

Foreword

It's been almost ten years since Andr  ka, van Benthem and N  meti proved decidability of the guarded fragment of first order logic (see 'Modal Logics and Bounded Fragments of Predicate Logic' in *Journal of Philosophical Logic*, 1998, 27(3), or 'Back and Forth between Modal Logic and Classical Logic' in *Bulletin of the Interest Group in Pure and Applied Logics*, 1995, 3). Given how expressive guarded quantification is, this result gave logicians a powerful tool for proving decidability of many formalisms arising in computer science applications, and generated extensive research into guarded quantification in logics other than first order (for example, Erich Gr  del and Igor Walukiewicz proved decidability of the guarded fragment of fixed point logic in *Guarded Fixed Point Logic*, Proceedings LICS'99, 1999, pp. 45-54) