is injective).

Examples. (1) All finite structures are automatic.

(2) Important examples of automatic structures are Presburger arithmetic $(\mathbb{N}, +)$ and its expansions $\mathfrak{N}_p := (\mathbb{N}, +, |_p)$ by the relation

$$x \mid_p y \ :\text{iff}\ \ x \text{ is a power of } p \text{ dividing } y.$$

Using p-ary encodings (starting with the least significant digit) it is not difficult to construct automata recognising equality, addition and $|_p$.

(3) Natural candidates for automatic structures are those consisting of words. (But note that free monoids with at least two generators do *not* have automatic presentations.) Fix some alphabet Σ and consider the structure $\mathit{Tree}(\Sigma) := (\Sigma^*, (\sigma_a)_{a \in \Sigma}, \preceq, \mathrm{el})$ where

$$\sigma_a(x) := xa, \qquad x \preceq y \ :\text{iff}\ \exists z(xz = y), \qquad \text{and} \qquad \mathrm{el}(x, y) \ :\text{iff}\ |x| = |y|.$$

Obviously, this structure is automatic as well.

The following two observations are simple, but useful.

(1) Every automatic structure admits an automatic presentation with alphabet $\{0, 1\}$ [5].

(2) Every automatic structure admits an injective automatic presentation [28].

Automatic Groups. The class of automatic structures that have been studied most intensively are automatic groups. Let $(G, \cdot)$ be a group and $S = \{s_1, \ldots, s_m\} \subseteq G$ a set of semigroup generators of G. This means that each $g \in G$ can be written as a product $s_{i_1} \cdots s_{i_r}$ of elements of S and hence the canonical homomorphism $\nu : S^* \to G$ is surjective. The *Cayley graph* $\Gamma(G, S)$ of G with respect to S is the graph $(G, S_1, \ldots, S_m)$ whose vertices are the group elements and where S_i is the set of pairs (g, h) such that $gs_i = h$. By definition $(G, \cdot)$ is *automatic* if there is a finite set S of semigroup generators and a regular language $L_\delta \subseteq S^*$ such that the restriction of ν to L_δ is surjective and provides an automatic presentation of $\Gamma(G,S)$. (In other words, the inverse image of equality,

$$L_\varepsilon = \{\,(w,w') \in L_\delta \times L_\delta \mid \nu w = \nu w'\,\},$$

and $\nu^{-1}(S_i)$, for $i = 1,\dots,m$, are regular).

Note that it is not the group structure $(G,\cdot)$ itself that is automatic in the sense of Definition 3.1, but the Cayley graph. There are many natural examples of automatic groups (see [15, 17]). The importance of this notion in computational group theory comes from the fact that an automatic presentation of a group yields (efficient) algorithmic solutions for computational problems that are undecidable in the general case.

ω-automatic structures. The notion of an automatic structure can be modified and generalised in a number of different directions (see [5, 28]). In particular, we obtain the interesting class ω-AutStr of ω-automatic structures. The definition is analogous to the one for automatic structures except that the elements of an ω-automatic structure are named by infinite words from some regular ω-language and the relations of the structure are recognisable by Büchi automata.

Examples. (1) All automatic structures are ω-automatic.

(2) The real numbers with addition, $(\mathbb{R},+)$, and indeed the expanded structure $\mathfrak{R}_p := (\mathbb{R},+,\le,|_p,1)$ are ω-automatic, where

$$x \mid_p y \;:\text{iff}\; \exists n,k \in \mathbb{Z}\colon x = p^n \text{ and } y = kx.$$

(3) The tree automatic structures $\mathit{Tree}(\Sigma)$ extend in a natural way to the (uncountable) ω-automatic structures $\mathit{Tree}^\omega(\Sigma) := (\Sigma^{\le\omega}, (\sigma_a)_{a\in\sigma}, \preceq, \mathrm{el})$.

4 Characterising automatic structures via interpretations

Interpretations constitute an important tool in mathematical logic. They are used to define a copy of a structure inside another one, and thus permit to transfer definability, decidability, and complexity results among theories.

Definition 4.1. Let L be a logic, and let $\mathfrak{A} = (A, R_0, \dots, R_n)$ and $\mathfrak{B}$ be relational structures. A (k-dimensional) *L-interpretation* of $\mathfrak{A}$ in $\mathfrak{B}$ is a sequence

$$\mathcal{I} = \big\langle \delta(\bar{x}),\ \varepsilon(\bar{x},\bar{y}),\ \varphi_{R_0}(\bar{x}_1,\dots,\bar{x}_r),\dots,\ \varphi_{R_n}(\bar{x}_1,\dots,\bar{x}_s)\big\rangle$$

of L-formulae of the vocabulary of $\mathfrak{B}$ (where each tuple $\bar{x}$, $\bar{y}$, $\bar{x}_i$ consists of k variables), such that

$$\mathfrak{A} \cong \mathcal{I}(\mathfrak{B}) := \big(\delta^{\mathfrak{B}},\ \varphi_{R_0}^{\mathfrak{B}},\dots,\ \varphi_{R_n}^{\mathfrak{B}}\big)/\varepsilon^{\mathfrak{B}}.$$

To make this expression well-defined we require that $\varepsilon^{\mathfrak{B}}$ is a congruence relation on the structure $\big(\delta^{\mathfrak{B}},\ \varphi_{R_0}^{\mathfrak{B}},\dots,\ \varphi_{R_n}^{\mathfrak{B}}\big)$. We denote the fact that $\mathcal{I}$ is an L-interpretation of $\mathfrak{A}$ in $\mathfrak{B}$ by $\mathcal{I} : \mathfrak{A} \le_L \mathfrak{B}$. If $\mathfrak{A} \le_L \mathfrak{B}$ and $\mathfrak{B} \le_L \mathfrak{A}$ we say $\mathfrak{A}$ and $\mathfrak{B}$ are *mutually L-interpretable.*

The epimorphism $\big(\delta^{\mathfrak{B}},\ \varphi_{R_0}^{\mathfrak{B}},\dots,\ \varphi_{R_n}^{\mathfrak{B}}\big) \to \mathfrak{A}$ is called *coordinate map* and is also denoted by $\mathcal{I}$. If it is the identity function, i.e., $\mathfrak{A} = \mathcal{I}(\mathfrak{B})$, we say that $\mathfrak{A}$ is *L-definable* in $\mathfrak{B}$.

Examples. (1) Recall that we write $a \mid_p b$ to denote that a is a power of p dividing b. Let $V_p : \mathbb{N} \to \mathbb{N}$ be the function that maps each number to the largest power of p dividing it. It is very easy to see that the structures $(\mathbb{N}, +, \mid_p)$ and $(\mathbb{N}, +, V_p)$ are mutually first-order interpretable. Indeed we can define the statement $x = V_p(y)$ in $(\mathbb{N}, +, \mid_p)$ by the formula $x \mid_p y \wedge \forall z(z \mid_p y \to z \mid_p x)$. In the other direction, $V_p(x) = x \wedge \exists z(x + z = V_p(y))$ is a definition of $x \mid_p y$.

(2) For every $p \in \mathbb{N}$ we write *Tree*(p) for the tree structure *Tree*$(\{0, \dots, p-1\})$. The structures $\mathfrak{N}_p$ and *Tree*(p) are mutually interpretable, for each $p \geq 2$ (see [5, 19]).

If $\mathcal{I} : \mathfrak{A} \leq_{\mathrm{FO}} \mathfrak{B}$ then every first-order formula φ over the vocabulary of $\mathfrak{A}$ can be translated to a formula $\varphi^{\mathcal{I}}$ over the vocabulary of $\mathfrak{B}$ by replacing every relation symbol R by its definition φ_R, by relativising every quantifier to δ, and by replacing equalities by ε.

Lemma 4.2 (Interpretation Lemma). *If $\mathcal{I} : \mathfrak{A} \leq_{\mathrm{FO}} \mathfrak{B}$ then*

$$\mathfrak{A} \models \varphi(\mathcal{I}(\bar{b})) \quad \text{iff} \quad \mathfrak{B} \models \varphi^{\mathcal{I}}(\bar{b}) \qquad \textit{for all } \varphi \in \mathrm{FO} \textit{ and } \bar{b} \subseteq B.$$

This lemma states the most important property of interpretations. For any logic L, a notion of interpretation is considered suitable if a similar statement holds. Note that in the case of MSO, arbitrary k-dimensional MSO-interpretations are to strong since they translate sets to relations of arity k. On the other hand, the Interpretation Lemma does hold for *one-dimensional* MSO*-interpretations.*

Interpretations provide a general and powerful method to obtain classes of finitely presented structures with a set of desired properties. One fixes some structure $\mathfrak{B}$ having these properties and chooses a kind of interpretation that preserves them. Then one considers the class of all structures which can be interpreted in $\mathfrak{B}$. Each structure $\mathfrak{A}$ of this class can be represented by an interpretation $\mathcal{I} : \mathfrak{A} \leq_{\mathrm{FO}} \mathfrak{B}$ which is a finite object, and model checking and query evaluation for such structures can be reduced to the corresponding problem for $\mathfrak{B}$. If $\mathcal{I} : \mathfrak{A} \leq_{\mathrm{FO}} \mathfrak{B}$ then Lemma 4.2 implies that

$$\varphi^{\mathfrak{A}} = \{\, \bar{a} \mid \mathfrak{A} \models \varphi(\bar{a}) \,\} = \{\, \mathcal{I}(\bar{b}) \mid \mathfrak{B} \models \varphi^{\mathcal{I}}(\bar{b}) \,\}.$$

Hence, the desired representation of $\varphi^{\mathfrak{A}}$ can be constructed by extending the interpretation $\mathcal{I}$ to $\langle \mathcal{I}, \varphi^{\mathcal{I}} \rangle : (\mathfrak{A}, \varphi^{\mathfrak{A}}) \leq_{\mathrm{FO}} \mathfrak{B}$.

Automatic structures are closed under first-order interpretations.

Proposition 4.3. *If $\mathfrak{A} \leq_{\mathrm{FO}} \mathfrak{B}$ and $\mathfrak{B}$ is* (ω-)*automatic, then so is $\mathfrak{A}$.*

Proof. Since $\mathfrak{B}$ is automatic there are regular languages L_δ for the universe, L_ε for equality, and L_R for each relation R of $\mathfrak{B}$. By the closure of regular languages under boolean operations and projections it follows that, for each first-order formula φ, the language encoding the relation $\varphi^{\mathfrak{B}}$ is also regular. □

Corollary 4.4. *The classes of automatic, resp. ω-automatic, structures are closed under* (i) *extensions by definable relations,* (ii) *factorisations by definable congruences,* (iii) *substructures with definable universe, and* (iv) *finite powers.*

As stated above the class of automatic structures can be characterised via first-order interpretations.

Theorem 4.5. *For every structure $\mathfrak{A}$, the following are equivalent:*

(i) $\mathfrak{A}$ *is automatic.*

(ii) $\mathfrak{A} \leq_{\mathrm{FO}} \mathfrak{N}_p$ *for some (and hence all)* $p \geq 2$.

(iii) $\mathfrak{A} \leq_{\mathrm{FO}}$ *Tree*(p) *for some (and hence all)* $p \geq 2$.

Proof. The facts that (ii) and (iii) are equivalent and that they imply (i) follow immediately from the mutual interpretability of $\mathfrak{N}_p$ and *Tree*(p), from the fact that these structures are automatic, and from the closure of automatic structures under interpretation.

It remains to show that every automatic structure is interpretable in $\mathfrak{N}_p$ (or *Tree*(p)). Suppose that $\mathfrak{d}$ is an automatic presentation of $\mathfrak{A}$ with alphabet $[p] := \{0, \ldots, p-1\}$ for some $p \geq 2$ (without loss of generality, we could take $p = 2$). For every word $w \in [p]^*$, let $\mathrm{val}(w)$ be the natural number whose p-ary encoding is w, i.e., $\mathrm{val}(w) := \sum_{i<|w|} w_i p^i$. By a classical result, sometimes called the Büchi-Bruyère Theorem, a relation $R \subseteq \mathbb{N}^k$ is first-order definable in $(\mathbb{N}, +, V_p)$ if and only if

$$\{\, (\mathrm{val}^{-1}(x_1), \ldots, \mathrm{val}^{-1}(x_k)) \mid (x_1, \ldots, x_k) \in R \,\}$$

is regular. (See [9] for a proof of this fact and for more information on the relationship between automata and definability in expansions of Presburger arithmetic.) The formulae that define in this sense the regular language and the regular relations in an automatic presentation of $\mathfrak{A}$ provide an interpretation of $\mathfrak{A}$ in $(\mathbb{N}, +, V_p)$. Hence also $\mathfrak{A} \leq_{\mathrm{FO}} \mathfrak{N}_p$. □

For automatic groups we are not free to change the coordinate map. Indeed, the definition of an automatic group requires that the function $\nu : L_\delta \to G$ is the restriction of the canonical homomorphism from S^* to G. Hence the arguments used above give us a characterisation of automatic groups in terms of definability rather than interpretability.

Theorem 4.6. $(G, \cdot)$ *is an automatic group if and only if there exists a finite set* $S \subseteq G$ *of semigroup generators such that* $\Gamma(G, S)$ *is* FO*-definable in* Tree(S).

By definition, if G is an automatic group, then for some set S of semigroup generators, the Cayley graph $\Gamma(G, S)$ is an automatic structure. Contrary to a claim in [28] the converse does not hold. A counterexample, which has been pointed out by Senizergues, is the *Heisenberg group* $\mathfrak{H}$ which is the group of affine transformations of $\mathbb{Z}^3$ generated by the maps

$$\begin{aligned}
&\alpha : (x, y, z) \mapsto (x+1, y, z+y),\\
&\beta : (x, y, z) \mapsto (x, y+1, z),\\
&\gamma : (x, y, z) \mapsto (x, y, z+1).
\end{aligned}$$

Using this matrix representation of $\mathfrak{H}$, it is not difficult to construct a (3-dimensional) interpretation of $\Gamma(\mathfrak{H}, S)$ in $(\mathbb{N}, +)$, which implies that $\Gamma(\mathfrak{H}, S) \in \mathrm{AutStr}$. However, in [15] it is shown that $\mathfrak{H}$ is not automatic.

Proposition 4.7. *There exist groups G with a set of semigroup generators S such that the Cayley graph $\Gamma(G, S)$ is an automatic structure without G being an automatic group.*

We now turn to ω-automatic structures. To provide a similar characterisation we can use an equivalent of the Büchi-Bruyère Theorem for encodings of ω-regular relations. One such result has been obtained by Boigelot, Rassart and Wolper [8]. Using natural translations between ω-words over $[p]$ and real numbers, they prove that a relation over $[p]^\omega$ can be recognised by a Büchi automaton if an only if its translation is first-order definable in the structure $(\mathbb{R}, +, <, \mathbb{Z}, X_p)$ where $X_p \subseteq \mathbb{R}^3$ is a relation that explicitly represents the translation between $[p]^\omega$ and $\mathbb{R}$. $X_p(x, y, z)$ holds iff there exists a representation of x by a word in $[p]^\omega$ such that the digit at the position specified by y is z. A somewhat unsatisfactory aspect of this result is the assumption that the encoding relation X_p must be given as a basic relation of the structure. It would be preferable if more natural expansions of the additive real group $(\mathbb{R}, +)$ could be used instead.

We show here that this is indeed possible if, as in the case of $\mathfrak{N}_p$, we use a restricted variant of the divisibility relation. Recall that the structures $\mathfrak{R}_p$ and *Tree*$^\omega(p)$ (introduced at the end of Section 3) are ω-automatic. As a first step we show that the behaviour of Büchi automata recognising regular relations over $[p]^\omega$ can be simulated by first-order formulae in *Tree*$^\omega(p)$. Secondly we show that *Tree*$^\omega(p)$ and $\mathfrak{R}_p$ are mutually interpretable. As a result we obtain the following model-theoretic characterisation of ω-automatic structures.

Theorem 4.8. *For every structure $\mathfrak{A}$, the following are equivalent:*

(i) $\mathfrak{A}$ *is ω-automatic.*

(ii) $\mathfrak{A} \leq_{\mathrm{FO}} \mathfrak{R}_p$ *for some (and hence all) $p \geq 2$.*

(iii) $\mathfrak{A} \leq_{\mathrm{FO}}$ *Tree*$^\omega(p)$ *for some (and hence all) $p \geq 2$.*

Proof. In order to construct interpretations of *Tree*$^\omega(p)$ in $\mathfrak{R}_p$ and vice versa we define formulae which allow to access the digits of, respectively, some number in $\mathfrak{R}_p$ and some word in *Tree*$^\omega(p)$. In the later case we set

$$\mathrm{dig}_k(x, y) := \exists z(\mathrm{el}(z, y) \wedge \sigma_k z \preceq x)$$

which states that the digit of x at position $|y|$ is k. For $\mathfrak{R}_p$ the situation is more complicated as some real numbers admit two encodings. The following formula describes that there is one encoding of x such that the digit at position y is k. (This corresponds to the predicate X of [8].)

$$\mathrm{dig}_k(x, y) := \exists s \exists t(|x| = s + k \cdot y + t \wedge p \cdot y \,|_p\, s \wedge 0 \leq s \wedge 0 \leq t \leq y)$$

For $\mathfrak{R}_p \leq_{\mathrm{FO}}$ *Tree*$^\omega(p)$ we represent each number as a pair of words. The first one is finite and encodes the integer part, the other one is infinite and contains the fractional part. In the other direction we map finite words $a_1 \cdots a_r \in [p]^*$ to the interval $[2, 3]$ via

$$p^{-r+1} + \sum_{i=1}^{r} a_i p^{-i} + 2 \in [2, 3].$$

Infinite words $a_1 a_2 \cdots \in [p]^\omega$ are mapped to two intervals $[-1, 0]$ and $[0, 1]$ via

$$\pm \sum_i a_i p^{-i} \in [-1, 1].$$

This is necessary because some words, e.g., $0(p-1)^\omega$ and 10^ω, would be mapped to the same number otherwise. Now the desired interpretations can be constructed easily using the formulae dig_k defined above.

It remains to prove that if $R \subseteq ([p]^*)^n$ is ω-regular then it is definable in *Tree*$^\omega(p)$. Let $M = (Q, [p]^n, \Delta, q_0, F)$ be a Büchi-automaton for R. W.l.o.g. assume $Q = [p]^m$ for some m and $q_0 = (0, \dots, 0)$. We prove the claim by constructing a formula $\psi_M(\bar{x}) \in \mathrm{FO}$ stating that there is a successful run of M on $x_1 \otimes \cdots \otimes x_n$. The run is encoded by a tuple $(q_1, \dots, q_m) \in ([p]^\omega)^m$ of ω-words such that the symbols of $q_1, \dots, q_m$ at some position equal $k_1, \dots, k_m$ iff the automaton is in state $(k_1, \dots, k_m)$ when scanning the input symbol at that position. $\psi_M(\bar{x})$ has the form

$$\exists q_1 \cdots \exists q_m [\mathrm{ADM}(\bar{q}, \bar{x}) \wedge \mathrm{START}(\bar{q}, \bar{x}) \wedge \mathrm{RUN}(\bar{q}, \bar{x}) \wedge \mathrm{ACC}(\bar{q}, \bar{x})]$$

where the admissibility condition $\mathrm{ADM}(\bar{x}, \bar{q})$ states that all components of $\bar{x}$ and $\bar{q}$ are infinite, $\mathrm{START}(\bar{x}, \bar{q})$ says that the first state is $\bar{0}$, $\mathrm{ACC}(\bar{x}, \bar{q})$ that some final state appears infinitely often, and $\mathrm{RUN}(\bar{x}, \bar{q})$ ensures that all transitions are correct.

Define the following auxiliary formulae. To access the digits of a tuple of words at some position we define $\mathrm{Sym}_{\bar{a}}(\bar{x}, z) := \bigwedge_i \mathrm{dig}_{a_i}(x_i, z)$, and to characterise the ω-words of $[p]^{\leq \omega}$ we set

$$\mathrm{Inf}(x) := \forall y (x \preceq y \rightarrow x = y).$$

ADM and START are defined as

$$\begin{aligned} \mathrm{ADM}(\bar{q}, \bar{x}) \quad &:= \bigwedge_{i=1}^{m} \mathrm{Inf}(q_i) \wedge \bigwedge_{i=1}^{n} \mathrm{Inf}(x_i), \\ \mathrm{START}(\bar{q}, \bar{x}) &:= \mathrm{Sym}_{\bar{0}}(\bar{q}, \varepsilon), \end{aligned}$$

RUN states that at every position a valid transition is used

$$\mathrm{RUN}(\bar{q}, \bar{x}) := \forall z \bigvee_{(\bar{k}, \bar{a}, \bar{k}') \in \Delta} \big(\mathrm{Sym}_{\bar{k}}(\bar{q}, z) \wedge \mathrm{Sym}_{\bar{a}}(\bar{x}, z) \wedge \mathrm{Sym}_{\bar{k}'}(\bar{q}, \sigma_0 z)\big),$$

and ACC says that there is one final state which appears infinitely often in $\bar{q}$

$$\mathrm{ACC}(\bar{q}, \bar{x}) := \bigvee_{\bar{k} \in F} \forall z \exists z' \big(|z'| > |z| \wedge \mathrm{Sym}_{\bar{k}}(\bar{q}, z')\big). \qquad \square$$

5 Model-checking and query-evaluation

In this section we study decidability and complexity issues for automatic structures. Two fundamental algorithmic problems are:

Model-checking. Given a (presentation of a) structure $\mathfrak{A}$, a formula $\varphi(\bar{x})$, and a tuple of parameters $\bar{a}$ in $\mathfrak{A}$, decide whether $\mathfrak{A} \models \varphi(\bar{a})$.

Query-evaluation. Given a presentation of a structure $\mathfrak{A}$ and some formula $\varphi(\bar{x})$, compute a presentation of $(\mathfrak{A}, \varphi^{\mathfrak{A}})$. That is, given automata for the relations of $\mathfrak{A}$, construct an automaton that recognises $\varphi^{\mathfrak{A}}$.

We first observe that all first-order queries on (ω-)automatic structures are effectively computable since the construction in Proposition 4.3 is effective. In fact, this is the case not only for first-order logic but also for formulae containing the quantifier $\exists^\omega$ meaning "there are infinitely many". To prove this result for the case ω-automatic structures we require some preparations.

Lemma 5.1. *Let $R \subseteq \Sigma^\omega \otimes \Gamma^\omega$ be regular. There is a regular relation $R' \subseteq R$ such that such that*

(i) *if* $(x, y) \in R$ *then there is some* y' *such that* $(x, y') \in R'$*, and*

(ii) *for all* x *there is at most one* y *with* $(x, y) \in R'$.

Proof. W.l.o.g. assume that $R \neq \emptyset$. First we introduce some notation. By $v[i, k)$ we denote the factor $v_i \dots v_{k-1}$ of $v = v_0 v_1 \dots \in \Sigma^\omega$. Similarly, $v[i, \omega)$ is equal to $v_i v_{i+1} \dots$, and $v[i] := v[i, i+1)$.

Let $\mathfrak{A} = (Q, \Sigma \times \Gamma, \Delta, q_0, F)$ be a Büchi-automaton recognising R. Fix an ordering $\sqsubseteq$ of Q such that all final states are less than non-final ones. For a run $\varrho \in Q^\omega$ define

$$\begin{aligned} \mathrm{Inf}(\varrho) &:= \{\, q \in Q \mid \text{there are infinitely many } i \text{ such that } \varrho[i] = q \,\}, \\ \mu(\varrho) &:= \min \mathrm{Inf}(\varrho). \end{aligned}$$

Let $f_i(\varrho)$ be the greatest number such that $\varrho[k, f_i(\varrho))$ contains exactly i times the state $\mu(\varrho)$ where k is the least position such that all states appearing only finitely often are contained in $\varrho[0, k)$.

Denote the lexicographically least run of $\mathfrak{A}$ on $x \otimes y$ by $\varrho(x, y)$. Fix $x \in \Sigma^\omega$ and set $\mu(y) := \mu(\varrho(x, y))$, $f_i(y) := f_i(\varrho(x, y))$. We define an order on Γ^ω by

$$\begin{aligned} y \leq y' \text{ iff } & \mu(y) < \mu(y'), \\ & \text{or } \mu(y) = \mu(y') \text{ and there is some } n \text{ such that} \\ & \quad f_n(y) < f_n(y') \text{ and } f_i(y) = f_i(y') \text{ for all } i < n, \\ & \text{or } \mu(y) = \mu(y'), f_i(y) = f_i(y') \text{ for all } i, \text{ and} \\ & \quad y \text{ is lexicographically less than or equal to } y'. \end{aligned}$$

It should be obvious that the relation $\leq$ is regular. Finally, R' is defined by

$$R' := \{\, (x, y) \in R \mid \text{there is no } y' < y \text{ such that } (x, y') \in R \,\}.$$

Clearly, R' is regular, contained in R and satisfies (ii). Hence, it remains to prove (i). We directly construct the minimal element of $V := \{\, y \mid (x, y) \in R \,\}$ as follows. Let $Y_{-1} \subseteq V$ be the subset of those y with minimal $\mu(y)$. We define a sequence of sets $Y_{-1} \supseteq Y_0 \supseteq Y_1 \supseteq \cdots$ by

$$Y_i := \{\, y \in Y_{i-1} \mid f_i(y) \leq f_i(y') \text{ for all } y' \in Y_{i-1} \,\}.$$

Let y_i be the lexicographically least element of Y_i. Hence, $y_0 \geq y_1 \geq \cdots$. Define $\hat{y}$ by

$$\hat{y}[n] := \lim\nolimits_k y_k[n].$$

We claim that $\hat{y}$ is the minimal element of V.

(a) $\hat{y}$ exists. Set $f_n := \lim_k f_n(y_k) = f_n(y_n)$. The pointwise limit of y_k exists since $y_{n+1}[0, f_n) = y_n[0, f_n)$ for all n. For, otherwise, there is some $k < f_n$ such that

$$y_{n+1}[0, k) = y_n[0, k) \text{ and } y_{n+1}[k] \neq y_n[k].$$

Since y_n, $y_{n+1} \in Y_n$ it follows that $y_n[k] < y_{n+1}[k]$. On the other hand, $y' := y_n[0, f_n) y_{n+1}[f_n, \omega)$ is in V and thus in Y_{n+1}, as $\varrho(x, y_n)[f_n] = \varrho(x, y_{n+1})[f_n]$. Thus, $y_{n+1}[k] \leq y'[k] = y_n[k]$ by choice of y_{n+1}. Contradiction.

(b) $\hat{y} \in V$. An accepting run of $\mathfrak{A}$ on $x \otimes y$ is given by $\hat{\varrho}$ where

$$\hat{\varrho}[f_{n-1}, f_n) = \varrho(x, y_n)[f_{n-1}, f_n)$$

since $\hat{\varrho}[f_n] = \mu(y_0) \in F$ for all n.

(c) $\hat{y}$ is minimal. Suppose $y' \leq y$ for some $y' \in V$. Then $\mu(y') \leq \mu(\hat{y})$ and, by induction on n, one can show that $y' \in Y_n$ since $f_n(y') \leq f_n(\hat{y})$. Thus by construction $\mu(y') = \mu(\hat{y})$ and $f_n(y') = f_n(\hat{y})$. Suppose $y' < y$. Then y' must be lexicographically less than $\hat{y}$ and there exists some k such that $y'[0,k) = \hat{y}[0,k)$ and $y'[k] < \hat{y}[k]$. Choose n such that $f_{n-1} \leq k < f_n$. Then $y_n \leq y'$ by construction. But $y_n[0, f_n) = \hat{y}[0, f_n)$ and hence $y_n[0,k) = y'[0,k)$ which implies that $y'[k] \geq y_n[k] = \hat{y}[k]$. Contradiction. □

Proposition 5.2. *Every (ω-)automatic structure has an injective presentation.*

Proof. For automatic structures this result is due to Khoussainov and Nerode [28].

Let $\nu : D \to A$ be a presentation of an ω-automatic structure $\mathfrak{A}$. By the preceding lemma applied to the kernel of ν there is a function e such that

(i) $\nu x = \nu e x$ for all $x \in \Sigma^\omega$, and

(ii) $\nu x = \nu y$ implies $ex = ey$.

Thus we obtain a regular subset $D' \subseteq D$ containing exactly one representation for each element of $\mathfrak{A}$ by defining $D' := \{\, x \in D \mid ex = x \,\}$. □

We say that a logic L *effectively collapses* to $L_0 \subseteq L$ on a structure $\mathfrak{A}$ if, given a formula $\varphi(\bar{x}) \in L$, one can compute a formula $\varphi_0(\bar{x}) \in L_0$ such that $\varphi_0^{\mathfrak{A}} = \varphi^{\mathfrak{A}}$.

Proposition 5.3. (1) FO($\exists^\omega$) *effectively collapses to* FO *on Tree*(p).

(2) FO($\exists^\omega$) *effectively collapses to* FO *on Tree*$^\omega$(p).

Proof. (1) In case of automatic structures the quantifier $\exists^\omega$ can be handled using a pumping argument. Consider for simplicity the formula $\exists^\omega x \psi(x,y)$. By induction the formula ψ is equivalent to some first-order formula and, hence, there is some automaton recognising the relation defined by ψ. There are infinitely many x satisfying ψ iff for any m there are infinitely many elements x whose encoding is at least m symbols longer than that of y. If we take m to be the number of states of the automaton for ψ then, by the Pumping Lemma, the last condition is equivalent to the existence of at least one such x. Thus $\exists^\omega x \psi(x,y) \equiv \exists x(\psi(x,y) \wedge$ "x is long enough") for which we can obviously construct an automaton.

(2) For ω-automatic structures the proof is more involved.

Let M be a deterministic Muller automaton with s states recognising the language $L(M) \subseteq \Gamma^\omega \otimes \Sigma^\omega$. For $w \in \Gamma^\omega$ let $V(w) := \{\, v \in \Sigma^\omega \mid w \otimes v \in L(M) \,\}$.

Let v, $w \in \Sigma^\omega$ and define $v \approx^* w$ iff $v[n,\omega) = w[n,\omega)$ for some n. Let $[v]_* := \{\, v' \in V(w) \mid v' \approx^* v \,\}$ be the $\approx^*$-class of v in $V(w)$.

Claim. $V(w)$ is infinite if and only if there is some $v \in \Sigma^\omega$ such that $[v]_* \in V(w)/{\approx^*}$ is infinite.

Proof. ($\Leftarrow$) is trivial and ($\Rightarrow$) is proved by showing that $V/{\approx^*}$ contains at most s finite $\approx^*$-classes.

Assume there are words $v_0, \dots, v_s \in V(w)$ belonging to different finite $\approx^*$-classes. Denote the run (sequence of states) of M on $w \otimes v_i$ by ϱ_i. Define $I_{ij} := \{\, k < \omega \mid \varrho_i[k] = \varrho_j[k] \,\}$. Since there are only s states, for each $k < \omega$ there have to be indices i, j such that $k \in I_{ij}$, i.e., $\bigcup_{i,j} I_{ij} = \omega$. Thus, at least one I_{ij} is infinite. For each $[v_i]_*$ there is a position n_i such that $v[n_i, \omega) = v'[n_i, \omega)$ for all $v, v' \in [v_i]_*$. Let m be the maximum of $n_0, \dots, n_s$. Fix i, j such that I_{ij} is infinite. Since $v_i \not\approx^* v_j$ there is a position $m' > m$ such that $v_i[m, m') \neq v_j[m, m')$. Choose some $m'' \in I_{ij}$ with $m'' \geq m'$. Let $u := v_i[0, m)v_j[m, m'')v_i[m'', \omega)$. Then, $w \otimes v_i \in L(M)$ iff $w \otimes u \in L(M)$ which implies that $u \in [v_i]_*$. But $u[m, \omega) \neq v_i[m, \omega)$ in contradiction to the choice of m. □

To finish the proof let $\varphi(\bar{x}) := \exists^\omega y \psi(\bar{x}, y)$ and $\mathfrak{A}$ be ω-automatic. One can express that $[v]_*$ is finite by

$$\text{finite}(\bar{x}, v) := \exists n \forall v' [\psi(\bar{x}, v') \wedge v \approx^* v' \rightarrow \text{equal}(v, v', n)],$$

where

$$\text{equal}(v, v', n) := n = 1^i 0^\omega \ \wedge \ v[i, \omega) = v'[i, \omega).$$

Clearly, $\approx^*$ and equal can be recognised by ω-automata. By the claim above,

$$\varphi(\bar{x}) \equiv \exists v(\psi(\bar{x}, v) \wedge \neg\text{finite}(\bar{x}, v)).$$

Hence, we can construct an automaton recognising $\varphi^{\mathfrak{A}}$. □

Corollary 5.4. *The* $\mathrm{FO}(\exists^\omega)$*-theory of any* (ω)*-automatic structure is decidable.*

Proof. Note that $\mathrm{FO}(\exists^\omega)$ is closed under *injective* interpretations, that is, if $\mathcal{I} : \mathfrak{A} \leq_{\mathrm{FO}} \mathit{Tree}(p)$ is an injective interpretation, then $\mathfrak{A} \models \varphi(\mathcal{I}(\bar{w}))$ iff $\mathit{Tree}(p) \models \varphi^{\mathcal{I}}(\bar{w})$ for every $\varphi \in \mathrm{FO}(\exists^\omega)$. □

As an immediate consequence we conclude that full arithmetic $(\mathbb{N}, +, \cdot)$ is neither automatic, nor ω-automatic. For most of the common extensions of first-order logic used in finite model theory, such as transitive closure logics, fixed point logics, monadic second-order logic, or first-order logic with counting, the model-checking problem on automatic structures becomes undecidable.

Complexity. The complexity of model-checking can be measured in three different ways. First, one can fix the formula and ask how the complexity depends on the input structure. This measure is called *structure complexity.* The *expression complexity* on the other hand is defined relative to a fixed structure in terms of the formula. Finally, one can look at the *combined complexity* where both parts may vary.

Of course, the complexity of these problems may very much depend on how automatic structures are presented. Since the decision methods for $\mathfrak{N}_p$ and $\mathit{Tree}(p)$ are automaton based, a presentation $\mathfrak{d}$ consisting of a list of automata, is more suitable for practical purposes than using an interpretation. Here, we focus on presentations by *deterministic* automata because these admit boolean operations to be performed in polynomial time, whereas for nondeterministic automata, complementation may cause an exponential blow-up.

In the following we always assume that the vocabulary of the given automatic structures and the alphabet of the automata we deal with are fixed. Furthermore the vocabulary is assumed to be relational when not stated otherwise.

	Structure-Complexity	Expression-Complexity
Model-Checking		
Σ_0	Logspace-complete	Alogtime-complete
$\Sigma_0 + \text{fun}$	Nlogspace	Ptime-complete
Σ_1	Nptime-complete	Pspace-complete
Query-Evaluation		
Σ_0	Logspace	Pspace
Σ_1	Pspace	Expspace

While we have seen above that query-evaluation and model-checking for first-order formulae are effective on AutStr, the complexity of these problems is non-elementary, i.e., it exceeds any fixed number of iterations of the exponential function. This follows immediately from the fact the the complexity of $\mathrm{Th}(\mathfrak{N}_p)$ is non-elementary (see [19]).

Proposition 5.5. *There exist automatic structures such that the expression complexity of the model-checking problem is non-elementary.*

It turns out that model-checking and query-evaluation for quantifier-free and existential formulae are still – to some extent – tractable. The table above summarises the complexity results obtained in [5, 7]. As usual, Σ_0 and Σ_1 denote, respectively the class of quantifier-free and the class of existential first-order formulae.

For most questions we can restrict attention to relational vocabularies and replace functions by their graphs at the expense of introducing additional quantifiers. When studying quantifier-free formulae we will not want do to this and hence need to consider the case of quantifier-free formulae with function symbols separately. This class is denoted $\Sigma_0 + \text{fun}$.

Undecidability. In the remainder of this section we present some undecidability results.

Lemma 5.6. *Every configuration graph of a Turing maching is automatic.*

Proof. We encode a configuration with state q, tape contents w, and head position p by the word w_0qw_1 where $w = w_0w_1$ and $|w_0| = p$. At every transition $w_0qw_1 \vdash_m w_0'q'w_1'$, only the symbols around the state are changed. This can be checked by an automaton. □

An immediate consequence is:

Proposition 5.7. Reachability *is undecidable for automatic structures.*

The proofs below are based on the following normal form for Turing machines:

Lemma 5.8. *For any deterministic 1-tape Turing machine M one can effectively construct a deterministic 2-tape Turing machine M' such that the configuration graph of M' consists of a disjoint union of*

(a) *a countably infinite number of infinite acyclic paths with a first but without last element;*

(b) *for each word $x \in L(M)$, one path starting at an initial configuration and ending in a loop.*

Proof. We slightly modify the construction of a reversible Turing machine. While simulating M on its first tape the machine M' appends to the second tape the transitions performed at each step. That way it is ensured that each configuration of M' has a unique predecessor. Further, we define M' such that, if M terminates without accepting, then M' enters an infinite loop while moving its head to the right. Thus, if $x \notin L(M)$ then the run of M' on input x consists of an infinite path of type (a).

The construction as presented so far does not suffice since there exist configurations that cannot be reached from an initial configuration. We have to ensure that every path containing such a configuration is of type (a). Clearly, every such path must have a first element since the contents of the second tape never decreases. To ensure that the path does not end we modify M' such that, if M accepts, then M' restarts the simulation of M with the initial configuration but without erasing the contents of the second tape. Instead, at every step M' checks that the symbol it normally would write to this tape is already present. That way, if the path starts at an unreachable configuration, a discrepancy is detected and M' can enter an infinite loop as above.

Finally, note that with this modification the run of M' on inputs x with $x \in L(M)$ is of type (b). □

Theorem 5.9. *It is undecidable whether two automatic structures are isomorphic.*

Proof. Let M be a deterministic 1-tape Turing machine. We construct a Turing machine M' as in the preceding lemma. The configuration graph G of M' is automatic. Let H be the graph consisting of $\aleph_0$ copies of (ω, suc). Then, $G \cong H$ iff $L(M) = \emptyset$. The latter question is undecidable. □

Theorem 5.10. *It is undecidable whether an automatic graph is connected.*

Proof. We modify the proof above. Again construct the Turing machine M' with configuration graph G. This time, we modify M' such that it enters a distinguished configuration c_0 if an error is detected in the second phase. Let T be the set of configurations with two predecessors. We add edges from c to c_0 for every $c \in T$. Since T is FO-definable it follows that the resulting graph G' is still automatic. Furthermore, G' is connected iff every run of M reaches an accepting configuration, i.e., iff $L(M) = \Sigma^*$. □

6 Composition of structures

The composition method developed by Feferman and Vaught, and by Shelah considers compositions (products and sums) of structures according to some index structure and allows one to compute – depending on the type of composition – the first-order or monadic second-order theory of the whole structure from the respective theories of its components and the monadic theory of the index structure.

The characterisation given in the previous section can be used to prove closure of automatic structures under such compositions of finitely many structures (see [18, 34, 23, 36]). A generalised product – as it is defined below – is a generalisation of a direct product, a disjoint union, and an ordered sum. We will prove that given a finite sequence $(\mathfrak{A}_i)_i$ of structures which belong to some class $\mathcal{K}$ containing a complete structure, all their generalised products are members of $\mathcal{K}$ as well.

The definition of such a product is a bit technical. Its relations are defined in terms of the types of the components of its elements. The *atomic n-type* $\mathrm{atp}_{\mathfrak{A}}(\bar{a})$ of a tuple $(a_0, \dots, a_{n-1})$ in a structure $\mathfrak{A}$ is the conjunction of all atomic and negated atomic formulae $\varphi(\bar{x})$ such that $\varphi(\bar{a})$ holds in $\mathfrak{A}$.

Let us first look at how a direct product and an ordered sum can be defined using types.

Example. (1) Let $\mathfrak{A} := \mathfrak{A}_0 \times \mathfrak{A}_1$ where $\mathfrak{A}_i = (A_i, R_i)$, for $i \in \{0,1\}$, and R is a binary relation. The universe of $\mathfrak{A}$ is $A_0 \times A_1$. Some pair $(\bar{a}, \bar{b})$ belongs to R iff $(a_0, b_0) \in R_0$ and $(a_1, b_1) \in R_1$. This is equivalent to the condition that the atomic types of $a_0 b_0$ and of $a_1 b_1$ both include the formula Rx_0x_1.

(2) Let $\mathfrak{A} := \mathfrak{A}_0 + \mathfrak{A}_1$ where $\mathfrak{A}_i = (A_i, <_i)$, for $i \in \{0,1\}$, and $<_0$, $<_1$ are partial orders. The universe of $\mathfrak{A}$ is $A_0 \cup A_1 \cong A_0 \times \{\Diamond\} \cup \{\Diamond\} \times A_1$, and we have

$$\begin{aligned}
\bar{a} < \bar{b} \text{ iff } & \bar{a} = (a_0, \Diamond),\ \bar{b} = (b_0, \Diamond) \text{ and } a_0 <_0 b_0, \\
\text{or } & \bar{a} = (\Diamond, a_1),\ \bar{b} = (\Diamond, b_1) \text{ and } a_1 <_1 b_1, \\
\text{or } & \bar{a} = (a_0, \Diamond),\ \bar{b} = (\Diamond, b_1).
\end{aligned}$$

Again, the condition $a_i <_i b_i$ can be expressed using types.

Definition 6.1. Let $\tau = \{R_0, \dots, R_s\}$ be a finite relational vocabulary, r_j the arity of R_j, and $\hat{r} := \max\{r_0, \dots, r_s\}$. Let $(\mathfrak{A}_i)_{i \in I}$ be a sequence of τ-structures, and $\mathfrak{I}$ be an arbitrary relational σ-structure with universe I.

Fix for each $k \leq \hat{r}$ an enumeration $\{t^k_0, \dots, t^k_{n(k)}\}$ of the atomic k-types and set

$$\sigma_k := \sigma \cup \{D_0, \dots, D_{k-1}\} \cup \{\, T^m_l \mid m \leq k, l \leq n(m) \,\}.$$

The σ_k-expansion $\mathfrak{I}(\bar{b})$ of $\mathfrak{I}$ belonging to a sequence $\bar{b} \in \bigl(\prod_{i \in I}(A_i \cup \{\Diamond\})\bigr)^k$ is given by

$$\begin{aligned}
D_l^{\mathfrak{I}(\bar{b})} &:= \bigl\{\, i \in I \bigm| (b_l)_i \neq \Diamond \,\bigr\}, \\
(T^m_l)^{\mathfrak{I}(\bar{b})} &:= \Bigl\{\, i \in I \Bigm| \mathrm{atp}_{\mathfrak{A}}((b_{j_0})_i \dots (b_{j_{m-1}})_i) = t^m_l \text{ and} \\
&\qquad\qquad \{\, j \mid (b_j)_i \neq \Diamond \,\} = \{j_0, \dots, j_{m-1}\} \,\Bigr\}.
\end{aligned}$$

For $D \subseteq \mathbb{B}^I$ and $\beta_j \in \mathrm{FO}[\sigma_{r_j}]$, $\mathcal{C} := (\mathfrak{I}, D, \beta_0, \dots, \beta_s)$ defines the *generalised product* $\mathcal{C}(\mathfrak{A}_i)_{i \in I} := (A, R_0, \dots, R_s)$ of $(\mathfrak{A}_i)_{i \in I}$ where

$$\begin{aligned}
A &:= \bigcup_{\bar{d} \in D} \prod_{i \in I} \chi_{d_i}(\{\Diamond\}, A_i), \\
R_i &:= \{\, \bar{b} \in A^{r_i} \mid \mathfrak{I}(\bar{b}) \models \beta_i \,\},
\end{aligned}$$

and $\chi_b(a_0, a_1) := a_b$.

Example. (continued)

(1) For the direct product of $\mathfrak{A}_0 \times \mathfrak{A}_1$ we would set $\mathfrak{I} := (I)$ with $I = \{0,1\}$, $D := \{(1,1)\}$, and

$$\beta := \bigvee_{l \in L} T^2_l 0 \wedge \bigvee_{l \in L} T^2_l 1,$$

where L is the set of atomic types containing the formula Rx_0x_1.

(2) In this case we would set $\mathfrak{I} := (I)$ with $I = \{0,1\}$, $D := \{(1,0),(0,1)\}$, and

$$\beta := \Bigl(D_0 0 \wedge D_1 0 \wedge \bigvee_{l \in L} T^2_l 0\Bigr) \vee \Bigl(D_0 1 \wedge D_1 1 \wedge \bigvee_{l \in L} T^2_l 1\Bigr) \vee (D_0 0 \wedge D_1 1),$$

where L is the set of atomic types containing the formula $x_0 < x_1$.

Theorem 6.2. *Let τ be a finite relational vocabulary, and $\mathcal{K}$ a class of τ-structures containing all finite τ-structures and a structure $\mathfrak{C}$ which is complete for $\mathcal{K}$ with regard to many-dimensional* FO*-interpretations.*

Let $\mathfrak{I}$ be a finite relational σ-structure, let $(\mathfrak{A}_i)_{i\in I}$ be a sequence of structures in $\mathcal{K}$, and $\mathcal{C} = (\mathfrak{I}, D, \bar{\beta})$ a generalised product. Then $\mathcal{C}(\mathfrak{A}_i)_{i\in I} \in \mathcal{K}$, and an interpretation $\mathcal{C}(\mathfrak{A}_i)_{i\in I} \leq_{\mathrm{FO}} \mathfrak{C}$ can be constructed effectively from the interpretations $\mathfrak{A}_i \leq_{\mathrm{FO}} \mathfrak{C}$ and $\mathfrak{I} \leq_{\mathrm{FO}} \mathfrak{C}$.

Proof. Let $\tau = \{R_0, \dots, R_s\}$. W.l.o.g. assume that $I = \{0, \dots, |I|-1\}$ and that $\mathfrak{C}$ contains constants 0 and 1. We have to construct an interpretation of $\mathfrak{A} := \mathcal{C}(\mathfrak{A}_i)_{i\in I}$ in $\mathfrak{C}$. Let r_j be the arity of R_j. Consider n_i-dimensional interpretations

$$\mathcal{I}^i := \big\langle h^i, \delta^i(\bar{x}^i), \varepsilon^i(\bar{x}^i, \bar{y}^i), \varphi_0^i(\bar{x}_0^i, \dots, \bar{x}_{r_0-1}^i), \dots, \varphi_s^i(\bar{x}_0^i, \dots, \bar{x}_{r_s-1}^i)\big\rangle$$

of $\mathfrak{A}_i$ in $\mathfrak{C}$. We represent an element a of $\mathfrak{A}$ by a tuple of $(|I| + n_0 + \dots + n_{|I|-1})$ elements

$$\bar{x} := \big(\bar{d}, \bar{x}^0, \dots, \bar{x}^{|I|-1}\big)$$

where $\bar{d} \in D$ determines which components are empty and $\bar{x}^i$ encodes the i^{th} component of a. The desired interpretation is constructed as follows.

$$\mathcal{I} := \big\langle h, \delta(\bar{x}), \varepsilon(\bar{x}, \bar{y}), \varphi_0(\bar{x}_0, \dots, \bar{x}_{r_0-1}), \dots, \varphi_s(\bar{x}_0, \dots, \bar{x}_{r_s-1})\big\rangle$$

where

$$h(\bar{d}, \bar{x}^0, \dots, \bar{x}^{|I|-1}) := \big(\chi_{d_0}(\Diamond, h^0(\bar{x}^0)), \dots, \chi_{d_{|I|-1}}(\Diamond, h^{|I|-1}(\bar{x}^{|I|-1}))\big),$$
$$\delta(\bar{d}, \bar{x}^0, \dots, \bar{x}^{|I|-1}) := \bigvee_{\bar{c}\in D} \Big(\bar{d} = \bar{c} \wedge \bigwedge_{i\,:\, c_i = 1} \delta^i(\bar{x}^i)\Big),$$

and

$$\varepsilon(\bar{d}, \bar{x}^0, \dots, \bar{x}^{|I|-1}, \bar{e}, \bar{y}^0, \dots, \bar{y}^{|I|-1}) := \bar{d} = \bar{e} \wedge \bigwedge_{i<|I|} \big(d_i = 1 \rightarrow \varepsilon^i(\bar{x}^i, \bar{y}^i)\big).$$

In order to define φ_j we consider an interpretation $\mathcal{I}^I$ of $\mathfrak{I}$ in $\mathfrak{C}$. Since $\mathfrak{I}$ is finite such an interpretation exists. Let $\alpha_j := \beta_j^{\mathcal{I}^I}$ be the formula defining R_j. Note that β_j contains additional relations D_l and T_l^m which are not in σ. Thus α_j is a sentence over the vocabulary τ extended by the symbols D_l and T_l^m for appropriate l and m. We have to be replace them in order to obtain a definition of φ_j. Let $\bar{x}_0, \dots, \bar{x}_{r_j-1}$ be the parameters of φ_j where

$$\bar{x}_k = (\bar{d}_k, \bar{x}_k^0, \dots, \bar{x}_k^{|I|-1})$$

for $k < r_j$. D_l and T_l^m can be defined by

$$D_l i := (d_l)_i = 1 \qquad \text{and} \qquad T_l^m i := (t_l^m)^{\mathcal{I}^i}(\bar{x}_0^i, \dots, \bar{x}_{r_j-1}^i).$$

Note that those definitions are only valid because i ranges over a finite set. φ_j can now be defined as α_j with D_l and T_l^m replaced by the above definitions.

Obviously, all steps in the construction above are effective. □

Corollary 6.3. *Both,* AutStr *and* ω-AutStr *are effectively closed under finitary generalised products.*

As promised we immediately obtain closure under several types of compositions.

Corollary 6.4. *Let $\mathfrak{A}_0, \dots, \mathfrak{A}_{n-1} \in \mathrm{AutStr}$. Then there exists automatic presentations of*

(i) *the direct product $\prod_{i<n} \mathfrak{A}_i$,*

(ii) *the disjoint union $\biguplus_{i<n} \mathfrak{A}_i$, and*

(iii) *the ω-fold disjoint union $\omega \cdot \mathfrak{A}_0$ of $\mathfrak{A}_0$.*

Corollary 6.5. *Let $\mathfrak{A}_0, \dots, \mathfrak{A}_{n-1} \in \mathrm{AutStr}$ be ordered structures. There exists automatic presentations of*

(i) *the ordered sum $\sum_{i<n} \mathfrak{A}_i$ and*

(ii) *the ω-fold ordered sum $\sum_{i<\omega} \mathfrak{A}_0$ of $\mathfrak{A}_0$.*

References

[1] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*, Addison-Wesley, 1995.

[2] C. Ash and J. Knight, *Computable structures and the hyperarithmetical hierarchy*, Elsevier, 2000.

[3] K. Barthelmann, *On equational simple graphs*, Tech. Rep. 9, Universität Mainz, Institut für Informatik, 1997.

[4] D. Berwanger and A. Blumensath, *The monadic theory of tree-like structures*, in Automata, Logic, and Infinite Games, E. Grädel, W. Thomas, and T. Wilke, eds., Springer Verlag, 2002. To appear.

[5] A. Blumensath, *Automatic structures.* Diplomarbeit, RWTH Aachen, 1999.

[6] A. Blumensath, *Prefix-recognisable graphs and monadic second-order logic*, Tech. Rep. AIB-06-2001, RWTH Aachen, 2001.

[7] A. Blumensath and E. Grädel, *Automatic structures*, in Proc. 15th IEEE Symp. on Logic in Computer Science, 2000, pp. 51–62.

[8] B. Boigelot, S. Rassart, and P. Wolper, *On the expressiveness of real and integer arithmetic automata*, in Proceedings of 25th International Colloquium on Automata, Languages and Programming ICALP 98, vol. 1443 of LNCS, 1998, pp. 152–163.

[9] V. Bruyère, G. Hansel, C. Michaux, and R. Villemaire, *Logic and p-recognizable sets of integers*, Bull. Belg. Math. Soc., 1 (1994), pp. 191–238.

[10] D. Caucal, *On infinite transition graphs having a decidable monadic theory*, in Automata, Languages and Programming, 23rd International Colloquium, ICALP96, Lecture Notes in Computer Science Nr. 1099, Springer, 1996, pp. 194–205.

[11] B. Courcelle, *The monadic second-order logic of graphs II: Infinite graphs of bounded width*, Math. System Theory, 21 (1989), pp. 187–221.

[12] B. Copurcelle, *The monadic second-order logic of graphs IX: Machines and their behaviours*, Theoretical Computer Science, 151 (1995), pp. 125–162.

[13] B. Courcelle and I. Walukiewicz, *Monadic second-order logic, graph coverings and unfoldings of transition systems*, Annals of Pure and Applied Logic, 92 (1998), pp. 35–62.

[14] H.-D. Ebbinghaus and J. Flum, *Finite Model Theory*, Springer, 1995.

[15] D. Epstein, J. Cannon, D. Holt, S. Levy, M. Paterson, and W. Thurston, *Word Processing in Groups*, Jones and Bartlett Publishers, Boston, 1992.

[16] Y. L. Ershov, S. S. Goncharov, A. Nerode, and J. B. Remmel, *Handbook of Recursive Mathematics*, North-Holland, 1998.

[17] B. Farb, *Automatic groups: A guided tour*, L' Enseignement Mathématique, 38 (1992), pp. 291–313.

[18] S. Feferman and R. L. Vaught, *The first order properties of products of algebraic systems*, Fundamenta Mathematicae, XLVII (1959), pp. 57–103.

[19] E. Grädel, *Simple interpretations among complicated theories*, Information Processing Letters, 35 (1990), pp. 235–238.

[20] E. Grädel and Y. Gurevich, *Metafinite model theory*, Information and Computation, 140 (1998), pp. 26–81.

[21] E. Grädel and A. Malmström, *0-1 laws for recursive structures*, Archive of Mathematical Logic, 38 (1999), pp. 205–215.

[22] E. Grädel and K. Meer, *Descriptive complexity theory over the real numbers*, in Mathematics of Numerical Analysis: Real Number Algorithms, J. Renegar, M. Shub, and S. Smale, eds., vol. 32 of Lectures in Applied Mathematics, AMS, 1996, pp. 381–403.

[23] Y. Gurevich, *Monadic second-order theories*, in Model-Theoretic Logics, J. Barwise and S. Feferman, eds., Springer, 1985, pp. 479–506.

[24] D. Harel, *Towards a theory of recursive structures*, in Proceedings of 23rd International Symposium on Mathematical Foundations of Computer Science MFCS 98, vol. 1450 of Lecture Notes in Computer Science, Springer, 1998, pp. 36–53.

[25] T. Hirst and D. Harel, *More about recursive structures: Descriptive complexity and zero-one laws*, in Proc. 11th IEEE Symp. on Logic in Computer Science, 1996, pp. 334–348.

[26] N. Immerman, *Descriptive complexity*, Graduate Texts in Computer Science, Springer, 1998.

[27] P. Kanellakis, G. Kuper, and P. Revesz, *Constraint query languages*, Journal of Computer and Systems Sciences, 51 (1995), pp. 26–52. (An extended abstract appeared in the Proceedings of PODS'90).

[28] B. Khoussainov and A. Nerode, *Automatic presentations of structures*, in Logic and Computational Complexity, vol. 960 of LNCS, Springer, 1995, pp. 367–392.

[29] G. Kuper, L. Libkin, and J. Paredaens, eds., *Constraint Databases*, Springer, 2000.

[30] C. Löding, *Model-checking infinite systems generated by ground tree rewriting*, in Proceedings of Foundations of Software Science and Computation Structures FoSSaCS 2002, Lecture Notes in Computer Science Nr. 2303, Springer, 2002.

[31] C. Morvan, *On rational graphs*, in Proceedings of FOSSACS 2000, Lecture Notes in Computer Science Nr. 1784, Springer, 2000, pp. 252–266.

[32] D. Muller and P. Schupp, *Groups, the theory of ends, and context-free languages*, Journal of Computer and System Sciences, 26 (1983), pp. 295–310.

[33] D. Muller and P. Schupp, *The theory of ends, pushdown automata, and second-order logic*, Theoretical Computer Science, 37 (1985), pp. 51–75.

[34] S. Shelah, *The monadic theory of order*, Annals of Mathematics, 102 (1975), pp. 379–419.

[35] A. Stolboushkin, *Towards recursive model theory*, in Logic Colloquium 95, J. Makowsky and E. Ravve, eds., no. 11 in Lecture Notes in Logic, Springer, 1998, pp. 325–338.

[36] W. Thomas, *Ehrenfeucht games, the composition method, and the monadic theory of ordinal words*, in Structures in Logic and Computer Science. A Selection of Essays in Honor of Andrzej Ehrenfeucht, G. Rozenberg, A. Salomaa, and J. Mycielski, eds., no. 1261 in LNCS, Springer, 1997, pp. 118–143.

[37] I. Walukiewicz, *Monadic second-order logic on tree-like structures*, Theoretical Computer Science, 275 (2001), pp. 311–346.

The expressive power of Binary Linear Programming[*]

Marco Cadoli

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
cadoli@dis.uniroma1.it

Abstract. Very efficient solvers for Integer Programming exist, when the constraints and the objective function are linear. In this paper we tackle a fundamental question: what is the expressive power of Integer Linear Programming? We are able to prove that ILP, more precisely *Binary* LP, expresses the complexity class NP. As a consequence, in principle all specifications of combinatorial problems in NP formulated in constraint languages can be translated as BLP models.

1 Introduction

Both mathematical (in particular integer) programming (IP) and constraint programming (CP) languages are widely used for the specification of combinatorial problems. Apart from that, there are various differences between the two approaches. As highlighted in [8], CP languages are typically easier to use because they have a richer syntax for representing constraints. On the other hand, IP solvers like CPLEX [5] have advanced techniques such as cut generation and presolve reductions that improve their performance. In general, IP and CP technologies are complementary, and it is difficult to say which one is better. There are considerable research efforts in taking "the best of both worlds" (cf., e.g., [10]), and in particular in finding suitable translations in IP of specifications formulated in CP [9].

In [9] the main effort is in trying to translate CP specifications into *linear* constraints, because this allows to use a solver for Integer Linear Programming (ILP) for solving a CP problem without having to give a linear formulation, which is sometimes far from natural. In fact ILP (and in general *mixed* IP) solvers maintain a *relaxed* solution of the *linear relaxation* of the problem, and generate cutting planes to strengthen the relaxation. Moreover, non-linear IP solvers are relatively rare: for example, ILOG's CPLEX solver [5] handles only linear integer constraints (plus non-linear, non-integer constraints).

[*] Extended version of a paper appeared in the Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming (CP '01), Lecture Notes in Computer Science, volume 2239, pages 570-574, Springer, 2001.

For such an approach to be effective, we have to be sure that imposing the syntactic constraint of linearity does not rule out the possibility of solving problems of interest.

In this paper we tackle a fundamental question: what is the *expressive power* of Integer Linear Programming, i.e., which problems can be expressed with linear models? We are able to prove that ILP expresses the complexity class NP, i.e., that for each problem ψ in NP there is an ILP model π such that, for all instances, ψ and π are equivalent. As a consequence, in principle all specifications of combinatorial problems in NP formulated in constraint programming can be translated as ILP models. Actually, we need only integer variables taking values in $\{0, 1\}$, hence the result holds for *Binary* Linear Programming (BLP).

Expressive power must not be confused with computational complexity. The latter refers to the difficulty to solve an *instance* of a problem, while the former refers to the capability of a language to *describe problems*, i.e., functions. In fact, the expressive power of a language is not necessarily the same as its complexity: for examples of languages with this property, cf., e.g., [1]. Separating data from problem description, called *model* in the terminology of operations research, is a fundamental idea in database theory, and is enforced also by mathematical programming modeling languages such as AMPL [3] and OPL [10]. Using database terminology, it is obvious that the *data complexity* of BLP, i.e., the complexity wrt the size of data and disregarding the model, is NP-hard: as an example, an instance of the SAT problem can be translated (cf. e.g., [6]) into a set of linear constraints over binary variables. On the other hand, to the best of our knowledge, the question of whether all problems in NP can be stated as BLP models has not been addressed so far.

The paper is organized as follows. In Section 2 we recall the basic notions on logical specification of problems and integer programming. In Section 3 we provide the main result, i.e., that BLP expresses the complexity class NP. Conclusions, related and future work are discussed in Section 4.

2 Preliminaries

2.1 Logical specification of problems

We refer to the data of a problem, i.e., the representation of an instance, with the term *database*. All constants appearing in a database are *uninterpreted*, i.e., they don't have a specific meaning.

In the following, σ denotes a fixed set of relational symbols not including equality "=" and $S_1, \ldots, S_h$ denote variables ranging over relational symbols distinct from those in $\sigma \cup \{=\}$. By Fagin's theorem [2] any collection $\mathbf{D}$ of finite databases over σ recognizable in NP time is defined by an existential second-order (ESO) formula of the kind:

$$(\exists S_1, \ldots, S_h)\ \phi, \tag{1}$$

where $S_1, \ldots, S_h$ are relational variables of various arities and ϕ is a function-free first-order formula containing occurrences of relational symbols from $\sigma \cup$

$\{S_1, \ldots, S_h\} \cup \{=\}$. The symbol "=" is always interpreted in the obvious way, i.e., as "identity".

A database D is in $\mathbf{D}$ if and only if there is a list of relations $\Sigma_1, \ldots, \Sigma_h$ (matching the list of relational variables $S_1, \ldots, S_h$) which, along with D, satisfies formula (1), i.e., such that $(D, \Sigma_1, \ldots, \Sigma_h) \models \phi$. The tuples of $\Sigma_1, \ldots, \Sigma_h$ must take elements from the *Herbrand universe* of D, i.e., the set of constant symbols occurring in it.

Example 1 ([7]). In the propositional satisfiability problem the input is a set V of propositional variables, a set C of names of propositional clauses, and two sets N, P of pairs $\langle c, v \rangle$ $(c \in C, v \in V)$ encoding the clauses, i.e., $N(c,v)$ $[P(c,v)]$ holds iff v occurs negatively [positively] in clause c. The question is whether there is an assignment S of truth values to variables in V such that each clause in C is satisfied. The question can be specified as an ESO formula as follows:

$$\begin{array}{ll}(\exists S)\ (\forall x)(\exists y) & [S(x) \rightarrow V(x)] \wedge \\ & [\neg V(x) \rightarrow (P(x,y) \wedge S(y)) \vee (N(x,y) \wedge \neg S(y))]\end{array} \qquad (2)$$

2.2 Integer linear programming

An ILP problem is a set of integer variables $x_1, \ldots, x_n$ and a set of *linear* constraints over such variables. Linearity of constraints means that they must have one of the following forms:

$$c_1 \cdot x_1 + \cdots + c_n \cdot x_n = b$$
$$c_1 \cdot x_1 + \cdots + c_n \cdot x_n \geq b$$

where $c_1, \ldots, c_n, b$ are variable-free expressions. Linearity enforces the search space to be a convex polyhedron, thus allowing specific algorithms to be implemented.

Typically, an ILP problem has also a linear *objective function* to be maximized or minimized. In what follows we will ignore objective functions, because we are dealing with decision problems. Moreover, since we really need only variables taking values in $\{0, 1\}$, we will deal just with Binary Linear Programming.

In mathematical programming, data and the so-called *model* (which is a description of the problem and is not to be confused with the notion of model in logic) are separated. Modeling languages such as AMPL allow very convenient ways to express models and data.

Example 2. In the "one-in-three" NP-complete decision problem (cf. [4, Problem LO4]) the input is a set U and a collection T of triplets of elements of U, and the question is whether there is a selection A of elements of U such that each triplet has exactly one selected item. A BLP model for the problem is as follows:

variables: $A \in \{0,1\}^{|U|}$ (one for each element of the set U).
constraints: for each $(x, y, z) \in T$, $A(x) + A(y) + A(z) = 1$

In the popular mathematical programming modeling language AMPL [3], the model is as follows[1]:

```
set U;  set T within {U,U,U};                     # declarations for the input
var A {U} binary;                                 # variables
s.t. Sat {(x,y,z) in T}: A[x] + A[y] + A[z] = 1; # constraints
```

AMPL allows also to specify instances, typically in a separate file. An example for U and T (five elements and three triplets, respectively) is the following:

```
set U := a b c d e;
set T := (a,b,c) (c,d,e) (a,c,d);
```

The above example shows that, for some NP-complete problems, it is quite easy to write a linear model using only binary variables. In the next section we prove that BLP models exist for all problems in NP, although writing them is not always so simple.

3 The expressive power of binary linear programming

3.1 Normalization of ESO formulae

As explained in [7], instead of the general formula (1), we can restrict our attention to second-order formulae in the following form:

$$(\exists S_1, \ldots, S_h)(\forall \mathbf{X})(\exists \mathbf{Y})\ \psi(\mathbf{X}, \mathbf{Y}), \tag{3}$$

where $\mathbf{X}$ and $\mathbf{Y}$ are lists of first-order variables and $\psi(\mathbf{X}, \mathbf{Y})$ is a quantifier-free first-order formula involving relational symbols which belong to the set $\sigma \cup \{S_1, \ldots, S_h\} \cup \{=\}$. Since $\psi(\mathbf{X}, \mathbf{Y})$ can be put in Disjunctive Normal Form, i.e., disjunctions of conjunctions, in what follows we refer to the following form:

$$(\exists S_1, \ldots, S_h)(\forall \mathbf{X})(\exists \mathbf{Y})(\theta_1(\mathbf{X}, \mathbf{Y}) \vee \cdots \vee \theta_k(\mathbf{X}, \mathbf{Y})), \tag{4}$$

where $\theta_1, \ldots, \theta_k$ are conjunctions of literals, and each conjunction θ_i contains the occurrence of some variables among $\mathbf{X}, \mathbf{Y}$.

A conjunction $\theta_i(\mathbf{X}, \mathbf{Y})$ $(1 \leq i \leq k)$ of the kind occurring in formula (4) will be denoted as $\alpha_i(\mathbf{X}, \mathbf{Y}) \wedge \delta_i(\mathbf{X}, \mathbf{Y})$, where $\delta_i(\mathbf{X}, \mathbf{Y})$ is a conjunction of literals whose relational symbol are in $\{S_1, \ldots, S_h\}$, while $\alpha_i(\mathbf{X}, \mathbf{Y})$ is a conjunction of literals whose relational symbols are not from that set.

The first step of a method that transforms a formula of the kind (4) into a BLP model is the introduction of a modified ESO formula:

$$\begin{array}{lr}
(\exists S_1, \ldots, S_h, D_1, \ldots, D_k) & \\
\quad (\forall \mathbf{X})(\exists \mathbf{Y})\ (\alpha_1(\mathbf{X}, \mathbf{Y}) \wedge D_1(\mathbf{X}, \mathbf{Y})) \vee \cdots \vee (\alpha_k(\mathbf{X}, \mathbf{Y}) \wedge D_k(\mathbf{X}, \mathbf{Y})) & \wedge \\
\quad\quad (\forall \mathbf{X}, \mathbf{Y})\ D_1(\mathbf{X}, \mathbf{Y}) \equiv \delta_1(\mathbf{X}, \mathbf{Y}) & \wedge \\
\quad\quad\quad \cdots & \wedge \\
\quad\quad (\forall \mathbf{X}, \mathbf{Y})\ D_k(\mathbf{X}, \mathbf{Y}) \equiv \delta_k(\mathbf{X}, \mathbf{Y}) &
\end{array} \tag{5}$$

[1] In the third line, Sat is just a name for the constraint.

in which there are k new relational symbols $D_1, \ldots, D_k$ which are existentially quantified. Each symbol $D_i(\mathbf{X}, \mathbf{Y})$ $(1 \leq i \leq k)$ is defined as the conjunction $\delta_i(\mathbf{X}, \mathbf{Y})$. The advantage of formula (5) over formula (4) is that –as we show in what follows– the former generates *linear* constraints, while the latter generates *non-linear* constraints. The following lemma clarifies that the satisfiability problems for the two formulae are equivalent.

Lemma 1. *Given a database D, formula (4) is satisfiable if and only if formula (5) is satisfiable.*

Proof (only if part.) We assume that formula (4) is satisfiable for a given database D. Let $\Sigma_1, \ldots, \Sigma_h$ be a generic list of relations such that

$$(D, \Sigma_1, \ldots, \Sigma_h) \models (\forall \mathbf{X})(\exists \mathbf{Y})(\theta_1(\mathbf{X}, \mathbf{Y}) \vee \cdots \vee \theta_k(\mathbf{X}, \mathbf{Y})). \tag{6}$$

We build an assignment $\Delta_1, \ldots, \Delta_k$ to the relational variables $D_1, \ldots, D_k$ as follows. For each tuple $\mathbf{X}, \mathbf{Y}$ and for each i $(1 \leq i \leq k)$, $\Delta_i(\mathbf{X}, \mathbf{Y})$ is true if and only if $(\Sigma_1, \ldots, \Sigma_h) \models \delta_i(\mathbf{X}, \mathbf{Y})$. As a consequence, it holds that:

$$\begin{aligned}(D, \Sigma_1, \ldots, \Sigma_h, \Delta_1, \ldots, \Delta_k) \models (\forall \mathbf{X}, \mathbf{Y}) D_1(\mathbf{X}, \mathbf{Y}) \equiv \delta_1(\mathbf{X}, \mathbf{Y}) \ \wedge \\ \cdots \qquad \wedge \\ (\forall \mathbf{X}, \mathbf{Y}) D_k(\mathbf{X}, \mathbf{Y}) \equiv \delta_k(\mathbf{X}, \mathbf{Y})\end{aligned}$$

Because of (6), it also holds that:

$$\begin{aligned}(D, \Sigma_1, \ldots, \Sigma_h, \Delta_1, \ldots, \Delta_k) \models (\forall \mathbf{X})(\exists \mathbf{Y}) \ (\alpha_1(\mathbf{X}, \mathbf{Y}) \wedge D_1(\mathbf{X}, \mathbf{Y})) \\ \vee \cdots \vee \\ (\alpha_k(\mathbf{X}, \mathbf{Y}) \wedge D_k(\mathbf{X}, \mathbf{Y})),\end{aligned}$$

which concludes the proof.
Proof (if part.) We assume that formula (5) is satisfiable for a given database D. Let $\Sigma_1, \ldots, \Sigma_h, \Delta_1, \ldots, \Delta_k$ be a generic list of relations such that

$$\begin{aligned}(D, \Sigma_1, \ldots, \Sigma_h, \Delta_1, \ldots, \Delta_k) \models (\forall \mathbf{X})(\exists \mathbf{Y}) \ (\alpha_1(\mathbf{X}, \mathbf{Y}) \wedge D_1(\mathbf{X}, \mathbf{Y})) \\ \vee \cdots \vee \\ (\alpha_k(\mathbf{X}, \mathbf{Y}) \wedge D_k(\mathbf{X}, \mathbf{Y})) \ \wedge \\ (\forall \mathbf{X}, \mathbf{Y}) \quad D_1(\mathbf{X}, \mathbf{Y}) \equiv \delta_1(\mathbf{X}, \mathbf{Y}) \ \wedge \\ \cdots \qquad \wedge \\ (\forall \mathbf{X}, \mathbf{Y}) \quad D_k(\mathbf{X}, \mathbf{Y}) \equiv \delta_k(\mathbf{X}, \mathbf{Y})\end{aligned}$$

Because of the definition of formula (5), it follows that

$$(D, \Sigma_1, \ldots, \Sigma_h) \models (\forall \mathbf{X})(\exists \mathbf{Y})(\theta_1(\mathbf{X}, \mathbf{Y}) \vee \cdots \vee \theta_k(\mathbf{X}, \mathbf{Y})),$$

which concludes the proof. □

As shown by the above construction, each solution for the formula (4) is preserved after the transformation.

Example 1 (cont.) The ESO formula for satisfiability in the form (4) is:

$$(\exists S)\ (\forall x)(\exists y)\ V(x) \vee [P(x,y) \wedge \neg S(x) \wedge S(y)] \quad \vee \\ [N(x,y) \wedge \neg S(x) \wedge \neg S(y)] \tag{7}$$

For obtaining the form (5) we need two more relational variables D_1 and D_2 of arity 2, as follows:

$$\begin{array}{lll} (\exists S, D_1, D_2) & & \\ \quad (\forall x)(\exists y)\ \ V(x)\ \vee\ [P(x,y) \wedge D_1(x,y)]\ \vee\ [N(x,y) \wedge D_2(x,y)] \wedge & (a) & \\ \quad (\forall x\ y)\ \ D_1(x,y) \equiv \neg S(x) \wedge S(y) \hfill \wedge & (b) & (8) \\ \quad (\forall x\ y)\ \ D_2(x,y) \equiv \neg S(x) \wedge \neg S(y) & (c) & \end{array}$$

3.2 Translation of ESO formulae in BLP models

The second step is to prove that every ESO formula of the form (5) can be translated into an equivalent BLP model. Since such formulae contain quantifications of several kinds and various propositional connectives, we have to take into account several aspects. The most important thing we have to remember is that, in linear constraints, products of variables are not allowed. Instead, products of variables and non-variables are allowed. In what follows, we use the terminology of [3]. The translation rules are, intuitively, the following:

– Unquantified relational symbols, i.e., those in the database, correspond to *sets*, i.e., collections of uninterpreted symbols.
– Existentially quantified relational symbols, i.e., those representing the search space in which solutions are to be found, correspond to collection of binary *variables*.
– Positive literals, such as $P(x,y)$ and $S(y)$ in formula (8), can be directly mapped into binary terms.
– As for quantifier-free formulae, the general idea is to translate disjunctions ($\vee$) into sums, conjunctions ($\wedge$) into products, and negations ($\neg$) into difference from 1.
– First-order existential quantification can be modeled taking into account that the existential quantifier is an iterated disjunction. As a consequence, we obviously translate it into a sum over all elements of the Cartesian product of the Herbrand universe, provided that terms not depending on the quantified variable are taken out of the sum.
– First-order universal quantification can be easily modeled declaring constraints for each element of the Cartesian product of the Herbrand universe, with the appropriate arity.
– Finally, a constraint is true iff the corresponding integer expression is assigned a value greater than or equal to 1.

In the following, we formalize the transformation. First of all, we need some further notation; referring to formula (5):

– each conjunction $\alpha_i(\mathbf{X},\mathbf{Y})$ is expanded as

$$a_i^1(\mathbf{X},\mathbf{Y}) \wedge \cdots \wedge a_i^{l_i}(\mathbf{X},\mathbf{Y});$$

– each conjunction $\delta_i(\mathbf{X},\mathbf{Y})$ is expanded as

$$d_i^1(\mathbf{X},\mathbf{Y}) \wedge \cdots \wedge d_i^{m_i}(\mathbf{X},\mathbf{Y}).$$

We now introduce integer terms corresponding to the above literals:

– for each i $(1 \leq i \leq k)$ and each j $(1 \leq j \leq l_i)$, literal $a_i^j(\mathbf{X},\mathbf{Y})$ corresponds to integer term $b_i^j(\mathbf{X},\mathbf{Y})$, defined as:
  - $a_i^j(\mathbf{X},\mathbf{Y})$, if the literal is positive, or
  - $1 - a_i^j(\mathbf{X},\mathbf{Y})$, if the literal is negative;
– for each i $(1 \leq i \leq k)$ and each j $(1 \leq j \leq m_i)$, literal $d_i^j(\mathbf{X},\mathbf{Y})$ corresponds to integer term $c_i^j(\mathbf{X},\mathbf{Y})$, defined as:
  - $d_i^j(\mathbf{X},\mathbf{Y})$, if the literal is positive, or
  - $1 - d_i^j(\mathbf{X},\mathbf{Y})$, if the literal is negative.

Without loss of generality, we assume that in formula (4) the conjuncts θ_i's are sorted in such a way that conjuncts in which $\mathbf{Y}$ does not occur have lower index. In other words, there is an index r $(0 \leq r \leq k)$ such that for each i $(1 \leq i \leq r)$ $\mathbf{Y}$ does not occur in θ_i, and for each j $(r+1 \leq j \leq k)$ $\mathbf{Y}$ occurs in θ_j. Referring to formula (7) of Example 1, $r = 1$.

We are now ready to show the translation of an ESO formula into a BLP model. Given a formula of the form (5), we define the following model:

variables:

– denoting with a_i $(1 \leq i \leq h)$ the arity of S_i:

$$S_1 \in \{0,1\}^{|U|^{a_1}}, \ldots, S_h \in \{0,1\}^{|U|^{a_h}}, \tag{9}$$

– denoting with b_i $(1 \leq i \leq k)$ the arity of D_i:

$$D_1 \in \{0,1\}^{|U|^{b_1}}, \ldots, D_k \in \{0,1\}^{|U|^{b_k}}, \tag{10}$$

constraints:

– for each $\mathbf{X}$,

$$\begin{array}{l} b_1^1(\mathbf{X}) \times \cdots \times b_1^{l_1}(\mathbf{X}) \times D_1(\mathbf{X}) + \cdots + \\ b_r^1(\mathbf{X}) \times \cdots \times b_r^{l_r}(\mathbf{X}) \times D_r(\mathbf{X}) + \\ \left(\begin{array}{l} \sum\limits_{\mathbf{Y}} b_{r+1}^1(\mathbf{X},\mathbf{Y}) \times \cdots \times b_{r+1}^{l_{r+1}}(\mathbf{X},\mathbf{Y}) \times D_{r+1}(\mathbf{X},\mathbf{Y}) + \cdots + \\ \quad b_k^1(\mathbf{X},\mathbf{Y}) \times \cdots \times b_k^{l_k}(\mathbf{X},\mathbf{Y}) \times D_k(\mathbf{X},\mathbf{Y}) \end{array} \right) \\ \hfill \geq 1 \end{array} \tag{11}$$

– for each $\mathbf{X}$ and $\mathbf{Y}$,

$$\begin{array}{rcl} c_1^1(\mathbf{X},\mathbf{Y}) & \geq & D_1(\mathbf{X},\mathbf{Y}) \\ & \cdots & \\ c_1^{m_1}(\mathbf{X},\mathbf{Y}) & \geq & D_1(\mathbf{X},\mathbf{Y}) \\ D_1(\mathbf{X},\mathbf{Y}) & \geq & c_1^1(\mathbf{X},\mathbf{Y}) + \cdots + c_1^{m_1}(\mathbf{X},\mathbf{Y}) + 1 - m_1 \\ & \vdots & \\ c_k^1(\mathbf{X},\mathbf{Y}) & \geq & D_k(\mathbf{X},\mathbf{Y}) \\ & \cdots & \\ c_k^{m_k}(\mathbf{X},\mathbf{Y}) & \geq & D_k(\mathbf{X},\mathbf{Y}) \\ D_k(\mathbf{X},\mathbf{Y}) & \geq & c_k^1(\mathbf{X},\mathbf{Y}) + \cdots + c_k^{m_k}(\mathbf{X},\mathbf{Y}) + 1 - m_k \end{array} \qquad (12)$$

We note that all variables in (9-10) are binary, and that all constraints in (11-12) are linear (products in (11) are between non-variables). As a consequence, (9-12) defines a BLP model. The following theorem shows that the satisfiability problems for the formula (5) and for the above BLP model are equivalent.

Theorem 1. *Given a database D, formula (5) is satisfiable if and only if BLP model (9-12) is satisfiable.*

Proof (only if part.) We assume that formula (5) is satisfiable for a given database D, i.e., there is a list L of relations corresponding to $S_1, \ldots, S_h, D_1, \ldots, D_k$ that, along with D, satisfies the formula obtained eliminating second-order quantifiers from formula (5).

We define an assignment to binary variables in (9-10) in the obvious way, i.e., for each $\mathbf{X}$, $\mathbf{Y}$, i $(1 \leq i \leq h)$ and j $(1 \leq j \leq k)$, $S_i(\mathbf{X},\mathbf{Y})$ $[D_j(\mathbf{X},\mathbf{Y})]$ is assigned to 1 iff the corresponding logical term is assigned to true, and to 0 otherwise.

We prove that the above assignment satisfies all constraints in (11). By the above assumption, we know that for each tuple $\mathbf{X}$ there is a tuple $\mathbf{Y}$ and an index j $(1 \leq j \leq k)$ such that $\alpha_j(\mathbf{X},\mathbf{Y}) \wedge D_j(\mathbf{X},\mathbf{Y})$ is assigned to true by L, D. There are two cases:

- $1 \leq j \leq r$: then $b_j^1(\mathbf{X}) \times \cdots \times b_{l_j}^1(\mathbf{X}) \times D_j(\mathbf{X})$ evaluates to 1;
- $r+1 \leq j \leq k$: then $b_j^1(\mathbf{X},\mathbf{Y}) \times \cdots \times b_{l_j}^1(\mathbf{X},\mathbf{Y}) \times D_j(\mathbf{X},\mathbf{Y})$ evaluates to 1.

In both cases, the left hand side of each constraint in (11) is greater than or equal to 1.

We now prove that the above assignment satisfies also all constraints in (12). By the above assumption, we know that for each $\mathbf{X}$, $\mathbf{Y}$ and i $(1 \leq i \leq k)$, $D_i(\mathbf{X},\mathbf{Y}) \equiv d_i^1(\mathbf{X},\mathbf{Y}) \wedge \cdots \wedge d_i^{m_i}(\mathbf{X},\mathbf{Y})$ is assigned to true by L, D. There are two cases:

- $D_i(\mathbf{X},\mathbf{Y})$ is assigned to true: then each $d_i^j(\mathbf{X},\mathbf{Y})$ $(1 \leq j \leq m_i)$ is true as well;
- $D_i(\mathbf{X},\mathbf{Y})$ is assigned to false: then at least one $d_i^j(\mathbf{X},\mathbf{Y})$ $(1 \leq j \leq m_i)$ is false as well.

In both cases constraints (12) are satisfied, and the proof is concluded.
Proof (if part.) We assume that the BLP model (9-12) is satisfiable for a given database D, i.e., there is an assignment A to variables in (9-10) such that constraints in (11-12) are satisfied.

We define an interpretation of relations in (5) in the obvious way, i.e., for each $\mathbf{X}$, $\mathbf{Y}$, i $(1 \leq i \leq h)$ and j $(1 \leq j \leq k)$, $S_i(\mathbf{X}, \mathbf{Y})$ $[D_j(\mathbf{X}, \mathbf{Y})]$ is interpreted as true iff the corresponding binary variable is assigned to 1, and to false otherwise.

Since all constraints in (11) are satisfied, there must be at least one product in the left hand side of (11) which is assigned to 1 by D, A. This implies that the sub-formula $(\forall \mathbf{X})(\exists \mathbf{Y})(\alpha_1(\mathbf{X}, \mathbf{Y}) \wedge D_1(\mathbf{X}, \mathbf{Y})) \vee \cdots \vee (\alpha_k(\mathbf{X}, \mathbf{Y}) \wedge D_k(\mathbf{X}, \mathbf{Y}))$ of (5) is satisfied by the interpretation.

Analogously, since all constraints in (12) are satisfied, for each $\mathbf{X}$, $\mathbf{Y}$ and i $(1 \leq i \leq k)$ the formula $D_i(\mathbf{X}, \mathbf{Y}) \equiv d_i^1(\mathbf{X}, \mathbf{Y}) \wedge \cdots \wedge d_i^{m_i}(\mathbf{X}, \mathbf{Y})$ must be true, and the proof is concluded. $\square$

As shown by the above construction, each solution for the formula (5) is preserved after the transformation.

Example 1 (cont.) The translation in a BLP model of formula (8) using the AMPL syntax is the following:

```
set V;                # names of propositional variables
set C;                # names of propositional clauses
set U := V union C;   # Herbrand universe
set N within {C,V};   # negative occurrences of variables in clauses
set P within {C,V};   # positive occurrences of variables in clauses

var S {U} binary;     # satisfying assignment
var D1 {U,U} binary;  # auxiliary guessed relation
var D2 {U,U} binary;  # auxiliary guessed relation

s.t. A {x in U}:             # translation of constraint (8a)
         (if x in V then 1) +
          sum {y in U} ((D1[x,y] * if (x,y) in P then 1) +
                     (D2[x,y] * if (x,y) in N then 1)) >= 1;

s.t. D1_1 {x in U, y in U}:  # translation of constraint (8b)
         1 - S[x] >= D1[x,y];            # D1[x,y] IMPLIES !S[x]
s.t. D1_2 {x in U, y in U}:
         S[y] >= D1[x,y];                # D1[x,y] IMPLIES S[y]
s.t. D1_3 {x in U, y in U}:
         S[x] + 1 - S[y] + D1[x,y] >= 1; # !S[x]&&S[y] IMPLIES D1[x,y]

s.t. D2_1 {x in U, y in U}:  # translation of constraint (8c)
         1 - S[x] >= D2[x,y];            # D2[x,y] IMPLIES !S[x]
s.t. D2_2 {x in U, y in U}:
         1 - S[y] >= D2[x,y];            # D2[x,y] IMPLIES !S[y]
s.t. D2_3 {x in U, y in U}:
         S[x] + S[y] + D2[x,y] >= 1;     # !S[x]&&!S[y] IMPLIES D2[x,y]
```

Database literals such as $V(x)$ and non-database ones such as $S(x)$ have different translations: the former must be translated as `if x in V then 1`, while the latter is translated as `S[x]`. Apart from such minor syntactic peculiarities of AMPL, formula (8) is translated in a modular way.

We note that all constraints are linear, since there are no products among variables, i.e., terms originating from the translation of existentially quantified relations. The reason of introducing normal form (5) is that the same translation applied to formulae of the form (4) may introduce non-linear constraints. As an example, the same translation applied to the second disjunct of non-normalized formula (7) would yield the integer expression:

```
(if (x,y) in P then 1) * (1 - S[x]) * S[y]
```

which is clearly non-linear. For the same reason, an equivalence such as (8b) which involves non-database literals, is split into several implications, all of them admitting a linear translation.

4 Conclusions and future work

The transformation exhibited in Section 3 shows that the language of BLP models is a notational variant of ESO. This implies that it is in principle possible to model all problems in the complexity class NP by means of BLP. It is important to remark that the translation from ESO to BLP is done at the intensional level, i.e., not considering data, but just problem specifications.

Practical considerations about the best way to perform the translation deserve further research. In particular, it would be interesting to know whether new existentially quantified relational symbols introduced by the transformation of Section 3.1 (which may lead to an inefficient BLP model) can be avoided. Moreover, it would be interesting to study the expressive power of richer CP languages such as OPL [10], which allow integer variables and a richer syntax for expressing constraints.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison Wesley Publ. Co., Reading, Massachussetts, 1995.
2. R. Fagin. Generalized First-Order Spectra and Polynomial-Time Recognizable Sets. In R. M. Karp, editor, *Complexity of Computation*, pages 43–74. AMS, 1974.
3. Robert Fourer, David M. Gay, and Brian W. Kernigham. *AMPL: A Modeling Language for Mathematical Programming.* International Thomson Publishing, 1993.
4. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, San Francisco, Ca, 1979.
5. ILOG AMPL CPLEX system version 7.0 user's guide. Available at `www.ilog.com`, 2001.
6. R. G. Jeroslow and J. Wang. Solving propositional satisfiability problems. *Ann. of Mathematics and Artificial Intelligence*, 1:167–187, 1990.

7. P. G. Kolaitis and C. H. Papadimitriou. Why not negation by fixpoint? *J. of Computer and System Sciences*, 43:125–144, 1991.
8. Irvin J. Lustig and Jean-François Puget. Program $\neq$ program: Constraint programming and its relationship to mathematical programming. *Interfaces*, 31(6):29–53, 2001.
9. Philippe Refalo. Linear formulation of constraint programming models and hybrid solvers. In *Proc. of CP 2000*, LNCS, pages 369–383. Springer, 2000.
10. Pascal Van Hentenryck. *The OPL Optimization Programming Language*. The MIT Press, 1999.

The Inference Problem for Propositional Circumscription of Affine Formulas is coNP-complete

Arnaud Durand
LACL, Dept. of Computer Science
Université Paris 12
94010 Créteil, France
durand@univ-paris12.fr

Miki Hermann
LIX (CNRS, UMR 7650)
École Polytechnique
91128 Palaiseau cedex, France
hermann@lix.polytechnique.fr

Abstract

We prove that the inference problem of propositional circumscription for affine formulas is coNP-complete, settling this way an open question in the complexity analysis of this problem. We also show that the considered problem becomes polynomial-time decidable if only a single literal has to be inferred from an affine formula. Our intractability result has also a relation to other complexity results in coding theory.

1 Introduction and Summary of Results

Circumscription, introduced by McCarthy [McC80], is a well-developed formalism of common-sense reasoning and extensively studied by the artificial intelligence community. The key idea behind circumscription is that one is interested in the *minimal models* of formulas, since they are the ones that have as few "exceptions" as possible and, therefore, embody common sense. Moreover, propositional circumscription inference has been shown by Gelfond *et al.* [GPP89] to coincide with reasoning under the extended closed world assumption, which is one of the main formalisms for reasoning with incomplete information. In the context of Boolean logic, circumscription amounts to the study of models of Boolean formulas that are *minimal* with respect to the *pointwise partial order* on models.

Several algorithmic problems have been studied in connection with propositional circumscription: among them the *model checking* and the *inference* problems. Given a propositional formula φ and a truth assignment s, the model checking problem asks whether s is a minimal model of φ. Given two propositional formulas φ and ψ, the inference problem asks whether ψ is true in every minimal model of φ. Cadoli proved in [Cad92] the model checking problem to be coNP-complete, whereas Kirousis and Kolaitis settled in [KK01a] the question of the dichotomy theorem for this problem. The inference problem was proved Π_2P-complete by Eiter and Gottlob in [EG93]. Cadoli and Lenzerini proved in [CL94] that the inference problem becomes coNP-complete if φ is a Krom or a dual Horn formula. The complexity of the inference problem for affine formulas remained open for ten years. It was known that the problem is in coNP, but there was no proved lower bound.

Beyond the already mentioned applications, minimal models of affine formulas are at the heart of several intriguing computational problems. The complexity of enumerating minimal models of affine formulas was left open by Kavvadias *et al.* [KSS00]. Together with the hypergraph transversal [EG95] problem, they represent the last subcases of minimal model enumeration whose precise complexity is unknown. Another important application constitutes the domain of secret sharing schemes [Sha79], complexity issues in coding theory [BMvT78, Bar98], and minimum distance decoding [AB98].

In fact, this paper is a partial result of our effort to find an output-polynomial algorithm for enumerating the minimal models of affine formulas. Following the result of Berlekamp *et al.* [BMvT78], it is clear that we cannot develop an output-polynomial algorithm for this enumeration problem by producing consecutive minimal models of the affine system with increasing Hamming weight, unless P = NP. Another natural approach consists of producing partial assignments to the variables that are extended to minimal models afterwards. However, as our result indicates, this new approach does not lead to an output-polynomial algorithm either, unless the same collapse occurs.

In this paper, we settle the complexity of the inference problem for the propositional circumscription of affine formulas, proving that the problem is coNP-complete. First, we prove a new criterion for determining minimal models of affine formulas in polynomial time. This new criterion is then extensively used in the coNP-hardness proof for the inference problem of affine formulas. Finally, we prove that the restriction of the affine inference problem with ψ being a single literal is decidable in polynomial time.

2 Preliminaries

Let $s = (s_1, \ldots, s_n)$ and $s' = (s'_1, \ldots, s'_n)$ be two Boolean vectors from $\{0,1\}^n$. We write $s < s'$ to denote that $s \neq s'$ and $s_i \leq s'_i$ holds for every $i \leq n$. Let $\varphi(x_1, \ldots, x_n)$ be a Boolean formula having $x_1, \ldots, x_n$ as its variables and let $s \in \{0,1\}^n$ be a truth assignment. We say that s is a *minimal model* of φ if s is a satisfying truth assignment of φ and there is no satisfying truth assignment s' of φ that satisfies the relation $s < s'$. This relation is called the *pointwise partial order* on models.

Let $\varphi(x_1, \ldots, x_n)$ be a propositional formula in conjunctive normal form. We say that $\varphi(x)$ is *Horn* if φ has at most one positive literal per clause, *dual Horn* if φ has at most one negative literal per clause, *Krom* if φ has at most two literals per clause, and *affine* if φ is a conjunction of clauses of the type $x_1 \oplus \cdots \oplus x_n = 0$ or $x_1 \oplus \cdots \oplus x_n = 1$, where $\oplus$ is the exclusive-or logical connective, what is equivalent to an affine system of equations $S\colon Ax = b$ over $\mathbb{Z}_2$.

Let φ and ψ be two propositional formulas in conjunctive normal form. We say that ψ follows from φ in propositional circumscription, denoted by $\varphi \models_{\min} \psi$, if ψ is true in every minimal model of φ. Since ψ is a conjunction $c_1 \wedge \cdots \wedge c_k$ of clauses c_i, then $\varphi \models_{\min} \psi$ if and only if $\varphi \models_{\min} c_i$ for each i. Hence we can restrict ourselves to consider only a single clause instead of a formula ψ at the right-hand side of the propositional inference problem $\varphi \models_{\min} c$. We can further restrict the clause c to one containing only negative literals $c = \neg u_1 \vee \cdots \vee \neg u_n$, as it was showed in [KK01b].

If x and y are two vectors, we denote by $z = xy$ the vector obtained by concatenation of x and y. Let $S\colon Az = b$ be a $k \times n$ affine system of equations over $\mathbb{Z}_2$. Without loss of generality, we assume that the system S is in standard form, i.e., that the matrix A has the form $(I\ B)$, where I is the $k \times k$ identity matrix and B is an arbitrary $k \times (n-k)$ matrix of full column rank. For convenience, we denote by x the variables from z associated with I and by y the ones associated with B. Hence, we consider affine systems of the form $S\colon (I\ B)(xy) = b$.

If A is a $k \times n$ matrix, we denote by $A(i,j)$ the element of A positioned at row i and column j. The vector forming the row i of the matrix A is denoted by $A(i,-)$, whereas the column vector j of A is denoted by $A(-,j)$. Let $I \subseteq \{1,\ldots,k\}$ and $J \subseteq \{1,\ldots,n\}$ be two index sets. Then $A(I,-)$ denotes the submatrix of A restricted to the rows I. Similarly, $A(-,J)$ is then the submatrix of A restricted to the columns J, whereas $A(I,J)$ stands for the submatrix of A restricted to the rows I and columns J. There are also two matrices with a special notation: the $k \times k$ identity matrix I_k and the $k \times n$ all-zero matrix O_k^n.

For a $k \times n$ affine system $S\colon Az = b$ over $\mathbb{Z}_2$, an index set $J = \{j_1, \ldots, j_m\} \subseteq \{1, \ldots, n\}$ of cardinality $|J| = m$, and a Boolean vector $v = (v_1, \ldots, v_m)$ of length m, we denote by $S[J/v]$ the new system $S'\colon A'z' = b'$ formed by replacing each variable z_{j_i} by the value v_i. We also denote by $one(v) = \{i \mid v_i = 1\}$ and $zero(v) = \{i \mid v_i = 0\}$ the positions in the vector v assigned to the values 1 and 0, respectively. The Hamming weight $weight(v)$ of a vector v is equal to the cardinality of the set $one(v)$, i.e., $weight(v) = |one(v)|$.

Suppose that s is a variable assignment for the variables y, i.e., for each $y_i \in y$ there exists a value $s(y_i) \in \mathbb{Z}_2$. The vector s is a *partial assignment* for variables $z = xy$. An *extension* of the vector s is a variable assignment $\bar{s}$ for each variable from z, i.e., for each $z_i \in z$ there exists a value $\bar{s}(z_i) \in \mathbb{Z}_2$, such that $s(y_i) = \bar{s}(y_i)$ for each y_i. If the affine system $S\colon (I\ B)(xy) = b$ is in the standard form and s is a variable assignment for the variables y then the extension $\bar{s}$ to a solution of the system S is *unique*. In fact, if the variables y in the system $S\colon (I\ B)(xy) = b$ have been assigned, then the values for the variables x are already determined. In connection with the previous notions we define the following two index sets

$$eq(s) = \{i \mid (Bs)_i = b_i\} \qquad \text{and} \qquad neq(s) = \{i \mid (Bs)_i \neq b_i\},$$

where $b = (b_1, \ldots, b_k)$ and $(Bs)_i$ means the i-th position of the vector obtained after multiplication of the matrix B by the vector s. The set $eq(s)$ (resp. $neq(s)$) is the subset of row indices i for which the unique extension $\bar{s}$ satisfies the equality $\bar{s}(x_i) = 0$ (resp. $\bar{s}(x_i) = 1$). It is clear that $eq(s) \cap neq(s) = \emptyset$ and $eq(s) \cup neq(s) = \{1, \ldots, k\}$ hold for each s.

3 A New Criterion For Affine Minimality

There exists a straightforward method to determine in polynomial time whether a solution s is minimal for an affine system S over $\mathbb{Z}_2$. We propose here an alternative method especially well-suited to decide whether an extension $\bar{s}$ is a minimal solution of S.

Proposition 3.1 *Let $S\colon (I\ B)(xy) = b$ be an affine $k \times n$ system over $\mathbb{Z}_2$ and let s be a Boolean vector of length $n - k$. The extension $\bar{s}$ is a minimal solution of S if and only if $B(eq(s), one(s))$ is a matrix of column rank $weight(s)$, i.e., all its columns are linearly independent.*

Proof: Suppose that $\bar{s}$ is minimal and the matrix $B(eq(s), one(s))$ has the column rank smaller than $weight(s)$. This means that the columns of $B(eq(s), one(s))$ are linearly dependent, therefore there exists a subset $J \subseteq one(s)$, such that $\sum_{j \in J} B(eq(s), j) = \vec{0}$ holds. Let t be a Boolean vector satisfying the condition $one(t) = one(s) \setminus J$. The columns of the matrix $B(eq(s), one(s))$ can be partitioned into two sets: those in J and those in $one(t)$. Knowing that the columns in J add up to the zero vector $\vec{0}$, we derive the following equality.

$$\sum_{j \in one(s)} B(eq(s), j) = \sum_{j \in one(t)} B(eq(s), j) + \sum_{j \in J} B(eq(s), j) = \sum_{j \in one(t)} B(eq(s), j)$$

The vector t is smaller than s in the pointwise order. We will show that also the extensions $\bar{s}$ and $\bar{t}$ satisfy the relation $\bar{t} < \bar{s}$. For each row $i \in eq(s)$, the coefficients $B(i, j)$ sum up to the value b_i, i.e., that $\sum_{j \in one(s)} B(i, j) = \sum_{j \in one(t)} B(i, j) = b_i$. Recall that each variable in the vector x occurs in the system S exactly once, because of the associated identity matrix I_k. Since already the assignments s and t to the variables y sum up to the value b_i, this determines the value of the variable x_i in the extensions $\bar{s}$ and $\bar{t}$ to be $\bar{s}(x_i) = \bar{t}(x_i) = 0$ for each row $i \in eq(s)$. In the same spirit, the assignment s to the variables y sums up to the value $1 - b_i$ for each row $i \in neq(s)$, what determines the value of the variable x_i in the extension $\bar{s}$ to be $\bar{s}(x_i) = 1$. Therefore we have $\bar{t}(x_i) \leq \bar{s}(x_i) = 1$ for each row $i \in neq(s)$. This shows that $\bar{t}$ is a solution of S smaller than $\bar{s}$, what contradicts our assumption that $\bar{s}$ is minimal.

Conversely, suppose that the matrix $B(eq(s), one(s))$ has the column rank $weight(s)$ but $\bar{s}$ is not minimal. The latter condition implies that there exists a variable assignment t, such that the extension $\bar{t}$ is a solution of S satisfying the relation $\bar{t} < \bar{s}$. Let $J = one(\bar{s}) \smallsetminus one(\bar{t})$ be the set of positions on which the extensions $\bar{s}$ and $\bar{t}$ differ. Both extensions $\bar{s}$ and $\bar{t}$ are solutions of S, therefore we have $(I\ B)\bar{s} + (I\ B)\bar{t} = \sum_{j \in J}(I\ B)(-,j) = \vec{0}$. The index set J can be partitioned into two disjoint sets J_1 containing the positions smaller or equal to k, that are associated with the identity matrix I, and the set J_2 containing the positions greater than k, that are associated with the matrix B. Hence the inclusion $J_2 \subseteq one(s)$ holds. The columns of the identity matrix I are linearly independent, therefore the set J_2 must be nonempty in order to get the above sum equal to $\vec{0}$. The partition of J implies the equality $\sum_{j \in J_1} I(-,j) + \sum_{j \in J_2} B(-,j) = \vec{0}$. The restriction of this equality to the rows in $eq(s)$ yields $\sum_{j \in J_1} I(eq(s),j) + \sum_{j \in J_2} B(eq(s),j) = \vec{0}$. The vector $\bar{s}$ is a solution of S and for each row $i \in eq(s)$ we have $\bar{s}(x_i) = 0$, since already the values $s(y_j)$ with $j \in J_2$ sum up to b_i. This implies together with the previous equation that $i \notin J_1$, since $i \leq k$ holds, and for all indices $j \in J_1$ the column $I(eq(s),j)$ is the all-zero vector. This yields the equality $\sum_{j \in J_1} I(eq(s),j) = \vec{0}$, what implies the final equality $\sum_{j \in J_2} B(eq(s),j) = \vec{0}$. Recalling that J_2 is a subset of the columns $one(s)$, this contradicts the fact that the matrix $B(eq(s), one(s))$ has the column rank equal to $weight(s)$. $\square$

4 Extension and Inference Problems

In this paper we will be interested in the complexity of the inference problem of propositional circumscription with affine formulas. Since affine propositional formulas are equivalent to affine systems $S\colon Az = b$ over $\mathbb{Z}_2$, this problem can be formulated as follows.

Problem: AFFINF
Input: An affine system $S\colon Az = b$ over $\mathbb{Z}_2$ with a Boolean $k \times n$ matrix A, a Boolean vector b of length k, a variable vector $z = (z_1, \ldots, z_n)$, and a negative clause $c = \neg u_1 \vee \cdots \vee \neg u_m$, where $u_i \in z$ holds for each i.
Question: Does $S \models_{\min} c$ hold?

Another interesting problem, closely related to the previous one, is the problem of extending a Boolean vector to a minimal solution of an affine system.

Problem: MINEXT
Input: An affine system $S\colon Az = b$ over $\mathbb{Z}_2$ with a Boolean $k \times n$ matrix A, a Boolean vector b of length k, a variable vector $z = (z_1, \ldots, z_n)$, and a partial assignment s for the variables y, where $z = xy$.
Question: Can s be extended to a vector $\bar{s}$, such that $\bar{s}$ is a minimal solution of the system S?

In fact, the minimal extension problem appears naturally within algorithms enumerating minimal solutions. For any given class of propositional formulas, when the corresponding minimal extension problem is polynomial-time decidable, then there exists an algorithm that enumerates each consecutive pair of minimal solutions with polynomial delay.

To derive the lower bound of the complexity of the latter problem, we need to consider the following well-known NP-complete problem.

Problem: POSITIVE 1-IN-3 SAT
Input: A propositional formula φ in conjunctive normal form with three positive literals per clause.
Question: Is there a truth assignment to the variable of φ, such that exactly one literal is assigned to *true* and the two others are assigned to *false* in every clause?

Theorem 4.1 MINEXT *is* NP-*complete. The problem remains* NP-*complete even if the partial assignment s contains no 0.*

Proof: Membership of MINEXT in NP is obvious. For the lower bound, we construct a polynomial reduction from the problem POSITIVE 1-IN-3 SAT.

Let $\varphi(x_1, \dots, x_n)$ be a propositional formula in conjunctive normal form $c_1 \wedge \cdots \wedge c_m$ with the clauses $c_i = x_i^1 \vee x_i^2 \vee x_i^3$. We construct an affine system S: $(I\ B)(zxy) = b$, where I is the $(4m+n) \times (4m+n)$ identity matrix, z, x, and y are variable vectors of respective lengths $4m+n$, n, and $3m$, and B is a special $(4m+n) \times (3m+n)$ matrix encoding the formula φ. We also construct a partial assignment s and show that the formula φ has a model satisfying exactly one variable per clause if and only if s can be extended to a minimal solution of S.

The matrix B is composed from six blocks as follows

$$\begin{pmatrix} B_1^1 & B_1^2 \\ B_2^1 & B_2^2 \\ B_3^1 & B_3^2 \end{pmatrix}$$

The matrix B_1^1 of size $m \times n$ is the clause-variable incidence matrix of the formula φ, i.e., $B_1^1(i,j) = 1$ holds if and only if $x_j \in c_i$. The matrix B_1^2 of size $m \times 3m$ is the identity matrix I_m with each column tripled, i.e., it verifies the conditions $B_1^2(i, 3(i-1)+1) = B_1^2(i, 3(i-1)+2) = B_1^2(i, 3i) = 1$ for all i and $B_1^2(i,j) = 0$ otherwise. The matrix B_2^1 of size $3m \times n$ encodes the polynomials $x_i^1 + x_i^2$, $x_i^2 + x_i^3$, and $x_i^3 + x_i^1$ over $\mathbb{Z}_2$ for each clause $c_i = x_i^1 \vee x_i^2 \vee x_i^3$. This encoding is done for each $i = 1, \dots, m$ in three consecutive rows. Hence, we have $B_2^1(3i, i_1) = B_2^1(3i, i_2) = 1$, $B_2^1(3i+1, i_2) = B_2^1(3i+1, i_3) = 1$, and $B_2^1(3i+2, i_3) = B_2^1(3i+2, i_1) = 1$, where i_j is the position of the variable x_i^j in the vector $x = (x_1, \dots, x_n)$. Otherwise we have $B_2^1(3i+q, j) = 0$ for $q = 0, 1, 2$ and $j \neq i_1, i_2, i_3$. In another words, the rows $B_2^1(3i, -)$, $B_2^1(3i+1, -)$, and $B_2^1(3i+2, -)$ are the incidence vectors of the polynomials $x_i^1 + x_i^2$, $x_i^2 + x_i^3$, and $x_i^3 + x_i^1$, respectively. The matrix B_2^2 of size $3m \times 3m$ is the identity matrix I_{3m}. The matrix B_3^1 of size $n \times n$ is the identity matrix I_n, whereas the matrix B_3^2 of size $n \times 3m$ is the all-zero matrix O_n^{3m}. Note that due to the blocks B_2^2 and B_3^1, that are identity matrices, as well as the block B_3^2 that is an all-zero matrix, the matrix B has the column rank $n + 3m$. Denote by B_1 the submatrix of B restricted to the first m rows, i.e., $B_1 = B(\{1, \dots, m\}, -)$. Analogously, we define $B_2 = B(\{m+1, \dots, 4m\}, -)$ and $B_3 = B(\{4m+1, \dots, 4m+n\}, -)$. In the same spirit, we denote by $B^1 = B(-, \{1, \dots, n\})$ and $B^2 = B(-, \{n+1, \dots, n+3m\})$ the left and the right part of the columns, respectively, of the matrix B.

The vector b of length $4m+n$ in the system S is a concatenation of three vectors b_1, b_2, and b_3, where b_1 is the all-zero vector of length m, b_2 is the all-zero vector of length $3m$, and b_3 is the all-one vector of length m. The parts b_i of the vector b correspond to the row blocks B_i of the matrix B for $i = 1, 2, 3$. Figure 1 describes the constructed matrix B and vector b.

Finally, we set the vector s of size $3m$ to be equal to one in each coordinate, i.e., $s(y_i) = 1$ for each $i = 1, \dots, 3m$. The Hamming weight of s is equal to $\mathit{weight}(s) = 3m$.

Let v be a model of the formula φ that satisfies exactly one literal per clause. We will prove that when we append the all-one vector s to v, forming the vector $t = vs$, then the extension $\bar{t}$ is a

$$\begin{array}{c|c|c|c c|c|c}
 & B^1 & B^2 & & & & \\ \hline
B_1 & \varphi & \begin{matrix}111 & & \\ & \ddots & \\ & & 111\end{matrix} & m & & \begin{matrix}0\\ \vdots \\ 0\end{matrix} & b_1 \\ \hline
B_2 & \begin{matrix}\forall i \leq m \\ \\ x_i^1 + x_i^2 \\ x_i^2 + x_i^3 \\ x_i^3 + x_i^1\end{matrix} & I_{3m} & 3m & & \begin{matrix}0\\ \vdots \\ 0\end{matrix} & b_2 \\ \hline
B_3 & I_n & O_n^{3m} & n & & \begin{matrix}1\\ \vdots \\ 1\end{matrix} & b_3 \\ \hline
 & n & 3m & & & &
\end{array}$$

Figure 1: Matrix B and the associated vector b

minimal solution of S. Let us study the set $eq(t)$. Since every clause $c_i = x_i^1 \vee x_i^2 \vee x_i^3$ of φ is satisfied, the sum of literal values is equal to $v(x_i^1) + v(x_i^2) + v(x_i^3) = 1$. Moreover, for each $j = 1, \ldots, m$ we have $s(x_j) = 1$, therefore all m rows of B_1 belong to $eq(t)$. Exactly two of the polynomials $x_i^1 + x_i^2$, $x_i^2 + x_i^3$, and $x_i^3 + x_i^1$ are evaluated to 1 for each clause c_i and for each $j = 1, \ldots, 3m$ we have $s(x_j) = 1$, what implies that exactly $2m$ rows from B_2 belong to the set $eq(t)$. The row i of B_1 and the rows $3(i-1)+1$, $3(i-1)+2$, and $3i$ of B_2 correspond to the clause c_i. Form the corresponding row index set $I(i) = \{i,\ m+3(i-1)+1,\ m+3(i-1)+2,\ m+3i\}$ for a given i. Consider the restriction of the block B^2 to the rows $I(i)$. This restriction $B^2(I(i), -)$ will have plenty of all-zero columns. Keep only the columns containing at least one value 1. These columns will be $3(i-1)+1$, $3(i-1)+2$, and $3i$. Form the corresponding column index set $J(i) = \{n+3(i-1)+1,\ n+3(i-1)+2,\ n+3i\}$ for a given i. The restriction of B to the rows $I(i)$ and columns $J(i)$ is the matrix

$$B(I(i), J(i)) \quad = \quad \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = B^*(i).$$

Note that the first row of $B^*(i)$ and exactly two out of the three last rows of $B^*(i)$ are also represented in the set $eq(t)$. If we delete one of the last three rows of $B^*(i)$, the resulting square matrix will remain non-singular. Note that the column index sets $J(i)$ are pairwise disjoint and that their union equals the index set $J^* = \{n+1, \ldots, n+3m\}$. Since $B(-, J^*) = B^2$ holds, we easily see that the restriction $B^2(\{1, \ldots, 4m\}, -)$ is equal, modulo a suitable row permutation, to the block matrix

$$B^2_{1+2} \quad = \quad \begin{pmatrix} B^*(1) & O & O \\ O & \ddots & O \\ O & O & B^*(m) \end{pmatrix}.$$

The restriction $B^2(eq(t), -)$ deletes from B^2_{1+2} one of the last three rows of each block corresponding to $B^*(i)$. The matrix B^2_{1+2} is non-singular, what implies that the restriction $B^2(eq(t), -)$ is also non-singular, since $B^*(i)$ with one row deleted remains non-singular. Finally, the block B_3 contributes

$\mathit{weight}(v)$ rows to $\mathit{eq}(t)$. Hence, the set $\mathit{eq}(t)$ contains $3m + \mathit{weight}(v)$ row indices and the equality $\mathit{weight}(t) = 3m + \mathit{weight}(v)$ holds. This means that $B(\mathit{eq}(t), \mathit{one}(t))$ is a square matrix. Note that $B(\mathit{eq}(t), \mathit{one}(t))$ is the concatenation of the matrices $B(\mathit{eq}(t), \mathit{one}(v))$ and $B(\mathit{eq}(t), \mathit{one}(s))$, since $t = vs$. Because s is the all-one vector, the matrix $B(\mathit{eq}(t), \mathit{one}(s))$ is equal to $B^2(\mathit{eq}(t), -)$. Notice that $B(\mathit{eq}(t) \cap \{4m+1, \ldots, 4m+n\}, \mathit{one}(v))$ (i.e. the restriction of $B^1(\mathit{eq}(t), \mathit{one}(v))$ to rows of B_3^1) is once more an identity matrix, what makes the block $B^1(\mathit{eq}(t), \mathit{one}(v)) = B(\mathit{eq}(t), \mathit{one}(v))$ non-singular. Finally, the block B_3^2 is an all-zero matrix, therefore the concatenation of matrices $B(\mathit{eq}(t), \mathit{one}(v)) B(\mathit{eq}(t), \mathit{one}(s)) = B(\mathit{eq}(t), \mathit{one}(t))$ is non-singular, what means that its columns are linearly independent. According to Proposition 3.1, the extension $\bar{t}$ is a minimal solution of S, hence s can be extended to a minimal solution of the system S.

Conversely, suppose that s can be extended to a minimal solution of S. Then there exists a partial assignment v to the variables x, forming with s the concatenation $t = vs$, such that $\bar{t}$ is minimal and $\mathit{weight}(t) = 3m + \mathit{weight}(v)$ holds. Note that independently from the choice of the values $v(x_i^1)$, $v(x_i^2)$, and $v(x_i^3)$, at most two of the polynomials $x_i^1 + x_i^2$, and $x_i^2 + x_i^3$, and $x_i^3 + x_i^1$ evaluate to 1. Hence, at most $2m$ rows of B_2 are evaluated to 0 by the assignment t.

Let us analyze the row indices of B that belong to $\mathit{eq}(t)$. The block B_2 contributes always at most $2m$ elements and the block B_3 contributes exactly $\mathit{weight}(v)$ elements to $\mathit{eq}(t)$. Suppose that not all indices of B_1 belong to $\mathit{eq}(t)$. In this case, the block B_1 contributes at most $m-1$ elements to $\mathit{eq}(t)$. This implies that the cardinality of the set $\mathit{eq}(t)$ is smaller or equal than $3m - 1 + \mathit{weight}(v)$ and $B(\mathit{eq}(t), \mathit{one}(t))$ is a $(3m - 1 + \mathit{weight}(v)) \times (3m + \mathit{weight}(v))$ matrix. In this case the column rank of the matrix $B(\mathit{eq}(t), \mathit{one}(t))$ is smaller than $3m + \mathit{weight}(v)$, i.e., the columns are linearly dependent. Following Proposition 3.1, the extension $\bar{t}$ cannot be minimal. Hence, all m row indices of B_1 must belong to $\mathit{eq}(t)$.

Since all m rows of B_1 belong to $\mathit{eq}(t)$ and $s(y_j) = 1$ holds for each j, the structure of B_1^1, encoding the clauses $c_i = x_i^1 \vee x_i^2 \vee x_i^3$ of φ, implies that the equality

$$t(x_i^1) + t(x_i^2) + t(x_i^3) = v(x_i^1) + v(x_i^2) + v(x_i^3) = 1$$

holds over $\mathbb{Z}_2$ for each i. There are two cases to analyze: (1) either $v(x_i^1) = v(x_i^2) = v(x_i^3) = 1$ or (2) exactly one of the values $v(x_i^1)$, $v(x_i^2)$, $v(x_i^3)$ is equal to 1 and the two others are equal to 0. Suppose that there exists an i such that Case 1 is satisfied. Then the maximal number of row indices in $\mathit{eq}(t)$ contributed by B_2 is $2(m-1)$. This is because the equalities $v(x_i^1) + v(x_i^2) = v(x_i^2) + v(x_i^3) = v(x_i^3) + v(x_i^1) = 0$ hold over $\mathbb{Z}_2$. The cardinality of $\mathit{eq}(t)$ is then bounded by $3m - 2 + \mathit{weight}(v)$, what implies once more that the columns of $B(\mathit{eq}(t), \mathit{one}(t))$ are linearly dependent and this leads to the same contradiction, implying that the extension $\bar{t}$ is not minimal, as in the previous paragraph. Case 2 presents a valid 1-in-3 assignment for the formula φ. □

The previous theorem allows us to prove the complexity of the inference problem for affine formulas.

Theorem 4.2 *The problem* AFFINF *is* coNP*-complete.*

Proof: The problem AFFINF is the dual of the problem MINEXT. Note that, given a formula φ and a clause $c = \neg u_1 \vee \cdots \vee \neg u_k$, the condition $\varphi \models_{\min} \neg u_1 \vee \cdots \vee \neg u_k$ holds if and only if there is no minimal model m of φ that satisfies $m(u_1) = \cdots = m(u_k) = 1$. The latter is true if and only if the partial assignment s with $s(u_1) = \cdots = s(u_k) = 1$ cannot be extended to a minimal model of φ, or equivalently, to a minimal solution of the affine system S corresponding to φ. □

5 Decompositions and Polynomial-time Decidable Cases

Eiter and Gottlob proved in [EG93] that the inference problem $\varphi \models_{\min} c$ for propositional circumscription remains Π_2P-complete even if the clause c consists of a single negative literal $\neg u$. However, it is not guaranteed that the complexity remains the same for one-literal clauses c for the usual subclasses of propositional formulas. Concerning the considered inference problem, Cadoli and Lenzerini proved in [CL94] that for dual Horn formulas it remains coNP-complete but for Krom formulas it becomes polynomial-time decidable for a clause c consisting of a single negative literal. It is a natural question to ask what happens in the case of affine formulas in the presence of a single literal. In the rest of the section we will focus on the restrictions AFFINF$_1$ and MINEXT$_1$ of the respective problems AFFINF and MINEXT to a single negative literal clause $c = \neg u$.

To be able to investigate the complexity of MINEXT$_1$ and AFFINF$_1$, we need to define a neighborhood and a congruence closure on the columns.

Definition 5.1 Let B be a $k \times n$ matrix over $\mathbb{Z}_2$ and let $j \in \{1, \dots, n\}$ be a column index. The **p-neighborhood** $N_p(j)$ of the column j in B, for $p = 0, 1, \dots, n$, is defined inductively by

$$\begin{aligned} N_0(j) &= \{j\}, \\ N_{p+1}(j) &= \{m \mid \ (\forall q)[(q \leq p) \rightarrow (m \notin N_q(j))] \ \wedge \\ &\qquad\qquad (\exists \ell)(\exists i)[(\ell \in N_p(j)) \wedge (B(i,\ell) = B(i,m) = 1)]\}. \end{aligned}$$

The **connected component** $CC(j)$ of the column j in B is the union of the p-neighborhoods for all p, i.e., $CC(j) = \bigcup_{p=0}^{n} N_p(j)$.

Speaking in terms of hypergraphs and matroids, where B is interpreted as the vertex-hyperedge incidence matrix, the p-neighborhood $N_p(j)$ is the set of vertices reachable from the vertex j by a path of length p. The vertex ℓ belongs to $N_p(j)$ if and only if the shortest path from j to ℓ in B has the length p. The connected component $CC(j)$ is the set of all reachable vertices from j.

Example 5.2 Consider the following following affine system S: $(I \ B)(xy) = b$, where I, B and b are represented by the successive blocks of the following matrix.

$$(I \mid B \mid b) \quad = \quad \left(\begin{array}{cccccc|ccccc|c} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \boxed{1} & \boxed{1} & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & \boxed{1} & 0 & \boxed{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \boxed{1} & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{array}\right)$$

Take $j = 7$ and compute the p-neighborhood from vertex 7 in the matrix B for each $p = 0, 1, \dots, 6$. We obtain $N_0(7) = \{7\}$, $N_1(7) = \{8, 9\}$, $N_2(7) = \{10\}$, $N_3(7) = \{11\}$, and $N_4(7) = N_5(7) = N_6(7) = \emptyset$. The connected component of the vertex 7 is $CC(7) = \{7, 8, 9, 10, 11\}$.

When computing the connected component for all columns of a given matrix B, we may get two or more disjoint sets of vertices. In this case we say that the matrix B is *decomposable*. The following lemma shows that we can compute the problems MINEXT and AFFINF by connected components without increasing the complexity.

Lemma 5.3 *Let S: $(I\ B)(xy) = b$ be an affine system over $\mathbb{Z}_2$. Suppose that the matrix B can be decomposed, up to a permutation of rows and columns, into the components*

$$\begin{pmatrix} B_1 & O \\ O & B_2 \end{pmatrix}$$

where B_1 is a $k_1 \times n_1$ matrix and B_2 is a $k_2 \times n_2$ matrix. Let b_1 and b_2 be two vectors of respective size n_1 and n_2, such that $b = b_1 b_2$. Then the set of minimal solutions of S is equal, up to a permutation, to the Cartesian product $M_1 \times M_2$ of the sets of minimal solutions M_1 and M_2 of the systems S_1: $(I\ B_1)(x'y') = b_1$ and S_2: $(I\ B_2)(x''y'') = b_2$, respectively, where $x = x'x''$ and $y = y'y''$.

Proof: Straightforward, since the set of solutions $Sol(S)$ of the system S is equal up to permutation to the Cartesian product $Sol(S_1) \times Sol(S_2)$ of the solution sets $Sol(S_1)$ and $Sol(S_2)$, respectively, as it is known from linear algebra, and the inclusion $M_1 \times M_2 \subseteq Sol(S_1) \times Sol(S_2)$ holds. □

The proof of the following theorem shows that finding a minimal extension $\bar{s}$ of a Boolean vector s with $weight(s) = 1$ can be done by finding a shortest path in a connected component of the matrix B from a given column to an inhomogeneous equation in the system S.

Theorem 5.4 *The problems* MINEXT$_1$ *and* AFFINF$_1$ *are decidable in polynomial time.*

Proof: Suppose without loss of generality that S is a $k \times n$ system of the form S: $(I\ B)(xy) = b$ and that the variable assigned by s is y_1. This can be achieved through a suitable permutation of rows and columns. We also suppose that the matrix B is indecomposable. Otherwise, we could apply the method described in this proof to one of the subsystems S_1 or S_2 separately, following Lemma 5.3. Since B is indecomposable, the connected component of the first column is $CC(1) = \{1, \ldots, n\}$, i.e., there are no unreachable columns. The following condition can be established for extensions of weight 1 vectors to minimal solutions.

There exists a minimal solution $\bar{s}$ with $\bar{s}(y_1) = 1$ if and only if $b \neq \vec{0}$.

If $b = \vec{0}$ then the system S is homogeneous and the all-zero assignment for xy is the unique minimal solution of S, what contradicts the existence of a minimal solution $\bar{s}$ with $\bar{s}(y_1) = 1$.

Conversely, suppose that $b \neq \vec{0}$. We construct a partial assignment s for the variables y with $s(y_1) = 1$, such that $\bar{s}$ is minimal. We must find the first inhomogeneous equation reachable from y_1. Since $b \neq \vec{0}$, there exists a shortest path with $p + 1$ indices $j_0 = 1$, j_1, …, j_p, such that (1) for all q, if $q \leq p$ then $j_q \in N_q(1)$; (2) for all $q < p$, for each row i the value $B(i, j_q) = 1$ implies $b_i = 0$, i.e., every variable y_{j_q} for $q < p$ occurs in homogeneous equations only; (3) there exists a row i, such that $B(i, j_p) = b_i = 1$, i.e., the variable y_{j_p} occurs in an inhomogeneous equation. Define the partial assignment s for the variables y by $s(y_{j_q}) = 1$ for each $q \leq p$ and set $s(y_j) = 0$ otherwise. We prove that $\bar{s}$ is a minimal solution of S.

For each $q < p$ there exists a row i_q, such that $B(i_q, j_q) = B(i_q, j_{q+1}) = 1$ holds. In other words, there exists an equation containing both variables y_{j_q} and $y_{j_{q+1}}$. Denote by ind the set of these rows i_q. Following from definition of the connected component, for each row $i \in ind$ there exists a $q < p$, such that $B(i, j_q) = B(i, j_{q+1}) = 1$ holds and for all q', such that $q' \neq q$ and $q' \neq q + 1$, we have $B(i, j_q) = 0$. There exists a row i_p, such that $B(i_p, j_p) = b_{i_p} = 1$ holds and for each $q < p$ we have $B(i_p, j_q) = 0$. In other words, the row index i_p points to an inhomogeneous equation reachable from y_1 in p steps.

Let us examine the value of the variables x in the extension $\bar{s}$. Following from the above facts, it is clear that for each row $i \in ind$ we have $\bar{s}(x_i) = 0$. This follows from the fact that p is minimal and that for each row $i \in \mathit{ind}$ there are exactly two columns j_q and j_{q+1} that verify the condition $B(i, j_q) = B(i, j_{q+1}) = 1$. Moreover, we have $\bar{s}(x_{i_p}) = 0$ also for the row i_p. This follows from the fact that $B(i_p, j_p) = b_{i_p} = 1$.

Suppose now that $\bar{s}$ is not minimal. Then there exists a vector t such that $\bar{t}$ is pointwise smaller than $\bar{s}$, therefore also t must be pointwise smaller that s. This means that there exists a variable from y that is evaluated to 1 by s and to 0 by t. By induction on q we prove that $t(y_{j_q}) = 0$ holds for all $q \leq p$. Let q be such that $s(y_{j_q}) = 1$ and $t(y_{j_q}) = 0$. Then for each row $i \in \mathit{ind}$ we have $\bar{t}(x_i) \leq \bar{s}(x_i) = 0$. Hence we have $t(y_{j_q}) = 0$ and $\bar{s}(x_{i_q}) = 0$. This forces $\bar{t}(x_{i_q}) = 0$, what implies $t(y_{j_{q+1}}) = 0$. Therefore we have $t(y_{j_p}) = 0$. Since $b_{i_p} = 1$ holds, this implies the relation $1 = \bar{t}(x_{i_p}) > \bar{s}(x_{i_p}) = 0$ that contradicts the fact that $\bar{t}$ is smaller than $\bar{s}$. Therefore the extension $\bar{s}$ must be minimal.

The connected component of a column can be computed in polynomial times, therefore both problems MINEXT$_1$ and AFFINF$_1$ are polynomial-time decidable. □

Example 5.5 (Example 5.2 continued) Start with the column $j_0 = 7$ and compute a shortest path reaching an inhomogeneous equation. There is a shortest path from the column 7 with the indices $j_0 = 7$, $j_1 = 9$, $j_2 = 10$, reaching the inhomogeneous row 5. The corresponding rows in *ind* are $i_0 = 4$, $i_1 = 1$, and $i_2 = 5$. The path $B(4,7) \to B(4,9) \to B(1,9) \to B(1,10) \to B(5,10)$ is indicated in the matrix by boxed values. Hence, we computed the partial assignment $s = (1,0,1,1,0)$ for the variables y and the extension $\bar{s} = (0,1,0,0,0,1,1,0,1,1,0)$ is a minimal solution of the system S.

6 Conclusion

formula φ	clause c		literal c	
CNF	Π_2P-complete	[EG93]	Π_2P-complete	[EG93]
Horn	in P		in P	
dual Horn	coNP-complete	[CL94]	coNP-complete	[CL94]
Krom	coNP-complete	[CL94]	in P	[CL94]
affine	coNP-complete	[Theorem 4.2]	in P	[Theorem 5.4]

Figure 2: Complexity of the inference problem of propositional circumscription

We proved that the inference problem of propositional circumscription for affine formulas is coNP-complete. It also shows that reasoning under the extended closed world assumption is intractable for affine formulas. In fact, the exact complexity of affine inference was implicitly an open problem since the beginning of the 1990s when several researchers started to investigate the propositional circumscription from algorithmic point of view. We also proved that the inference problem for affine formulas becomes polynomial-time decidable when only a single literal has to be inferred. The complexity classification of the inference problem of propositional circumscription for the usual classes of formulas is presented in Figure 2. Note that several complexity results mentioned in the survey [Bar98] can be obtained by a reduction from our intractability result.

References

[AB98] A. Ashikhmin and A. Barg. Minimal vectors in linear codes. *IEEE Transactions on Information Theory*, IT-44(5):2010–2018, 1998.

[Bar98] A. Barg. Complexity issues in coding theory. In V. S. Pless and W. C. Huffman, editors, *Handbook of Coding Theory*, volume 1, chapter 7, pages 649–754. Elsevier Science, Amsterdam, 1998.

[BMvT78] E. R. Berlekamp, R. J. McEliece, and H. C. A. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, IT-24(3):384–386, 1978.

[Cad92] M. Cadoli. The complexity of model checking for circumscriptive formulae. *Information Processing Letters*, 44(3):113–118, 1992.

[CL94] M. Cadoli and M. Lenzerini. The complexity of propositional closed world reasoning and circumscription. *Journal of Computer and System Science*, 48(2):255–310, 1994.

[EG93] T. Eiter and G. Gottlob. Propositional circumscription and extended closed-world reasoning are Π_2^p-complete. *Theoretical Computer Science*, 114(2):231–245, 1993.

[EG95] T. Eiter and G. Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278–1304, 1995.

[GPP89] M. Gelfond, H. Przymusinska, and T. C. Przymusinski. On the relationship between circumscription and negation as failure. *Artificial Intelligence*, 38(1):75–94, 1989.

[KK01a] L. M. Kirousis and P. G. Kolaitis. The complexity of minimal satisfiability problems. In A. Ferreira and H. Reichel, editors, *Proceedings 18th Symposium on Theoretical Aspects of Computer Science (STACS 2001), Dresden (Germany)*, volume 2010 of *Lecture Notes in Computer Science*, pages 407–418. Springer-Verlag, February 2001.

[KK01b] L. M. Kirousis and P. G. Kolaitis. A dichotomy in the complexity of propositional circumscription. In *Proceedings 16th IEEE Symposium on Logic in Computer Science (LICS 2001), Boston (Massachusetts, USA)*, pages 71–80. IEEE Computer Society, June 2001.

[KSS00] D. J. Kavvadias, M. Sideri, and E. C. Stavropoulos. Generating all maximal models of a boolean expression. *Information Processing Letters*, 74(3-4):157–162, 2000.

[McC80] J. McCarthy. Circumscription — A form of non-monotonic reasoning. *Artificial Intelligence*, 13(1-2):27–39, 1980.

[Sha79] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

Algorithms and Complexity of Model Representations

Reinhard Pichler
Technische Universität Wien
reini@logic.at

Abstract

Over the past decade, automated model building has evolved as an important subdiscipline of automated deduction. A crucial issue for constructing models on the computer is the selection of an appropriate formalism for representing models. In particular, (efficient) algorithms for evaluating clauses in a model thus constructed are required. Moreover, also the so-called model equivalence problem has received great interest.

In this survey, we recall three formalisms for representing models, namely atomic representations of Herbrand models ([9]), constrained atoms ([3, 4]), and ground atoms with ground equations ([10]). Specifically, we shall recall recent results on algorithms and complexity of the aforementioned decision problems for these model representations.

1 Introduction

Models play an important role in automated theorem proving. Their applicability is basically twofold:

- First, rather than just proving that some input formula is *not* a theorem, it would be desirable for a theorem prover to provide some insight as to *why* a given formula is not a theorem. To this end, the theorem prover tries to construct a countermodel rather than just giving the answer "NO".
- The second application of models arises from the idea of guiding the proof search by providing some additional knowledge on the domain from which the input formula is taken. This knowledge can be represented in the form of a model, which may then be used e.g. in semantic resolution.

Consequently, over the past few years, automated model building has evolved as an important discipline within the field of automated deduction (cf. e.g. [4], [31], [29, 16], [5], [34], [2], [9, 10], [24, 25], [1]).

In order to actually deal with models on a computer, we need an appropriate formalism for representing models. On the one hand, this formalism should provide "reasonable" expressive power (e.g., it should be possible to represent also infinite models). On the other hand, such a formalism should admit efficient algorithms for the following two decision problems:

- The CLAUSE EVALUATION problem: Given a representation M of a model and a clause C, does C evaluate to "true" in this model?

- The MODEL EQUIVALENCE problem: Given two representations M and N of models, do M and N represent the same model?

The importance of the first decision problem is obvious. Suppose that we want to use a model as an input to an automated theorem prover based on semantic resolution. Then an efficient CLAUSE EVALUATION algorithm is clearly indispensable. The significance of the second problem comes from the fact that formalisms presented in the model building literature for representing models usually allow for many different ways of representing the same model. However, when we actually want to compute the truth value of arbitrary clauses in such a model, then the efficiency of this computation depends to a large extend on the specific representation rather than just the model thus represented. It is therefore important to look for transformations of the model representation constructed in the first step into an equivalent one with "better" computational properties. But then we have to make sure, of course, that the model representation resulting from such a transformation is equivalent to the original one.

In this survey, we recall three such formalisms for representing models, namely atomic representations of Herbrand models (in Section 3), constrained atoms (in Section 4), and ground atoms with ground equations (in Section 5). Specifically, we shall recall recent results on algorithms and complexity of the aforementioned decision problems for these model representations. Moreover, in Section 2, we recall some basic notions and in Section 6, we give a conclusion.

2 Preliminaries

The reader is assumed to be familiar with the basic concepts of computational logic. In this section we only recall the most important concepts for our further discussion. Let Σ denote a finite set of predicate symbols and function symbols, each with some arity $k \geq 0$. The set H of all ground terms over Σ is called the *Herbrand universe* over Σ. The set HB of all ground atoms over some signature Σ is called the *Herbrand base*. In the sequel, we usually consider the signature Σ as arbitrary but fixed when talking about some Herbrand universe H or a Herbrand base HB. We call H *non-trivial*, if it contains at least 2 elements.

A *literal* is either an atom A or a negated atom $\neg A$. *Clauses* are written as $C = L_1 \vee \ldots \vee L_n$, where the L_i's are literals. Recall that clauses are basically a short-hand notation for closed first-order formulae where all variables are universally quantified. A *Herbrand interpretation* of a clause (or a set of clauses) is an interpretation which interprets all ground terms "by themselves", so to speak. Hence, a Herbrand interpretation is given by the interpretation of the predicate symbols only. In particular, such an interpretation is uniquely determined by a subset of the Herbrand base, namely by those ground atoms over the given signature Σ which evaluate to "true".

In general, a *model* is an *interpretation* which validates a certain formula (or, analogously, a clause set) As long as one is concerned with the actual model construction, it is clear which formula is validated by the interpretation thus constructed. However, when one starts to work with such an interpretation in a different context (e.g. as input to a theorem prover based on semantic resolution), the connection between the interpretation and the formula which is validated by this interpretation is no longer obvious. In fact, it is a bit inaccurate to talk about "models" rather than "interpretations", when it is not clear, which formula is actually validated by a given interpretation. However, this kind of inaccuracy is very common in the model building literature. We shall, therefore, also refer to "interpretations" as "models" without having a particular formula in mind which is validated by such an interpretation.

3 Atomic Representations of Herbrand Models

In [9], Atomic Representations of Herbrand Models (= ARMs, for short) were introduced as finite sets $\mathcal{A} = \{A_1, \ldots, A_n\}$ of atoms, s.t. a ground atom evaluates to "true" in the Herbrand model represented by $\mathcal{A}$, iff it is a ground instance of some atom $A_i \in \mathcal{A}$.

Example 3.1 Let $\mathcal{A} = \{P(f(x), a), P(a, a), Q(x, x), Q(a, f(x))\}$ be an ARM and let the signature $\Sigma = \{P, Q, a, b, f\}$. Then the clause $C_1 = P(x, a) \vee \neg Q(x, f(a))$ evaluates to "true" in the model defined by $\mathcal{A}$. In order to see this, we have to verify, that all H-ground instances of C_1 evaluate to "true": To this end, we distinguish three kinds of possible values that the variable x can take and show that in each case at least one literal in C_1 evaluates to "true":

- For $x = a$ the literal $P(a, a)$ evaluates to "true".
- Now let $x = b$. Then $Q(b, f(a))$ is not an instance of any atom in $\mathcal{A}$. Hence, $Q(b, f(a))$ evaluates to "false" and, thus, $\neg Q(b, f(a))$ evaluates to "true".
- Finally, let x be a term with leading symbol f, i.e., $x = f(t)$ for some term $t \in H$. Then $P(f(t), a)$ is an instance of $P(f(x), a) \in \mathcal{A}$ and, therefore, $P(f(t), a)$ evaluates to "true".

On the other hand, the clause $C_2 = P(x, y) \vee \neg Q(a, y)$ evaluates to "false". This can be seen by showing that there exists at least one H-ground instance of C_2 that evaluates to "false". In fact, $P(a, f(a)) \vee \neg Q(a, f(a))$ is such a ground instance. It is easy to verify that both literals evaluate to "false". $\square$

The key to the decision problems is yet another problem, which was called the ATOMIC H-SUBSUMPTION problem in [9], i.e.: Given an atom set $\mathcal{A} = \{A_1, \ldots, A_n\}$ and an atom B over some Herbrand universe H, is every H-ground instance of B an instance of some atom $A_i \in \mathcal{A}$? If this is the case, then we write $\mathcal{A} \leq_{sH} B$. In [9], the MODEL EQUIVALENCE problem and the CLAUSE EVALUATION problem are reduced to the (ATOMIC) H-SUBSUMPTION problem as follows:

Lemma 3.2 *Let $\mathcal{A} = \{A_1, \ldots, A_n\}$ and $\mathcal{B} = \{B_1, \ldots, B_m\}$ be ARMs w.r.t. some Herbrand universe H. Then $\mathcal{A}$ and $\mathcal{B}$ are equivalent, iff $\{A_1, \ldots, A_n\} \leq_{sH} B_j$ for every $j \in \{1, \ldots, m\}$ and $\{B_1, \ldots, B_m\} \leq_{sH} A_i$ for every $i \in \{1, \ldots, n\}$.*

In order to reduce also the CLAUSE EVALUATION problem to the H-SUBSUMPTION problem, we first have to recall that H-subsumption is not necessarily restricted to atoms, i.e.: For a clause set $\mathcal{C}$ and a clause D, we say that $\mathcal{C}$ *H-subsumes* D (written as $\mathcal{C} \leq_{sH} D$), iff every H-ground instance of D is subsumed by some clause $C \in \mathcal{C}$. More generally, if $\mathcal{D}$ is a clause set, we say that $\mathcal{C}$ *H-subsumes* $\mathcal{D}$ (written as $\mathcal{C} \leq_{sH} \mathcal{D}$), iff $\mathcal{C} \leq_{sH} D$ holds for every clause $D \in \mathcal{D}$. Then we have:

Lemma 3.3 *Let $\mathcal{A} = \{A_1, \ldots, A_n\}$ be an ARM w.r.t. some Herbrand universe H and let $C = L_1 \vee \ldots \vee L_l \vee \neg M_1 \vee \ldots \vee \neg M_m$ be a clause over H. Then we distinguish two cases:*

- Case 1*: $m = 0$, i.e.: C is a positive clause. Then C evaluates to "true" $\Leftrightarrow$ $\mathcal{A} \leq_{sH} C$.*
- Case 2*: $m > 0$, i.e.: C contains at least one negative literal. Now let $\rho_h(\mathcal{A} \cup \{C\})$ denote the set of all hyperresolvents that are derivable from $\mathcal{A} \cup \{C\}$. Then C evaluates to "true" $\Leftrightarrow$ $\mathcal{A} \leq_{sH} \rho_h(\mathcal{A} \cup \{C\})$.*

Equivalent problems to the H-SUBSUMPTION problem have been studied in many areas of computer science, such as in machine learning (cf. [21]) in *logic programming* (cf. [12], [20]), in functional programming (cf. [20]), etc. Consequently, many different approaches for deciding the H-SUBSUMPTION problem (or equivalent problems) have been presented in the literature. We only recall the algorithm from [9] here, which is based on the following property: *If $\mathcal{C}$ and $\mathcal{D}$ are sets of clauses, s.t. the minimum depth of variable occurrences in $\mathcal{D}$ is greater than the depth of $\mathcal{C}$, then H-subsumption and ordinary subsumption coincide.* Hence, the H-SUBSUMPTION problem $\mathcal{A} \leq_{sH} C$ for an atom set $\mathcal{A}$ and a clause C can be decided as follows: First, C is transformed into an equivalent clause set $\mathcal{C}'$ by partial saturation, s.t. the minimum depth of variable occurrences in $\mathcal{C}'$ is greater than the depth of $\mathcal{A}$. Then an ordinary subsumption test $\mathcal{A} \leq_s C'$ is applied to every clause $C' \in \mathcal{C}'$.

As to the complexity of the decision problems studied here, it is convenient to consider also the TOTAL COVER problem, which is defined as follows: Given an atom set $\mathcal{A} = \{P(\vec{t_1}), \ldots, P(\vec{t_n})\}$ over some Herbrand universe H, is every H-ground atom $P(\vec{s})$ an instance of some $P(\vec{t_i}) \in \mathcal{A}$?

It is easy to show the following relations between these problems (cf. [13, 14]): The TOTAL COVER problem can be reduced to the MODEL EQUIVALENCE problem, which in turn can be reduced to the MODEL EQUIVALENCE. Finally, MODEL EQUIVALENCE can be reduced to the CLAUSE EVALUATION problem. All these reductions are possible in polynomial time. In other words, TOTAL COVER is the "easiest" and CLAUSE EVALUATION is the "hardest" of these four problem.

The inherent complexity of the TOTAL COVER problem (or equivalent problems) has been investigated by several authors independently, who proved its coNP-completeness (cf. [18], [19], [17]). Together with the coNP-membership of clause evaluation (cf. [13, 14]), we get the following result:

Theorem 3.4 *The following decision problems are coNP-complete over any non-trivial Herbrand universe:* TOTAL COVER, MODEL EQUIVALENCE, ATOMIC H-SUBSUMPTION, *and* CLAUSE EVALUATION.

4 Constrained Atoms

ARMs have a somehow restricted expressive power. In particular, they are not closed under complement, i.e.: Let $\mathcal{A} = \{A_1, \ldots, A_n\}$ be a set of atoms. Then, in general, the set of ground atoms that are *not* instances of the atoms $A_i \in \mathcal{A}$ cannot be expressed in terms of an ARM itself. In [3, 4], the expressive power of ordinary clauses is increased by adding *equational constraints.* A constrained clause (= c-clause, for short) over some Herbrand universe H is a construct of the form $[\![c : \mathcal{P}]\!]$, where c is a clause and $\mathcal{P}$ is an equational formula over H. An H-ground clause $c\sigma$ is an instance of $[\![c : \mathcal{P}]\!]$, iff σ is a solution of $\mathcal{P}$. Standard clauses can be considered as a special case of constrained clauses with the trivially true constraint $\top$.

Example 4.1 Let $\mathcal{A} = \{P(x, x)\}$ be an ARM. It can be shown (see [21]), that the complement of $\mathcal{A}$ does not have a representation by an ARM. In contrast, such a representation is easy by using equational constraints, namely: $\mathcal{A}' = \{[\![P(u, v) : u \neq v]\!]\}$. Note that $P(x, x)$ can also be considered as a constrained atom, namely $[\![P(x, x) : \top]\!]$. □

Actually, in [3, 4], models are defined in a slightly different way. Rather than just specifying the set of ground atoms that evaluate to "true" (and requiring that all other ground atoms evaluate to "false"), Caferra et al. introduced so-called *partial interpretations definable by equational formulae*

(= peq-interpretations, for short). Such a peq-interpretation is given through a finite set $\mathcal{L} = \{[\![l_1 : \mathcal{P}_1]\!], \ldots, [\![l_n, \mathcal{P}_n]\!]\}$ of constrained literals (the l_i's are either atoms or negated atoms). A ground atom A evaluates to "true" in the interpretation defined by $\mathcal{L}$, iff A is an instance of some (positive) c-literal in $\mathcal{L}$. On the other hand, A evaluates to "false", if $\neg A$ is an instance of some (negative) c-literal in $\mathcal{L}$. Otherwise, the truth value of A is undefined. Of course, one has to make sure, that there exists no atom A, s.t. both A and $\neg A$ are instances of elements in $\mathcal{L}$.

The definition of the truth value of a negated ground atom $\neg A$ is obvious, i.e.: $\neg A$ is "true", iff A is "false". Likewise, $\neg A$ is "false", iff A is "true". Finally, $\neg A$ is "undefined", iff A is "undefined". The truth value of arbitrary c-clauses in a peq-interpretation is defined as follows: Let $C = M_1 \vee \ldots \vee M_k$ denote a ground clause. Then the truth value $I(C)$ in the peq-interpretation I represented by $\mathcal{L}$ is defined as follows:

$$I(C) = \begin{cases} \text{“true”} & \text{if } \exists i\text{: } I(M_i) = \text{“true”} \\ \text{“false”} & \text{if } \forall i\text{: } I(M_i) = \text{“false”} \\ \text{“undefined”} & \text{otherwise} \end{cases}$$

Now let $[\![c : \mathcal{Q}]\!]$ be an arbitrary c-clause. Then the truth value $I([\![c : \mathcal{Q}]\!])$ is defined as follows:

$$I([\![c : \mathcal{Q}]\!]) = \begin{cases} \text{“true”} & \text{if } \forall\ H\text{-ground instances } c\sigma \text{ of } [\![c : \mathcal{Q}]\!]\text{: } I(c\sigma) = \text{“true”} \\ \text{“false”} & \text{if } \exists\ H\text{-ground instance } c\sigma \text{ of } [\![c : \mathcal{Q}]\!]\text{: } I(c\sigma) = \text{“false”} \\ \text{“undefined”} & \text{otherwise} \end{cases}$$

Example 4.2 Let the peq-interpretation I be given through the set $\mathcal{L} = \{[\![P(x, f(y)) : x \neq y]\!], [\![\neg P(x, y) : x \neq a \wedge (\forall z) y \neq f(z)]\!]\}$ of c-literals and let the signature $\Sigma = \{P, a, b, f\}$. It is easy to check that $\{[\![P(x, f(y)) : x \neq y]\!]$ and $[\![\neg P(x, y) : x \neq a \wedge (\forall z) y \neq f(z)]\!]$ have no H-ground instances in common. Hence, the peq-interpretation I is well-defined.

Now consider the clause $C = P(x, x) \vee \neg P(a, f(x))$ or, equivalently, the c-clause $C' = [\![P(x, x) \vee \neg P(a, f(x)) : \top]\!]$. Then these clauses evaluate to "false" in I, since the H-ground instance $P(b, b) \vee \neg P(a, f(b))$ does. Indeed, $P(b, b)$ evaluates to "false", since $\neg P(b, b)$ is an instance $[\![\neg P(x, y) : x \neq a \wedge (\forall z) y \neq f(z)]\!]$. Likewise, the second literal $\neg P(a, f(b))$ evaluates to "false", since $P(a, f(b))$ is an instance of $[\![P(x, f(y)) : x \neq y]\!]$. □

The above condition for an arbitrary c-clause to evaluate to "true" can be expressed as the validity of an equational formula in the following way (cf. [2]):

Definition 4.3 *Let the peq-interpretation I over H be given through the set $\mathcal{L} = \{[\![L_1(\vec{s}_1) : \mathcal{P}_1]\!], \ldots, [\![L_n(\vec{s}_n) : \mathcal{P}_n]\!]\}$ of c-literals and let $C = [\![M_1(\vec{t}_1) \vee \ldots \vee M_k(\vec{t}_k) : \mathcal{Q}]\!]$ be a c-clause over H, where the L_i's and M_j's denote literal symbols (i.e., either unnegated or negated predicate symbols). Moreover, let $\vec{y}_i = Var([\![L_i(\vec{s}_i) : \mathcal{P}_i]\!])$ and suppose that $\vec{y}_1, \ldots, \vec{y}_n$, $Var(C)$ are pairwise disjoint. Then the equational formula $\mathcal{F}_I(C)$ is defined as follows:*

$$\mathcal{F}_I(C) \equiv \mathcal{Q} \wedge \bigvee_{M_j = L_i} (\exists \vec{y}_i)[\mathcal{P}_i \wedge \vec{s}_i = \vec{t}_j]$$

Then the following condition holds:

Lemma 4.4 *Let I, $\mathcal{L}$, C, and $\mathcal{F}_I(C)$ be defined as above. Moreover, let $C\sigma$ be an arbitrary H-ground instance of C. Then the following equivalence holds:*

$$C\sigma \textit{ evaluates to “true” in } I \Leftrightarrow \sigma \textit{ is a solution of } \mathcal{F}_I(C)$$

But then the condition that C evaluates to "true" in I is clearly equivalent to the condition that $\mathcal{F}_I(C)$ and $\mathcal{Q}$ are equivalent. This is the case, iff the equational formula $\mathcal{F}_I(C) \leftrightarrow \mathcal{Q}$ is valid.

Likewise, the MODEL EQUIVALENCE problem can of course be reduced to the validity problem of equational formulae. This can be easily seen by first reducing the MODEL EQUIVALENCE problem to the CLAUSE EVALUATION problem and then applying the problem reduction via the formula $\mathcal{F}_I(C)$:

Lemma 4.5 *Let the peq-interpretations I and J over H be given through the sets of c-literals $\mathcal{L} = \{[\![L_1(\vec{s}_1) : \mathcal{P}_1]\!], \dots, [\![L_l(\vec{s}_l) : \mathcal{P}_l]\!]\}$ and $\mathcal{M} = \{[\![M_1(\vec{t}_1) : \mathcal{Q}_1]\!], \dots, [\![M_n(\vec{t})_m) : \mathcal{Q}_m]\!]\}$, respectively.*

Then the peq-interpretations I and J are equivalent, iff every c-literal $[\![L_i(\vec{s}_i) : \mathcal{P}_i]\!] \in \mathcal{L}$ is "true" in J and every c-literal $[\![M_j(\vec{t}_j) : \mathcal{Q}_j]\!] \in \mathcal{M}$ is "true" in I.

Equational formulae have been studied by various authors (cf. [23], [18], [22], [8], [7], [32], [33], [26]). Several decision methods for the validity problem of equational formulae have been proposed. Unfortunately, they all have a very high computational complexity. But this cannot be helped by the following result from [32]:

Theorem 4.6 *The validity problem of equational formulae over an infinite Herbrand universe (i.e., where the signature contains at least one proper function symbol) is non-elementary recursive.*

Of course, we cannot expect to do better for the CLAUSE EVALUATION problem and the MODEL EQUIVALENCE problem of peq-interpretations. We thus have

Corollary 4.7 *The* CLAUSE EVALUATION *problem and the* MODEL EQUIVALENCE *problem of peq-interpretations over an infinite Herbrand universe is non-elementary recursive.*

Actually, in case of a finite Herbrand universe, the validity problem of equational formulae is "only" PSPACE-complete (cf. [18]). But of course, in the area of automated model building, the infinite case is far more relevant.

5 Ground Atoms with Ground Equations

The representation of Herbrand models via ground atoms with ground equations (= GAE-models) was introduced in [10]. A GAE-model $\mathcal{M}$ is given through a finite set $\mathcal{A} = \{A_1, \dots, A_m\}$ of ground atoms and a finite set $\mathcal{E} = \{s_1 = t_1, \dots, s_n = t_n\}$ of ground equations over the signature Σ. A ground atom B over Σ evaluates to "true" in such a model $\mathcal{M}$, iff B is equal to some atom $A_i \in \mathcal{A}$ in the equational theory defined by $\mathcal{E}$.

Example 5.1 Let $\mathcal{C} = \{P(x) \vee P(f(x)), \neg P(x) \vee \neg P(f(x))\}$ be a clause set over the signature $\Sigma = \{P, f, a\}$. Then $\mathcal{M} = (\mathcal{A}, \mathcal{E})$ with $\mathcal{A} = \{P(a)\}$ and $\mathcal{E} = \{a = f(f(a))\}$ is a model of the clause set $\mathcal{C}$. Note that the set of ground atoms that are "true" in $\mathcal{M}$ is $\{P(a), P(f^2(a)), P(f^4(a)), \dots\}$.

Another model of $\mathcal{C}$ is $\mathcal{M}' = (\mathcal{A}', \mathcal{E}')$ with $\mathcal{A}' = \{P(f(a))\}$ and $\mathcal{E}' = \{a = f(f(a))\} = \mathcal{E}$. The set of ground atoms that evaluate to "true" in $\mathcal{M}'$ is $\{P(f(a)), P(f^3(a)), P(f^5(a)), \dots\}$. □

Note that the models of the clause set $\mathcal{C}$ in the above example can neither be represented by atomic representations nor by constrained atoms. On the other hand, the simple ARM $\mathcal{A} = \{P(x, x)\}$ from Example 4.1 cannot be expressed in terms of a GAE-model. This is due to the fact that GAE-models cannot handle "non-linearities" (i.e., multiply occurring variables).

Algorithms both for the CLAUSE EVALUATION problem and the MODEL EQUIVALENCE problem are based on the computation of a canonical *ground term rewrite system* (= GTRS) equivalent to the ground equations in $\mathcal{E}$. It is well-known that this can be achieved quite efficiently, namely in polynomial time (w.r.t. the size of the ground equation system). The earlier *completion* algorithms for that purpose are in essence *congruence closure* based methods (cf. e.g. [11]) where the original signature is extended by fresh constants in order to name certain congruence classes. Yet, direct polynomial completion without extending signatures is also possible via a more sophisticated approach as demonstrated in [27] (cf. also [30]).

In [10], the following CLAUSE EVALUATION algorithm for GAE-models is proposed: First, note that the evaluation of ground atoms is done via a canonical GTRS, i.e.: Let a GAE-model $\mathcal{M}$ be given through a set of ground atoms $\mathcal{A}$ and a set of ground equations $\mathcal{E}$. Moreover, let $\mathcal{R}$ be a canonical GTRS equivalent to $\mathcal{E}$ and let $\widehat{\mathcal{A}}$ denote the set of normal forms of $\mathcal{A}$ w.r.t. $\mathcal{R}$. Then a ground atom B evaluates to "true" in $\mathcal{M}$, iff $NF_{\mathcal{R}}(B) \in \widehat{\mathcal{A}}$, where $NF_{\mathcal{R}}$ denotes the normal form w.r.t. $\mathcal{R}$. The evaluation of ground clauses $B_1 \vee \ldots \vee B_k \vee \neg B_{k+1} \vee \ldots \vee \neg B_l$ is then also obvious.

Now let $\mathcal{U}$ denote the set of all terms and subterms occurring in $\mathcal{A}$ and $\mathcal{E}$. Moreover, let $\widehat{\mathcal{U}}$ denote the set of normal forms w.r.t. $\mathcal{R}$ of the terms in $\mathcal{U}$. The evaluation of a non-ground clause C is based on the following observations:

1. Any subterm u occurring in B may be replaced by its normal form $NF_{\mathcal{R}}(u)$ without affecting the truth value of B. Hence, suppose that all terms of H can be normalized into $\widehat{\mathcal{U}}$. Then it is not necessary to check all possible ground instances $C\vartheta$ of C, that are obtained by substituting all possible terms in H for the variables in C. Instead, it suffices to consider those ground clauses $C\vartheta$, where the range $rg(\vartheta)$ is restricted to the *finite* set $\widehat{\mathcal{U}}$.

2. Suppose that some subterm u occurring in B does not normalize into $\widehat{\mathcal{U}}$. In this case, the normal form $NF_{\mathcal{R}}(B)$ will definitely lie outside $\widehat{\mathcal{A}}$. In other words, normalization of any subterm in B into the set $\widehat{\mathcal{U}}$ is a necessary condition for B to have the truth value "true". So let t be an arbitrary term in H with $NF_{\mathcal{R}}(t) \notin \widehat{\mathcal{U}}$. Then we may in fact substitute t for any subterm u in B with $NF_{\mathcal{R}}(u) \notin \widehat{\mathcal{U}}$ without changing the truth value of B (which is "false"). But then, it is again not necessary to check all possible ground instances $C\vartheta$ of C, that are obtained by substituting all possible terms in H for the variables in C. Instead, it suffices to consider those ground instances $C\vartheta$ of C with $rg(\vartheta) \subseteq \widehat{\mathcal{U}} \cup \{t\}$.

Thus, the CLAUSE EVALUATION procedure in [10] for an arbitrary clause C in a GAE-model $\mathcal{M}$ consists of two steps:

First, we have to check, whether the whole Herbrand universe H normalizes into $\widehat{\mathcal{U}}$ or not. This is done by the following criterion: There exists a ground term t over Σ with $NF_{\mathcal{R}}(t) \in \widehat{\mathcal{U}}$, iff there exists a term of the form $t' = f(t_1, \ldots, t_k)$ with $f \in \Sigma$ and $\forall i \in \{1, \ldots, k\}$, $t_i \in \widehat{\mathcal{U}}$, s.t. $NF_{\mathcal{R}}(t') \in \widehat{\mathcal{U}}$. The correctness of this equivalence can be easily shown by structural induction.

Depending on the outcome of this procedure, we have to distinguish two cases:

- *Case 1*: If $\forall t \in H$, $NF_{\mathcal{R}}(t) \in \widehat{\mathcal{U}}$ holds, then the truth value of C can be determined via the following equivalence: C evaluates to "true" in $\mathcal{M}$, iff every ground instance $C\vartheta$ does, where ϑ is an arbitrary ground substitution with range $rg(\vartheta) \subseteq \widehat{\mathcal{U}}$.

- *Case 2*: If $\exists t \in H$, s.t. $NF_{\mathcal{R}}(t) \notin \widehat{\mathcal{U}}$ holds, then the truth value of C can be determined via another equivalence, namely: C evaluates to "true" in $\mathcal{M}$, iff every ground instance $C\vartheta$ does, where ϑ is an arbitrary ground substitution with range $rg(\vartheta) \subseteq \widehat{\mathcal{U}} \cup \{t\}$.

The correctness of this algorithm follows easily by the above considerations. Of course, in the worst case, we have to check for exponentially many ground instances $C\vartheta$ of C whether they evaluate to "true". In fact, this cannot be helped by the following complexity result from [15]:

Theorem 5.2 *The* CLAUSE EVALUATION *problem of GAE-models over any non-trivial Herbrand universe is coNP-complete.*

In contrast, the MODEL EQUIVALENCE problem of GAE-models can be decided in polynomial time. In [15], an efficient algorithm for this problem is presented. However, below, we sketch a different approach, namely via finite tree automata (FTA, for short): Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two input GAE-models over some Herbrand universe H. Of course, we can check the equivalence of $\mathcal{M}_1$ and $\mathcal{M}_2$ separately for each predicate symbol of the signature Σ. Hence, we may assume w.l.o.g. that Σ contains exactly one predicate symbol P (with arity $k \geq 1$). In [24, 25], it is shown how a GAE-model $\mathcal{M}_i$ can be transformed into an equivalent term grammar $\mathcal{G}_i$, which can then be further transformed into an equivalent FTA A_i, i.e., $P(t_1, \ldots, t_k)$ evaluates to "true" in $\mathcal{M}_i \Leftrightarrow P(t_1, \ldots, t_k)$ is recognized by the FTA A_i. Let $\mathcal{M}_1$ be given in the form $(\mathcal{P}, \widehat{\mathcal{A}}, \widehat{\mathcal{S}})$, where $\mathcal{P}$ is a canonical GTRS, $\widehat{\mathcal{A}}$ denotes the normal forms (w.r.t. $\mathcal{P}$) of all ground atoms that are "true" in $\mathcal{M}_1$ and $\widehat{\mathcal{S}}$ denotes the set of normal forms of all subterms occurring in $\mathcal{P}$ and $\widehat{\mathcal{A}}$. Then our FTA $\mathcal{A}_1$ is given through the following quadruple (Σ, Q, Q_f, δ):

The signature of H is also taken as the signature of the FTA (recall that we are assuming that P is the only predicate symbol in Σ). $Q = \{q_s \,|\, s \in \widehat{\mathcal{S}}\} \cup \{q_P\}$, i.e., Q contains a state q_s for every normal form $s \in \widehat{\mathcal{S}}$ plus an additional state q_P which will ultimately correspond to the atoms that are "true" in $\mathcal{M}_1$. Moreover, we set $Q_f = \{q_P\}$. Finally, the transition rules δ are defined as follows:

- *Rules depending on the structure of the $s_i \in \widehat{\mathcal{S}}$*: For every constant $a \in \widehat{\mathcal{S}}$, δ contains a transition rule $a \to q_a$. For every functional term $s \in \widehat{\mathcal{S}}$ with $s = f(s_1, \ldots, s_\alpha)$, δ contains a transition rule $f(q_{s_1}, \ldots, q_{s_\alpha}) \to q_s$. Note that $\widehat{\mathcal{S}}$ is closed w.r.t. subterms. Hence, we indeed have $s_i \in \widehat{\mathcal{S}}$ for every i.

- *Rules depending on $\mathcal{P}$*: For every rewrite rule $b \to a$ in $\mathcal{P}$, δ contains a transition rule $b \to q_a$. For every rewrite rule $f(s_1, \ldots, s_\alpha) \to s$ in $\mathcal{P}$ (with $\alpha > 0$), δ contains a transition rule $f(q_{s_1}, \ldots, q_{s_\alpha}) \to q_s$.

- *Rules depending on the atoms in $\widehat{\mathcal{A}}$*: For every atom $P(s_1, \ldots, s_k) \in \widehat{\mathcal{A}}$, δ contains a transition rule $P(q_{s_1}, \ldots, q_{s_\alpha}) \to q_P$.

Now suppose that we have transformed both GAE-models $\mathcal{M}_1$ and $\mathcal{M}_2$ into equivalent FTAs $\mathcal{A}_1$ and $\mathcal{A}_2$. Then we have, in fact, reduced the MODEL EQUIVALENCE problem of GAE-models into the equivalence problem of FTA's. The latter problem can be decided by well-known methods (cf. [6], [28]). We thus have:

Theorem 5.3 *The* MODEL EQUIVALENCE *problem of GAE-models over any non-trivial Herbrand universe can be decided in polynomial time.*

6 Conclusion

In this survey, we have recalled some algorithms and complexity results related to three model representation formalisms, namely: atomic representations of Herbrand models, constrained atoms, and ground atoms with ground equations. However, many more formalisms for representing models can be found in the literature like (various forms of) term schematizations, (linear) atoms with positional constraints, term grammars and finite tree automata, tree automata with brotherhood constraints, etc. A good overview with a detailed comparison of the expressive power of these formalisms is given in (cf. [24, 25]).

Efficient CLAUSE EVALUATION and MODEL EQUIVALENCE algorithms are an important issue, if one wants to work with models after they have been constructed. Nevertheless, in the model building community, no particular emphasis is usually put on such algorithms. Hence, a thorough complexity analysis and the search for reasonably efficient algorithms also for other model representation formalisms is an interesting task for future work in this area.

References

[1] P. Baumgartner, C. Fermüller, N. Peltier, and H. Zhang. Workshop: Model Computation – Principles, Algorithms, Applications. In *Proc. CADE-17*, volume 1831 of *LNAI*, page 513. Springer Verlag, 2000.

[2] R. Caferra and N. Peltier. Decision procedures using model building techniques. In *Proc. CSL'95*, volume 1092 of *LNCS*, pages 130–144. Springer Verlag, 1995.

[3] R. Caferra and N. Zabel. Extending resolution for model construction. In *Proc. JELIA'90*, volume 478 of *LNCS*, pages 153–169. Springer Verlag, 1991.

[4] R. Caferra and N. Zabel. A method for simultanous search for refutations and models by equational constraint solving. *Journal of Symbolic Computation*, 13:613–642, 1992.

[5] H. Chu and D. Plaisted. CLIN-S - a semantically guided first-order theorem prover. *Journal of Automated Reasoning*, 18(2):183–188, 1997.

[6] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. *Tree automata techniques and applications.* 1999. available at: http://www.grappa.univ-lille3.fr/tata.

[7] H. Comon and C. Delor. Equational formulae with membership constraints. *Information and Computation*, 112(2):167–216, 1994.

[8] H. Comon and P. Lescanne. Equational problems and disunification. *Journal of Symbolic Computation*, 7(3/4):371–425, 1989.

[9] C. G. Fermüller and A. Leitsch. Hyperresolution and automated model building. *Journal of Logic and Computation*, 6(2):173–230, 1996.

[10] C. G. Fermüller and A. Leitsch. Decision procedures and model building in equational clause logic. *Logic Journal of the IGPL*, 6(1):17–41, 1998.

[11] Z. Fülöp and S. Vágvölgyi. Ground term rewriting rules for the word problem of ground term equations. *Bulletin of the European Association for Theoretical Computer Science*, 45:186–201, Oct. 1991.

[12] G. Gottlob, S. Marcus, A. Nerode, G. Salzer, and V. S. Subrahmanian. A non-ground realization of the stable and well-founded semantics. *Theoretical Computer Science*, 166:221–262, 1996.

[13] G. Gottlob and R. Pichler. Working with ARMs: Complexity results on atomic representations of Herbrand models. In *Proc. LICS'99*, pages 306–315. IEEE Computer Society, 1999.

[14] G. Gottlob and R. Pichler. Working with ARMs: Complexity results on atomic representations of Herbrand models. *Information and Computation*, 165:183–207, 2001.

[15] B. Gramlich and R. Pichler. Algorithmic aspects of Herbrand models represented by ground atoms with ground equations. In *Proc. CADE-19*, LNAI. Springer Verlag, 2002. to appear.

[16] K. Hodgson and J. Slaney. System description: Scott-5. In *Proc. IJCAR'01*, volume 2083 of *LNCS*, pages 443–447. Springer Verlag, 2001.

[17] D. Kapur, P. Narendran, D. Rosenkrantz, and H. Zhang. Sufficient-completeness, ground-reducibility and their complexity. *Acta Informatica*, 28(4):311–350, 1991.

[18] K. Kunen. Answer sets and negation as failure. In *Proc. ICLP'87*, pages 219–228. MIT Press, 1987.

[19] G. Kuper, K. McAloon, K. Palem, and K. Perry. Efficient parallel algorithms for anti-unification and relative complement. In *Proc. LICS'88*, pages 112–120. IEEE Computer Society, 1988.

[20] J.-L. Lassez, M. Maher, and K. Marriott. Elimination of negation in term algebras. In *Proc. MFCS'91*, number 520 in LNCS, pages 1–16. Springer Verlag, 1991.

[21] J.-L. Lassez and K. Marriott. Explicit representation of terms defined by counter examples. *Journal of Automated Reasoning*, 3(3):301–317, 1987.

[22] M. Maher. Complete axiomatizations of the algebras of finite, rational and infinite trees. In *Proc. LICS'88*, pages 348–357. IEEE Computer Society, 1988.

[23] A. Malćev. On the elementary theories of locally free universal algebras. *Soviet Mathematical Doklady*, 2(3):768–771, 1961.

[24] R. Matzinger. Comparing computational representations of Herbrand models. In *Proc. KGC'97*, number 1289 in LNCS, pages 203–218. Springer Verlag, 1997.

[25] R. Matzinger. *Computational Representations of Models in First-Order Logic.* PhD thesis, Vienna University of Technology, 2000.

[26] R. Pichler. Solving equational problems efficiently. In *Proc. CADE-16*, number 1632 in LNAI, pages 97 – 111. Springer Verlag, 1999.

[27] D. Plaisted and A. Sattler-Klein. Proof lengths for equational completion. *Information and Computation*, 125(2):154–170, 1996.

[28] H. Seidl. Deciding equivalence of finite tree automata. *SIAM Journal on Computing*, 19(3):424–430, 1990.

[29] J. Slaney. FINDER: Finite domain enumerator - system description. In *Proc. CADE-12*, volume 814 of *LNCS*, pages 798–801. Springer Verlag, 1994.

[30] W. Snyder. A fast algorithm for generating reduced ground rewriting systems from a set of ground equations. *Journal of Symbolic Computation*, 15:415–450, 1993.

[31] T. Tammet. *Resolution Methods for Decision Problems and Finite Model Building.* PhD thesis, Chalmers University of Technology, Göteborg, Sweden, 1992.

[32] S. Vorobyov. An improved lower bound for the elementary theories of trees. In *Proc. CADE-13*, number 1104 in LNAI, pages 275–287. Springer Verlag, 1996.

[33] S. Vorobyov and A. Voronkov. Complexity of nonrecursive logic programs with complex values. In *Proc. PODS'98*, pages 244–253. ACM Press, 1998.

[34] J. Zhang and H. Zhang. System description: Generating models by SEM. In *Proc. CADE-13*, volume 1104 of *LNAI*, pages 308–312. Springer Verlag, 1996.

On the complexity of linear and stratified context matching problems

Manfred Schmidt-Schauß[1] and Jürgen Stuber[2]

[1] Fachbereich Informatik, J.-W.-Goethe-Universität, Postfach 11 19 32, D-60054 Frankfurt, Germany. Tel: (+49)69-798-28597, Fax: (+49)69-798-28919, E-mail: schauss@ki.informatik.uni-frankfurt.de

[2] INRIA LORIA, 615 Rue Jardin Botanique, F-54600 Villers-les-Nancy, France. Tel: (+33)383-59 30 36, Fax: (+33)383-27 83 19, Email: stuber@loria.fr

Abstract. We show that stratified context matching (SCM) is NP-complete, but that linear context matching (LCM) and stratified simultaneous monadic context matching (SSMCM) are in P. SSMCM is equivalent to stratified simultaneous word matching (SSWM).

1 Introduction

Context matching extends first-order matching by the availability of context variables which may be instantiated by a context, that is a term with a hole. Standard first-order matching allows only (term) variables, variables that may be instantiated by a first-order term. While these are restricted to the leaves of a term, context variables can occur as monadic operators anywhere inside a term. Context matching allows much more freedom in the selection of subterms, which may be of use for example for querying data that is available in the form of a large term, like XML documents. For example, we may wish to select the titles of all subsections in a certain section of an XML document. We can express this by the linear context matching problem

$$X(\text{section}(\texttt{"Symbolic Data"}, Y(\text{subsection}(y, z)))) = s$$

where X and Y are context variables and y and z are first-order variables. Here each subsection of the section with title `"Symbolic Data"` will result in one solution of the matching problem with y bound to its title and z to its contents. More general queries to databases may be expressed as nonlinear but stratified context matching problems. For example, we might have a database *db* in a more flexible XML-like format like

$$\begin{array}{l} .(\text{book}(.(\text{author}(a_1), .(\text{title}(t_1), \text{nil}))), \\ \quad .(\text{book}(.(\text{author}(a_1), .(\text{title}(t_2), \text{nil}))), \\ \quad\quad .(\text{book}(.(\text{author}(a_2), .(\text{title}(t_3), \text{nil}))), \text{nil}))), \end{array}$$

where . is a binary list constructor to encode variable arity. We can pick out the titles of the books of author a_1 with a query $X(\mathit{book}(Y(\mathit{author}(a_1))))) =$

$db \wedge X(book(Z(title(x)))) = db$ by looking at the substitution for x. This query is stratified, because the two occurrences of X have the same prefix of context variables above them, which is empty here.

These two examples show that context matching can be used similarly to XPath [4] matching which is used in the XSLT transformation language [3] for XML documents, which was our initial motivation to start studying context matching. In the context of XML processing there are other proposed formalism such as regular expression matching [13] that directly take into account the variable arity of XML by allowing regular expressions over arguments. In our context this could be simulated by using argument lists plus regular restrictions on the contexts that can be instantiated for a context variable. This would allow to both separate the list used for emulating variable arity from other function symbols and for enforcing the regular expression types. Technically such regular restrictions could be expressed by Comon's membership constraints [5, 6]. We believe that the complexity results of this paper remain valid under the addition of such a restriction, however we choose not to discuss it in detail to keep the presentation simple.

Another application area for matching techniques is term rewriting [1]. In standard term rewriting the rules are implicitly extended by allowing an arbitrary context above the matching position, and preserving this context over a rewrite step. In many situations more control is needed over the positions where rules are applied, which motivates for example strategies [2]. With context matching we can achieve improved control in a more declarative way by making the context explicit in the rules. A standard rewrite rule $l \Rightarrow r$ would then become a transition rule $X(l) \Rightarrow X(r)$ that is applicable only at the root, and for finer control additional restrictions can be imposed on X, such as a sort discipline [5] or we may consider conditional rules whose conditions refer to X. Since stratified context unification is decidable, Knuth-Bendix like completion on rules with stratified contexts may be feasible, provided rules can be restricted in such a way that stratification is preserved and suitable termination orderings can be found. Logical calculi with context variables treated similar to a builtin theory [22, 23] would also be an interesting application. Together with the regular restrictions mentioned above this could be used for relations that unlike equality are sensitive to contexts, for example order relations for which certain monotonicity laws hold.

In this paper we try to establish a boundary between problems in P and NP-complete problems. To this end we consider the following problems:

Linear context matching (LCM): Variables may occur only once.

Stratified context matching (SCM): For each variable the paths from the root to its occurrences have the same sequence of context variables (its *variable prefix*).

Simultaneous stratified monadic context matching (SSMCM): A stratified conjunction of equations that contains only monadic function symbols. This is equivalent to stratified word matching (SWM).

We show that LCM and SSMCM are in P, while SCM is NP-complete. It is easily seen that context matching is in NP, hence its restriction SCM is in NP, too, and we need only prove NP-hardness. In the technical report [21] we discuss a few more simple restrictions of context matching and give a transformation rule base algorithm for solving context matching problems, together with a discussion how to improve its performance for easy cases.

General context matching was previously known to be NP-complete [19]. Context matching is a restricted form of linear higher-order matching, which was shown to be in NP and hence NP-complete by de Groote [9]. Here linear means that only solutions where all functions are linear, i.e. contain each of their bound variable exactly once, are considered. A context may be viewed as a linear second-order function with one argument, where the binder is left implicit and the hole is the single occurrence of the bound variable. An algorithm for general second-order matching is given by Huet and Lang [14]. Curien, Qian and Shi [8] give an algorithm for second-order matching that improves efficiency for the case of right-hand sides with many bound variables and few constants. Second-order and third order matching are NP-complete [7], while fourth-order matching is decidable but NEXPTIME-hard [24]. For orders above four it is still open whether higher-order matching is decidable [10]. Recently Loader has shown that higher-order beta-matching is undecidable, but this doesn't imply anything about decidability of the beta-eta case [16].

Hirata, Yamada and Harao [12] have studied the complexity landscape of the second-order matching problem with respect to several restrictions, i.e. number of second-order variables, number of occurrences of variables, ground, function-free, but not stratification.

Our interest for the stratified fragment has been motivated by the results on the decidability of stratified context unification [19, 18, 17]. Other fragments where unification is decidable are the varity 2 fragment [15][1] and the two-context-variable fragment [20], whose corresponding matching problems we consider in [21].

2 Preliminaries

We assume a fixed infinite set $\mathcal{X}$ of (individual) variables, a fixed infinite set $\mathcal{C}$ of context variables, and a fixed set Σ of function symbols of fixed arity. We will use x, y, z for individual variables, X, Y, Z for context variables, a, b, c, d for constants and f, g, h for other function symbols.

Context terms are constructed from the variables and function symbols in the usual way, where context variables are considered as monadic function symbols.

A *context* is a term with a single occurrence of the special operator $\square$, the *hole*. To emphasize that a term C is a context we write $C[\square]$ or just $C[]$. A context $C[]$ may be applied to a term t, written $C[t]$, and the result is the term consisting of C with $\square$ replaced by t. We specify the position of a subterm by $C[t/p]$.

[1] Note that the proof of decidability for the stratified case is flawed in this paper.

A *substitution* is a mapping from individual variables to terms and from context variables to contexts, such that almost all individual variables are mapped to themselves, and almost all context variables are mapped to themselves applied to the hole. Substitutions are extended to terms as follows:

$$\begin{aligned} f(t_1, \ldots, t_n)\sigma &= f(t_1\sigma, \ldots, t_n\sigma) \\ x\sigma &= \sigma(x) \\ (X(t))\sigma &= \sigma(X)[t\sigma]. \end{aligned}$$

3 Problem definition

We first briefly discuss different forms of equations and show that they are equivalent.

For instance, we may consider equations between (context) terms or between contexts themselves, where the holes must match. To get from a term equation to a context equation we may use a binary function symbol to put the hole into a side branch, i.e. reduce $s \approx t$ to $h(\square, s) \approx h(\square, t)$. In the other direction we may replace the hole by a special constant that doesn't occur elsewhere. This assumes that such a binary function symbol or constant exists.

We also consider *multi-contexts*, which are terms with an arbitrary number of occurrences of the hole. We write multi-contexts as $C[\square, \ldots, \square]$. Such a multi-context may be viewed as a second order lambda term $\lambda x_1 \ldots \lambda x_n.C[x_1, \ldots, x_n]$ where the bound variables $x_1, \ldots, x_n$ occur each exactly once and in left-to-right order. So $f[\square, \square]$ corresponds to $\lambda x \lambda y\, f(x, y)$ but not to $\lambda x \lambda y\, f(y, x)$. We may replace a subset of holes by terms by writing for instance $C[\ldots, s_1, \ldots, s_2, \ldots]$ where the dots stand for holes. Where we need to be more precise we indicate the number of holes by a subscript, i.e. $C[\ldots_i, s, \ldots_j]$. Note that we do not allow variables that stand for multi-contexts.

If we have an equation between multi-contexts, e.g. $C[f(s_1[\square], s_2[\square])] \approx C[g(t_1[\square], t_2[\square])]$, then the first (resp. second) hole on the left-hand side must match the first (resp. second) hole on the right-hand side, which forces a match between the function symbols f and g at the longest common prefixes of the positions of the holes. Thus either $f \neq g$ and there is no solution, or we can split this equation into the equations $s[] \approx t[]$, $s_1[] \approx t_1[]$ and $s_2[] \approx t_2[]$, reducing the multi-context equation to a conjunction of context equations. This elimination of multi-contexts can be done in linear space and time.

If at least one symbol of arity at least two is available, then a conjunction of equations can be coded as a single equation. For example, if a symbol h with arity 2 is available, we may replace

$$s_1 \approx t_1 \wedge \ldots \wedge s_n \approx t_n$$

by

$$h(\ldots h(s_1, s_2) \ldots, s_n) \approx h(\ldots h(t_1, t_2) \ldots, t_n).$$

By these equivalences we can define a *context equation* as any equation between context terms, contexts or multi-contexts, and a *context matching problem* P

as either a single context equation or a conjunction of context equations such that all right-hand sides of equations are ground (we will be more precise when the function symbols assumed above are not available). A substitution σ is a *solution* of a context equation $s \approx t$ if $s\sigma = t\sigma$. A solution of P is a substitution that solves all equations in P.

A context matching problem is called *linear* if each variable occurs at most once and *monadic* if all function symbols occurring in it have an arity of at most one.

The *variable prefix* of an occurrence in a term is the sequence of context variables on the path from the root down to and including that occurrence. A context matching problem is called *stratified* if for each individual or context variable all occurrences of that variable in the problem have the same variable prefix. For example, the problem

$$X(f(Y(g(a,x)),Y(g(x,Z(z))))) \approx f(g(a,b),b)$$

is stratified, X having the prefix X, Y the prefix XY, x the prefix XYx, Z the prefix XYZ and z the prefix $XYZz$.

$$X(Y(\square)) \approx f(g(\square)) \wedge Z(X(\square)) \approx h(f(\square))$$

is not stratified, because X occurs with the variable prefixes X and ZX.

4 Linear Context Matching is in P

We can solve linear context matching problems by dynamic programming. We build a table which for every pair $\langle s,t\rangle$ of a subterm s of the left-hand side and a subterm t of the right hand side of the problem records whether s matches t. The table is built recursively from the bottom up:

- If s is an individual variable then the answer is yes.
- If $s = f(s_1,\ldots,s_m)$ and $t = g(t_1,\ldots,t_n)$ then the answer is yes if and only if $f = g$ and s_i matches t_i for $1 \leq i \leq m = n$.
- If $s = X(s')$ then the answer is yes if and only if s' matches some subterm of t.

Let n be the size of the problem. We have $O(n^2)$ table entries of constant size, and to compute each entry we need at most $O(n)$ steps. Hence the algorithm uses $O(n^3)$ time and $O(n^2)$ space, and linear context matching is in P.

5 Stratified Context Matching is NP-complete

To show NP-hardness we reduce 3SAT to SCM. Let $c_1,\ldots,c_m$ be the clauses and $x_1,\ldots,x_n$ the propositional variables in an instance of 3SAT. We encode this as an instance of SCM with $m+1$ equations

$$s_0 \approx t_0 \wedge s_1 \approx t_1 \wedge \ldots \wedge s_m \approx t_m$$

where $s_0 \approx t_0$ is a dummy equation and $s_i \approx t_i$ corresponds to clause c_i. The dummy equation restricts matchings so that they correspond to boolean valuations, and equation i matches with such a valuation if and only if the valuation satisfies clause i. Each equation consists of $n+1$ *layers*, n layers for the variables and one at the bottom with constants that encodes whether clauses are satisfied. Each layer is either an *active* layer that switches between subterms in the right-hand side or a *passive* layer that doesn't. In equation i layer j is active when variable j occurs in clause i, otherwise it is passive. The dummy equation consists only of passive layers. The dummy equation assures that each layer allows only two different matches for two context variables on top of each other, which represent the truth values. In the case of an active layer these matches select different subterms on the right-hand side for matching in lower layers, which allows to test for satisfiability. In the bottom layer we need to distinguish the choices made in the layers above, in order to determine whether the clause is satisfied or not. We record the truth values of literals in a string containing $*$ in the j-th position when there is no literal containing x_j, 0 when there is a literal containing x_j that is false under the chosen assignment, and 1 when there is a literal containing x_j that is true under the chosen assignment. For the dummy equation we construct a satisfiable bottom layer, while for the other equations we chose a satisfiable bottom layer for a certain subterm only when the string leading to that subterm contains at least one 1, meaning that the clause is satisfied in this branch.

To formalize this we recursively define

$$s_{ij} = \begin{cases} f(X_j(g(g(x_{ij}, Y_j(s_{i,j+1})), y_{ij}))) & \text{if } x_j \text{ occurs in } c_i \text{, and} \\ f(X_j(Y_j(s_{i,j+1}))) & \text{otherwise.} \end{cases}$$

$$t_{i,\alpha} = \begin{cases} f(g(g(g(c, t_{i,\alpha 0}), g(t_{i,\alpha 1}, c)), c)) & \text{if } x_j \text{ occurs positively in } c_i, \\ f(g(g(g(c, t_{i,\alpha 1}), g(t_{i,\alpha 0}, c)), c)) & \text{if } x_j \text{ occurs negatively in } c_i \text{, and} \\ f(g(t_{i,\alpha *}, c)) & \text{otherwise.} \end{cases}$$

where $0 \leq i \leq m$ and $1 \leq j = \text{len}(\alpha) + 1 \leq n$. For $i = 0$ we always choose the (passive) otherwise-case. For the leaves we use $s_{i,n+1} = a$ and

$$t_{i,\alpha} = \begin{cases} a & \text{if } i = 0 \text{ or } \alpha \text{ contains at least one } 1 \\ b & \text{otherwise.} \end{cases}$$

Finally the stratified context matching problem P is

$$s_{01} \approx t_{0,\epsilon} \wedge \ldots \wedge s_{m1} \approx t_{m,\epsilon}.$$

It remains to show that P has a solution if and only if the original instance of 3SAT is solvable.

First we show that $s_{i,j}$ can only match $t_{i,\alpha}$ for $1 \leq i \leq m$ and some α with $j = \text{len}(\alpha) + 1$. This holds because there are the same number of fs in the left- and in the right-hand side of the dummy equation, hence none can be matched by a context variable and the layers are separated. Furthermore, the main paths must proceed into the nontrivial subterms, as f doesn't match the constant c. Let us now consider a single layer in the dummy equation. There are two possible matches in layer j:

$$X_j = \square \qquad Y_j = g(\square, c) \tag{1}$$
$$X_j = g(\square, c) \qquad Y_j = \square \tag{2}$$

These are the only matches for a passive layer. Now let us look what happens in a positive active layer: In case (1) the next lower layer $s_{i,j+1}$ is matched against $t_{i,\alpha 1}$, while in case (2) it is matched against $t_{i,\alpha 0}$. For a negative active layer it is vice-versa. Hence a match of the form (1) corresponds to assigning true to the logical variable x_j, and a match of the form (2) corresponds to assigning false to x_j. This in mind we see that at the leaves α reflects the truth values of the literals in the clause. If these make the clause true, we have put the solvable matching problem $a \approx a$ at the leaf, otherwise the unsolvable $a \approx b$. Hence each match corresponds to a satisfying truth assignment and vice-versa.

If we look at the size of P we see that each right-hand side of an equation branches at most at three points, since each clause contains at most three distinct variables. This gives a factor of $2^3 = 8$ but doesn't lead to exponential growth. The depth is proportional to the number of variables, and the number of equations to the number of clauses, hence the size of the context matching problem is linear in the size of the 3SAT instance.

We conclude that 3SAT can be reduced to SCM, and obtain:

Theorem 1 *SCM is NP-complete.*

6 Simultaneous stratified monadic context matching is in P

In this case conjunctions cannot be coded in a single equation, hence we make explicit that we allow them by defining a *simultaneous context matching problem* P as a conjunction of equations whose right-hand sides are ground terms. We will show that simultaneous stratified monadic context matching (SSMCM) is in P.

We may easily eliminate as unsolvable any problem containing an equation that has two distinct constants at the bottom of its left- and right-hand side, and assume from now on that all equations are context equations and have a hole at the bottom. In particular we replace an individual variable at the bottom of a left-hand side by a context variable applied to a hole. With these remarks it becomes obvious that SSMCM is equivalent to simultaneous stratified word matching, where the context variables become ordinary variables, the hole at the bottom is omitted, and where the prefix of a variable consists of all variables to its left. To simplify the notation we will consider terms as words in the rest of this section. We write ε for the empty word.

The crucial property that leads to a polynomial algorithm in this case is that there are no branchings in the terms. For any equation $w_1X_1 \dots w_nX_n \approx w$ we know that a solution σ must satisfy $|(X_1 \dots X_n)\sigma| = |w| - |w_1 \dots w_n|$, hence we call $|w| - |w_1 \dots w_n|$ the *substitution length* of this equation. Equations with negative substitution length and conjunctions containing two equations with the same context variables but different substitution lengths are inconsistent.

We give a transformation algorithm to solve SSMCM. At each step of the transformation algorithm the problem P is a conjunction of disjunctions of conjunctions of match equations. More precisely, it has the form $P_{s_1} \wedge \dots \wedge P_{s_n}$ where $\{s_1, \dots, s_n\}$ is the set of all variable prefixes for rightmost variables in P, and P_{s_i} is a disjunction of conjunctions of equations whose rightmost variable on the left-hand side has the prefix s_i. The number of prefixes is bounded by the number of prefixes in the initial problem, which in turn is bounded by the number of variables and hence the size of the problem. For each prefix s we require that P_s has the form $P_{s,0} \vee \dots \vee P_{s,k}$ where $P_{s,i}$ is a possibly empty conjunction of equations whose substitution length is i. Since the initial problem is a conjunction that is consistent with respect to substitution length, the disjunction for a given variable prefix contains a single nonempty conjunction for this substitution length. For the bound k we can use the largest substitution length in the initial problem, which is bounded by the size of the problem.

The transformation algorithm consists of simplification rules that are applied eagerly anywhere inside the problem, and the Variable Elimination rule that is applied to a full disjunction and only when no other rule is applicable. The rules are iterated until one of the solved forms $\top$ or $\bot$ is obtained. The simplification rules are the rules for idempotency and the laws of true and false together with the following rules:

Delete $\varepsilon \approx t \to \top$

if $t = \varepsilon$; otherwise fail.

Bottom Decompose $sf \approx tg \to s \approx t$

if $f = g$; otherwise fail,

where f and g are function symbols.

Substitution Length Clash 1 $s \approx t \to \bot$

if the substitution length of $s \approx t$ is negative.

Substitution Length Clash 2 $s_1 \approx t_1 \wedge s_2 \approx t_2 \to \bot$

if s_1 and s_2 have the same rightmost variable but $s_1 \approx t_1$ and $s_2 \approx t_2$ differ in substitution length.

We say a problem is *simplified* if these rules have been applied exhaustively.

Variable Elimination eliminates a rightmost context variable X by generating a disjunction of $k+1$ conjunctions from each original conjunction, where each conjunction corresponds to a possible length of the word substituted for X.

Variable Elimination $$P_{s,0} \vee \ldots \vee P_{s,k} \rightarrow \bigvee_{0 \leq i \leq k} \bigvee_{0 \leq j \leq k} P_{s,i}\{t_{s,j}/X\}$$

> if X is a variable with maximal variable prefix s, k is the maximal substitution length in P_s, and $t_{s,j}$ is a suffix of length j in some right-hand side of P_s.

Note that X having the maximal variable prefix s implies that X is the rightmost variable in the left-hand sides of equations in the disjunction $P_{s,0} \vee \ldots \vee P_{s,k}$ and that X occurs nowhere else in the problem. Note also that we do not keep the bindings of the variables, as in general this results in an exponential number of solutions.

We observe that for an equation transformed by Bottom Decompose each side of the result is a prefix of the corresponding side of the original equation, and that the other simplification rules do not generate new equations. For Variable Elimination this is not true for the left-hand sides, as a word is substituted for the variable. However, simplification by Bottom Decompose removes this word, so Variable Elimination followed by eager simplification also preserves this prefix property, and hence it is preserved by the algorithm in general.

An equation in a simplified problem is completely determined by the initial equation it is derived from, its variable prefix and its substitution length. The last variable in the variable prefix determines the end of the left-hand side, and given the left-hand side the substitution length determines the length of the right-hand side. Since variable prefix and substitution length are fixed for each inner conjunction $P_{s,i}$, it contains at most one equation derived from each initial equation. The top-level conjunction partitions the equations with respect to the variable prefix of their rightmost variable, hence the descendants of an original equation are grouped inside one disjunction, and since any inner conjunction can contain at most one of them, each original equation can lead to only k equations in an intermediate problem. This implies that the size of intermediate simplified problems is $O(n^2)$. Since the temporary blowup in Variable Elimination is also of size $O(n^2)$ all intermediate problems have size $O(n^2)$.

Theorem 2 *This algorithm decides solvability of SSMCM.*

Proof: By inspection we see that the rules preserve solvability. Any equation without a variable at the end of its left-hand side can be simplified either by Delete or Substitution Length Clash 1 if one side is empty, or by Bottom Decompose otherwise. To a simplified problem Variable Elimination is applicable. In particular, choosing k as the maximal substitution length of P_s ensures that a sufficiently long right-hand side exists. Hence a rule applies to any problem that is not in solved form. For termination we use as measure the lexicographic combination of the number of variables and the size of the problem.

There are $O(n)$ variables and each variable is removed in $O(n^2)$ steps, hence the derivation has at most length $O(n^3)$. We obtain the following theorem:

Theorem 3 *SSMCM is solvable in time $O(n^3)$ and space $O(n^2)$.*

7 Conclusion and Further Work

A borderline case that is a strengthening of stratification remains open: Consider all the variable prefixes of variables in the problem, ordered by the prefix ordering. For a general stratified context matching problem the Hasse diagram for the prefix ordering forms a tree. It is open whether stratified context matching problems with a linear prefix ordering are in P or NP-complete. In other words, this restricted *linearly stratified* case is defined by the property that there are no two different variables immediately below a context variable (ignoring function symbols).

Parallelism constraints [11] are equivalent in power to context unification but perform better on certain linguistic problems. It could be interesting to consider a matching variant of this problem where the tree is given.

Acknowledgments

We thank Claude Kirchner whose suggestion initiated this work.

References

1. Franz Baader and Tobias Nipkow. *Term rewriting and all that.* Cambridge University Press, Cambridge, UK, 1998.
2. Peter Borovanský, Claude Kirchner, Hélène Kirchner, and Christophe Ringeissen. Rewriting with strategies in ELAN: a functional semantics. *Int. Journal of Foundations of Computer Science*, 1999.
3. James Clark, editor. *XSL Transformation (XSLT) Version 1.0.* W3C, 16 November 1999. `http://www.w3.org/TR/1999/REC-xslt-19991116`.
4. James Clark and Steve DeRose, editors. *XML Path Language (XPath) Version 1.0.* W3C, 16 November 1999. `http://www.w3.org/TR/1999/REC-xpath-19991116`.
5. Hubert Comon. Completion of rewrite systems with membership constraints. Part I: Deduction rules. *Journal of Symbolic Computation*, 25(4):397–419, 1998.
6. Hubert Comon. Completion of rewrite systems with membership constraints. Part II: Constraint solving. *Journal of Symbolic Computation*, 25(4):421–453, 1998.
7. Hubert Comon and Yan Jurski. Higher-order matching and tree automata. In *Proc. Conf. on Computer Science Logic (CSL-97)*, LNCS 1414, pages 157–176, Aarhus, 1997. Springer.
8. Régis Curien, Zhenyu Qian, and Hui Shi. Efficient second-order matching. In *Proc. 7th Int. Conf. on Rewriting Techniques and Applications*, LNCS 1103, pages 317–331, New Brunswick, NJ, USA, 1996. Springer.
9. Philippe de Groote. Linear higher-order matching is NP-complete. In *Proc. 11th Int. Conf. on Rewriting Techniques and Applications*, LNCS 1833, pages 127–140, Norwich, UK, 2000. Springer.

10. Gilles Dowek. Higher-order unification and matching. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, chapter 16, pages 1009–1062. North-Holland, 2001.
11. Katrin Erk and Joachim Niehren. Parallelism constraints. In *Proc. 11th Int. Conf. on Rewriting Techniques and Applications*, LNCS 1833, pages 110–126, 2000.
12. Kouichi Hirata, Keizo Yamada, and Masateru Harao. Tractable and intractable second-order matching problems. In *Proc. 5th Ann. Int. Computing and Combinatorics Conference (COCOON'99)*, LNCS 1627, pages 432–441. Springer, 1999.
13. Haruo Hosoya and Benjamin Pierce. Regular expression pattern matching. In *Proc. 28th Ann. Symp. on Principles of Programming Languages (POPL)*, 2001.
14. Gérard Huet and Bernard Lang. Proving and applying program transformations expressed with second-order patterns. *Acta Informatica*, 11:31–55, 1978.
15. Jordi Levy. Linear second-order unification. In *Proc. 7th Int. Conf. on Rewriting Techniques and Applications*, LNCS 1103, pages 332–346, 1996.
16. Ralph Loader. Higher order β matching is undecidable. `http://homepages.ihug.co.nz/~suckfish/match/beta.ps`, October 2001.
17. Manfred Schmidt-Schauß. A decision algorithm for stratified context unification. Frank-Report 12, Fachbereich Informatik, J.W. Goethe-Universitat Frankfurt, Frankfurt, Germany, 1999. Accepted for publication in the Journal of Logic and Computation.
18. Manfred Schmidt-Schauß. Stratified context unification is in PSPACE. In *CSL 2001*, LNCS 2142, pages 498–512. Springer, 2001.
19. Manfred Schmidt-Schauß and Klaus U. Schulz. On the exponent of periodicity of minimal solutions of context equations. In *Proc. 9th Int. Conf. on Rewriting Techniques and Applications*, LNCS 1379, pages 61–75, Tsukuba, Japan, 1998. Springer.
20. Manfred Schmidt-Schauß and Klaus U. Schulz. Solvability of context equations with two context variables is decidable. Accepted for publication in the Journal of Symbolic Computation, 2001.
21. Manfred Schmidt-Schauß and Jürgen Stuber. On the complexity of linear and stratified context matching problems. Rapport de recherche A01-R-411, LORIA, December 2001.
22. Jürgen Stuber. Deriving theory superposition calculi from convergent term rewriting systems. In *Proc. 11th Int. Conf. on Rewriting Techniques and Applications (RTA 2000)*, volume 1833 of *LNCS*, pages 229–245, Norwich, UK, 2000. Springer.
23. Jürgen Stuber. A model-based completeness proof of Extended Narrowing And Resolution. In *Proc. 1st Int. Joint Conf. on Automated Reasoning (IJCAR-2001)*, LNCS 2083, pages 195–210, Siena, Italy, 2001.
24. ToMasz Wierzbicki. Complexity of the higher-order matching. In *Proc. 16th Int. Conf. on Automated Deduction (CADE-16)*, LNAI 1632, pages 82–96, Trento, Italy, 1999. Springer.

Computational Complexity of a Constraint Model-based Proof of the Envelope of Tendencies in a MAS-based Simulation Model

Oswaldo Terán[*][†], Bruce Edmonds[*]

[*]Centre for Policy Modelling, Manchester Metropolitan University,
Aytoun Building, Aytoun Street, Manchester, M1 3GH, UK.
Tel. +44 161 247 6478 *Fax.* +44 161 247 6802
{o.teran,b.edmonds}@mmu.ac.uk

[†]Department of Operation Research and Centre for Simulation
and Modelling, Universidad de Los Andes. Venezuela
Tel. +58 274 240 2879
oteran@ula.ve

Abstract.. This paper determines the complexity of a constraint model-based proof of the envelope of a tendency in the dynamics of a Multi-Agent-based simulation model. The proof is performed via a constraint model-based exploration of simulation trajectories using forward inference, by means of which a whole fragment of the simulation model theory is investigated. Such exploration allows for all simulation trajectories defined by a range of the model's parameters and a range of the agents' choices. The paper verifies that the search is *PSPACE-complete* for an infinite number of iterations, and suggests that the search is Σ_i *P-complete* for a finite number of iterations.

1 Introduction

There is a *need for* studying and *proving (emergent) tendencies* in the *simulation of social systems* (including simulation of organizations). This need has been especially remarkable in those works related with elaborating or testing theories [1, 2, 4]. Such a need has not been satisfied by *traditional approaches* for exploring the dynamics of simulations models, such as Scenario Analysis and the Monte Carlo method. Neither of these approaches performs exhaustive explorations of simulation trajectories in subspaces of the simulation theory. The explored trajectories are chosen, in the first case, by a domain expert, and in the second case, randomly. Owing to these facts, those approaches cannot be used for proving tendencies in the dynamics of a simulation model - the allowed conclusions are valid either according to the expertise of a domain expert or statistically.

As an *alternative* to these traditional methods, in previous papers [6-8] a *hierarchy of computational architectures* for searching for and proving tendencies in a Multi Agent System (MAS)-based simulation model is proposed. The first architecture, that at the higher level, consists of the MAS-based model where tendencies will be searched for by the modeller. After a tendency is found, at a second architectural level, a constraint logic model[1] proof of the envelope of the simulation trajectory is proposed. In those papers, a computational technique for doing this proof efficiently is implemented and illustrate by using an example. And, at a third architectural level, a more general proof of the envelope of the tendency would be implemented by exploring a wider fragment of the simulation theory by using a syntactic driven search. As explained better in [8], *this research contributes in bringing closer the simulation and the logic programming communities*.

This paper examines the computational complexity of the procedure implemented in the second architectural level. First, in the second section, the idea of envelope is reviewed. Afterwards, in the third section, the logic-based exploration of simulation trajectories implemented for proving the envelope of tendencies in simulation models is described. Then, in the fourth section, the computational complexity of such exploration is established. And finally, in the fifth section, some conclusions are presented.

2 Enveloping Tendencies in a Simulation Model

It does not seem convenient to use always the strong concept of envelope managed in mathematics. Apart from precision on the managed concepts, the idea of making the output comprehensible for a modeller is also important. An envelope will be chosen considering the trade-off between practical usefulness for a modeller and precision (by precision we mean how close the concept is to the ideal mathematical notion of a tangent curve/surface).

Consider the case of *enveloping a single simulation output, Y*. Each trajectory will generate a sequence of real values over time, Y. Calling y_{ij} the output value at time instant i for trajectory j, an envelope might consist of two sequences of values over time: E_{upper} and E_{lower}, which in some sense cover all trajectories. The value of E_{upper} at time instant i must be greater than or equal to y_{ij} for all j, and E_{lower} at time instant i must be lower than or equal to y_{ij} for all j. That is, the envelope would be given by two sequences of values over time, where for each time instant all values generated by the simulation trajectories are enclosed by the two values given by these two value sets. Putting this in other words, at each time instant, t, the smallest interval covering all points generated by the explored trajectories is included in the interval given by the two sequences E_{upper} and E_{lower} for instant t.

Alternatively, first an approximating function, f, for the output value set Y that each trajectory generates might be elaborated; then, the instances of these functions (one function for each trajectory) might be enveloped.

Among the procedures of interest for enveloping tendencies in simulation studies might be the followings:

- Enveloping certain *properties of the observed tendency rather than the tendency itself*. The results might permit one to relate the simulation results to theory developments and to elaborate conclusions with respect to the theory underlying the simulation model.
- Producing a *mathematical description* of some coarse borders of the space where the tendencies have been observed. This is useful if it is difficult to describe the subspace of the tendencies directly. Then, coarse borders are chosen as a first

[1] The term 'logical model' means model in the logical sense, which is different to the idea of model in modeling or in simulation. In this paper, a logical model corresponds to a simulation trajectory.

approximation to the envelope and, afterwards, these enclosing borders are expressed mathematically.

- *Using extreme cases of representative or typical instances* of a tendency. It is assumed that the observed tendencies in the simulation can be grouped qualitatively as similar or close enough (in accordance with some criteria) to a finite (small) number of typical tendencies.
- *Specifying a range of parameters and choices.* This is the description used in the exploration implemented in previous papers [6-8].

3 Proving Tendencies Via a Model-based Exploration of Simulation Trajectories in a MAS-based Simulation Model

3.1 Logical Model-Constrained Exploration of Simulation Trajectories

A simulation - either an event-driven, or a finite differences, or a MAS-based - can be seen as a partial logical model. Usually, in a trajectory only a partial set of all the facts of the logical model corresponding to the trajectory are explicitly generated. This partial set consists of those facts that are relevant, either because they are required for the modeller as outputs or because they are necessary to generate the simulation transition states. The remaining facts are left as unknown.

There are different methods to specify a theory in a language. One commonly employed in logic is by using a set of formulas of the language which become the axioms of the theory. In a declarative program a simulation model is specified via a set of rules and the underlying logic of the program. *Potential trajectories are defined via non-deterministic factors* of the simulation, e.g., parameters and choices.

The idea in the referenced previous studies was to analyse the *emergence of tendencies* in a simulation by exploring a subspace of the space of trajectories. That was done via a logical model-based constraint search, where the constraints standed for the selected parameters and choices. The exploration allows a modeller to explore that fragment of the simulation theory content over a range of parameters and choices (see Figure 1). Consequently, the resulting conclusions and proofs will be valid over that fragment of the theory and, under appropriate justifications, they can be extrapolated to the whole simulation theory.

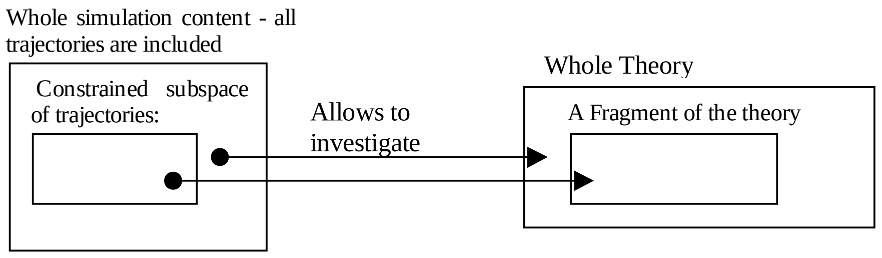
Whole simulation content - all trajectories are included
Constrained subspace of trajectories:
Allows to investigate
Whole Theory
A Fragment of the theory

Figure 1. Theory given by simulation trajectories

3.2 Logical Model Exploration for Proving the Necessity of a Tendency

The idea is to generalise about tendencies going from the observation of individual trajectories to observation of a group of trajectories generated for certain parameters and choices. In particular, it is intended to know if a certain tendency is necessary or contingent in the explored trajectories. We understand a simulation trajectory as a logical model embedded in a simulation program (a 'possible world' in semantic terms) and involving trajectories of entities (e.g., agents) inside the simulation and, hence, different from trajectories of these entities. It is a cross-product of all settings of the structure of the simulation model and all processes (e.g., agents' choices) into one path through a high-dimensional space (see Figure 2).

The character of the search is predominantly logical model, constraint, forward-chaining, and clausal ordered. A logical model is generated for each combination of parameters and choices. Each combination of parameters provides a different structure of the simulation model (see Figure 3). 'Paths' representing trajectories are generated for each structure. Then, while the simulation is going on, choices produce branch points where alternative settings for each choice turn out into a different simulation trajectory.

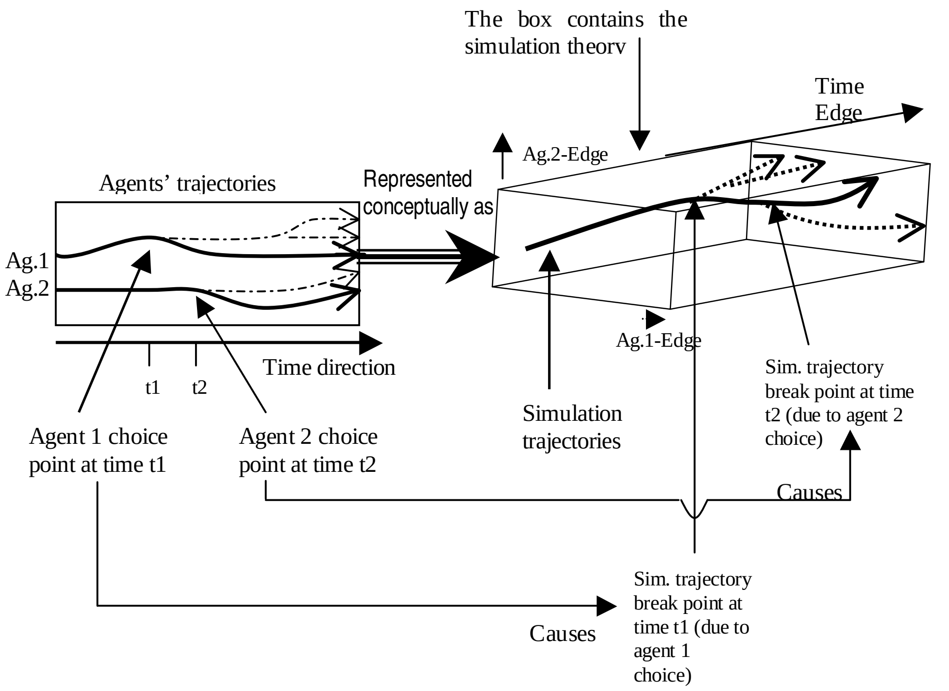
The box contains the simulation theory
Time
Edge
Ag.2-Edge
Agents' trajectories
Represented conceptually as
Ag.1
Ag.2
Time direction
t1
t2
Ag.1-Edge
Sim. trajectory break point at time t2 (due to agent 2 choice)
Simulation trajectories
Agent 1 choice point at time t1
Agent 2 choice point at time t2
Causes
Sim. trajectory break point at time t1 (due to agent 1 choice)
Causes

Figure 2. Representation of a simulation theory in terms of the simulation trajectories, and of these in terms of agents' choices (for a single parameter-setting and assuming there are two agents)

This exhaustive constraint-based search over a range of possible trajectories makes it possible to establish the necessity of postulated emergent tendencies. Following a procedure similar to that used in theorem-proving [3,10], a subset of the possible simulation parameterisations and agent choices is specified, the target emergent tendencies are prearranged in the form of negative constraints, and an automatic search over the possible trajectories is performed.

Tendencies are shown to be necessary, with respect to the range of parameterisations and non-deterministic choices, by first finding a possible trajectory without the negative constraint to show the rules are consistent and then showing that all possible trajectories violate the negation of the hypothetical tendency when this is added as a further constraint. This is equivalent to showing that all possible tendencies obey the positive form of the constraint, i.e., that the positive form is true for all tendencies.

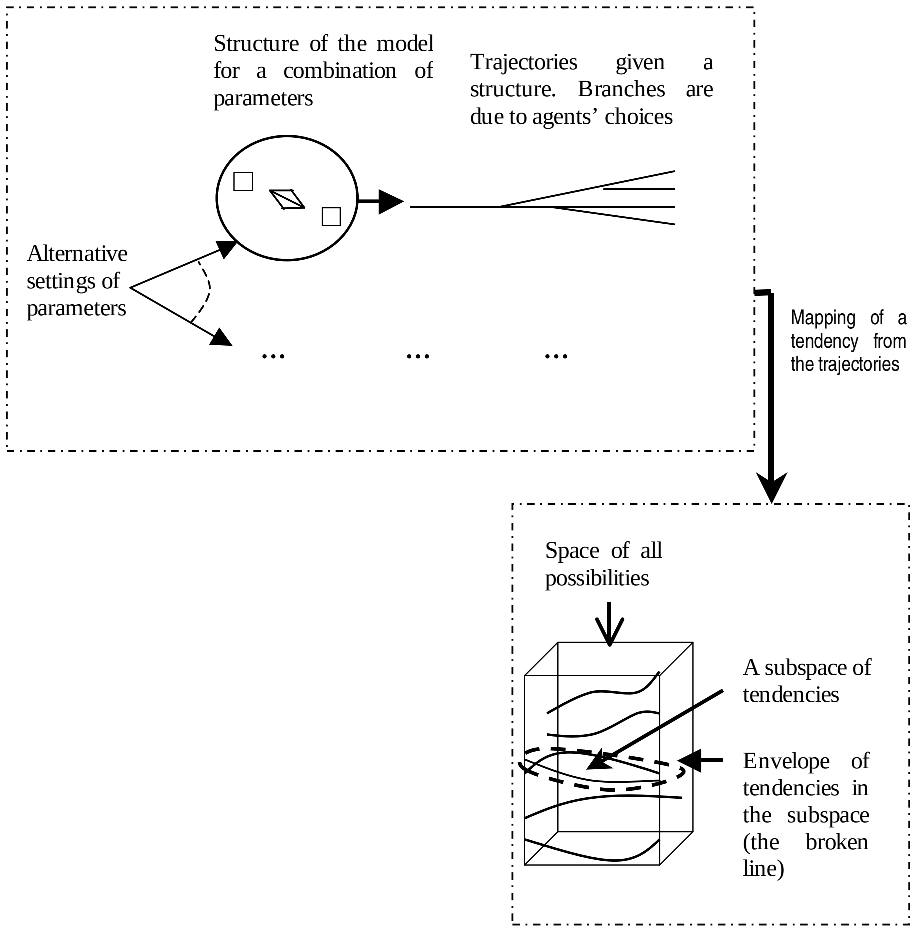
Structure of the model for a combination of parameters
Trajectories given a structure. Branches are due to agents' choices
Alternative settings of parameters
...
...
...
Mapping of a tendency from the trajectories
Space of all possibilities
A subspace of tendencies
Envelope of tendencies in the subspace (the broken line)

Figure 3. A model constraint-based exploration of the dynamics of a simulation model

4 Complexity of the Proof

The *aim* of this section is to demonstrate that the exploration of trajectories proposed in the previous section applied over an infinite (theoretically) number of iterations is *PSPACE-complete.* To make clearer the exposition, this aim is called the *target problem*. As is usual for this sort of verification, two steps are followed:

First, it will be proved that the target problem is in PSPACE by expressing it as a binary tree of depth *n*. According to Papadimitriou [5] this is sufficient (see examples in [5], pp. 455-462).

Second, it will be proved that the problem is also *PSPACE-complete* by translating another *PSPACE-complete* problem into the target problem. For this comparison, one of the problems Woolridge presents in [9] has been chosen, concretely that of agent-task-maintenance.

For the first part of the proof it must be possible to construct in polynomial space the game three, which is possible if the target problem is expressed in the form of a Boolean quantified expression (see examples 19.1 and 19.2 in [5]), as follows:

$$\exists x_1 \ \forall x_2 \ \exists x_3 \ \forall x_4 \ \exists x_5 \ldots Q_n \, x_n \ (F) \qquad (1)$$

where F is the formula to be evaluated over the variables $x_1 \ldots x_n$, and Q_n is the last quantifier, which will be $\exists$ in case of *n impair* or $\forall$ in case of *n even*.

The *impair* variables correspond to the environment's action. The *deterministic* part in the state transition of the simulation will be called *environment's actions*. In the target example, it corresponds to all those changes not associated with *agents' choices*. Consequently, there is only one alternative action for the *impair* variables. The *even* variables correspond to the agents' choices (which are going to be called *agents' actions*). In the particular case of the example presented in [6], there are eight alternative agents' choices. So far, a state transition in a simulation has been divided into two parts: that *deterministic* part associated with the *existential* variables and that *non-deterministic* part associated with the *quantified* variables. A whole simulation path (or simulation trajectory) is represented by a concatenation of branches, where each branch corresponds to an assignment of values to a variable x_i.

Finally, F will be the question: whether the searched *tendency* has occurred in a simulation trajectory, where that trajectory is associated with an assignment of values for the variables, x_i. The whole expression *(1)* is true if for all possible assignments of values to the variables the tendency is true (remember that there is only one choice for the existential variables). As each particular assignment of values to the whole set of quantified variables corresponds to a simulation trajectory, the proof is successful if this expression is valid for all possible values the quantified variables can take! (e.g., for all possible agents' choices).

To check if the proof is successful, a boolean circuit, where and *OR* gate stands for the $\exists$ quantifier and an *AND* gate stands for the $\forall$ quantifier, is written (see Figure 4). A leaf in this circuit is evaluated to *true* if the tendency is found in the corresponding simulation path and to *false* otherwise. The whole circuit will be *true* if and only if the tendency appears in all simulation paths. Hence, the proof is successful if and only if the circuit is true (e.g., the tendency is found in *all* paths).

These two expressions found of the problem (that is, the Boolean circuit shown in figure 4 and the expression of equation *(1)*) are sufficient to prove that the target problem is *PSPACE*. The next task is to prove that the problem is *PSPACE-complete*.

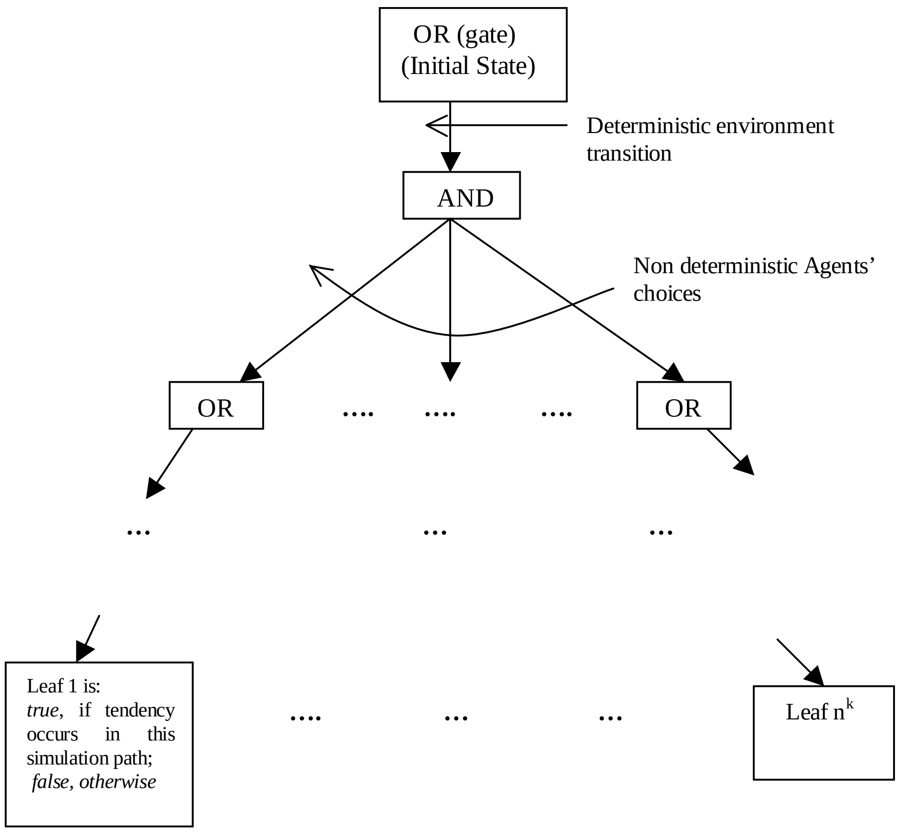
OR (gate)
(Initial State)
Deterministic environment transition
AND
Non deterministic Agents' choices
OR
OR
Leaf 1 is: true, if tendency occurs in this simulation path; false, otherwise
Leaf n^k

Figure 4. Boolean circuit for the target problem

Comparing with Woolridge [9], an algorithm to check the proof might be written. This will bring the example close to the one he uses when considering *maintenance tasks*. Assuming a Turing machine *M* is called recursively at each branch point (at agents' choices) and that this machine is kept in use while actions are deterministic (environment's action), the algorithm for *M* will be:

Algorithm 1:

1. If the tendency appears, then the branch is evaluated to *true (success*);
2. If there are no allowable simulation actions, the branch is evaluated to *false* (*fail);*
3. Execute the deterministic aspects of the state transition (environment action), then for each agent's choice recursively call *M;*
4. If all recursive calls in 3, are *successful* (i.e., evaluated to *true*), then *M* is *true (success).*

To prove that the target problem is *PSPACE-complete*, consider the maintenance problem in [9]. There, agents are chosen non-deterministically to act against the environment. Agent's actions are deterministic, while environment's actions are non-deterministic. The idea is to check if there is any choice of agents' actions that is successful in bringing the environment into one in a set of states whatever the environment chooses. It is like a game where agents play against the environment. Woolridge proves that the agent-maintenance problem is in NPSPACE using the following *algorithm*:

Algorithm 2:

```
1. if r [the run until a branch point] ends with state ∈ G [the set of goals], then M accepts;
2. if there are no allowable actions given r, then M rejects;
3. non-deterministically choose an action α from Ac (possible agents' actions, there is one per
   agent) and then for each e ∈ τ (set of possible environment's states) recursively call M with
   the run r · α · e;
4. if all of these accept, then M accepts, otherwise M rejects.
```

In Woolridge's problem, rather than searching for a tendency, the idea is to bring the simulation into one among a set of environment states. If the environment is brought into one of these states, it is said that the selected agents have been successful in their game against the environment. In Woolridge's example, the agents' actions are deterministic; e.g., they have only one choice, but different agents can be selected. Selection of agents corresponds to the *OR* nodes in the circuit shown in the Figure 4, each branch corresponding to the choice of a different agent. On the other hand, the environment has non-deterministic actions, and, correspondingly, their choices are associated with the *AND* gates in the circuit.

A difference between *algorithm 1* and Woolridge's algorithm (e.g., *algorithm 2*) is that in the latter, at step 3, *M* is called for each possible environment's state after an agents' choice is selected non-deterministically, while in the former *M* is called, in step 3, for each agent's choice after the (deterministic) environment action is performed. So, the deterministic action of the environment in step 3 in the former algorithm corresponds to the non-deterministic choice of agents in the latter. Consequently, though the translation of Woolridge's problem into the target problem seems straightforward, there is still a small difficulty: his case study is non-deterministic (owing to the non-deterministic choice of agents in step 3 in *algorithm 2*), while the target problem is deterministic. Woolridge's original problem is NPSPACE.

To solve the difficulty, consider the deterministic version of Woolridge's problem. Think about checking the successfulness of agents' actions in his problem once an agent has been chosen in advance at each branch point. This is a deterministic problem. It is in PSPACE but still as hard as Woolridge's original one as NPSPACE = PSPACE ([5], p. 150). Woolridge's algorithm for this deterministic version of the agent-maintenance task problem becomes:

Algorithm 3:

```
1. if r [the run until a branch point] ends with state ∈ G [the set of goals], then M accepts;
2. if there are no allowable actions given r, then M rejects;
3. deterministically use the action α given in advance from Ac (possible agents' actions, there is
   one per agent) and then, for each e ∈ τ (set of possible sates of the environment),
   recursively call M with the run r · α · e;
4. if all of these accept, then M accepts, otherwise M rejects.
```

The translation of the determinist version of Woolridge's problem into the target problem is straightforward from *algorithms 1* and *3*. The deterministic action of the environment at step 3 of *algorithm 1* corresponds in Woolridge's algorithm (*algorithm 3*) to the deterministic action of an agent already chosen. The recursive calls of *M* made for agents' choices in *algorithm 1* correspond in *algorithm 3* to the recursive calls of *M* for the environment's choices. With regard to the circuit shown in Figure 4, agents' choices in Woolridge's problem (now deterministic) are placed at the *OR* gates and environment (non-deterministic) choices in Wooldridge's problem are placed at the *AND* nodes. Therefore, the deterministic version of Woolridge's maintenance problem has been translated into the target problem, and, consequently, the target problem is also *PSPACE-complete*.

It has been demonstrated that the target problem is *PSPACE-complete* for an infinite number of iterations, *i*. Using the experience accumulated so far in this proof for *i* infinite (particularly useful is the expression of the problem in the circuit given above), and theorems 17.8 (especially its corollary 2) and 17.10 in [5], it should be possible to prove that the problem is $\Sigma_i P$*-complete* if the number of iterations, *i*, is finite.

5 Conclusion

This paper has verified that the complexity of a constraint model based exploration of simulation trajectories for proving the envelope of tendencies in the dynamics of a MAS-based simulation model is *PSPACE-complete* for an infinite number of iterations and has suggested that it is $\Sigma_i P$*-complete* for a finite number of iterations, *i*.

Proving the envelope of tendencies in simulation outputs is an alternative to traditional methods used for examining simulation outputs. The former allows elaborating more general conclusions than the latter.

As explained better in [8], constraint exploration of simulation trajectories brings closer the simulation and the logic programming communities. This paper contributes in making clearer a property of high interest to both of these communities, namely the computational complexity of a constraint exploration of simulation trajectories

Acknowledgements. The research reported here was funded by the CDCHT (the Council for Scientific, Humanistic and Technological Development) of the Universidad de Los Andes, Venezuela, under project I-524, by CONICIT (the Venezuelan Governmental Organisation for promoting Science), and by the Faculty of Management and Business, Manchester Metropolitan University.

References

1. Axtell, R., R. Axelrod, J. M. Epstein, and M. D. Cohen, "Aligning Simulation Models: A Case Study and Results", *Computational Mathematical Organization Theory,* 1(2), pp. 123-141, 1996.
2. Carley K., M. Prietula, and Z. Lin, "Design Versus Cognition: The Interaction of Agent Cognition and Organizational Design on Organizational Performance", *Journal of Artificial Societies and Social Simuation* 1(3), 1998 (accessible at: http://www.soc.surrey.ac.uk/JASSS/1/3/4.html).
3. Loveland, D. W., *Automated Theorem-proving: A Logical Basis*, North-Holland Pub., Amsterdam, 1978.

4. Moss, S., "Social Simulation Models and Reality: Three Approaches", *MAB's 98: Multi-agent Systems and Agent-Based Simulation*, Paris, 1998 (accessible at http://www.cpm.mmu.ac.uk/cpmrep35.html).
5. Papadimitriou, Christos, *Computational Complexity*, Addison-Wesley Publishing Company, California, USA, 1994.
6. Terán Oswaldo, Bruce Edmonds and Steve Wallis, "Mapping the Envelope of Social Simulation Trajectories", MABS2000 @ ICMAS-2000: The Second Workshop on Multi Agent Based Simulation, Boston, July 9, 2000. Published in: Moss, Scott and Paul Davidsson (Editors), *Multi Agent Based Simulation (MABS-2000), Lecture Notes in Artificial Intelligence, Vol. 1979*, Springer Verlag, Berlin
7. Terán Oswaldo, Bruce Edmonds and Steve Wallis, "Determining the Envelope of Emergent Agent Behaviour via Architectural Transformation", ATAL-2000: The Seventh International Workshop on Agent Theories, Architectures, and Languages, Boston, July 7-9, 2000. Published in: Castelfranchi, C. and Y. Lesperance (Editors), *Intelligent Agents VII. Agent Theories, Architectures, and Languages. Lecture Notes in Artificial Intelligence, Vol. 1986*, Springer-Verlag, Berlin
8. Terán Oswaldo, Bruce Edmonds and Steve Wallis, "Constraint Exploration and Envelope of Simulation Trajectories", First Workshop on Rule-Based Constraint Reasoning and Programming at the First International Conference on Computational Logic (CL2000), July 24-28, 2000, Imperial College, London, UK (this paper is accessible at: http://www.pst.informatik.uni-muenchen.de/personen/fruehwir/cl2000r.html; and at: http://arXiv.org/abs/cs/0007001)
9. Woolridge, Mike "The Computational Complexity of Agent Design Problems", in *Proceedings Fourth International Conference on MultiAgent Systems (ICMAS-2000)*, Boston, MA, USA, July 10-12, 2000, pp. 341-348.
10. Wos, L., *Automated Reasoning*: *Introduction and Applications*, Prentice Hall, London, 1984.
11. Zeigler, Bernard, *Theory of Modelling and Simulation*, Robert E. Krieger Publishing Company, Malabar, Fl, USA, 1976.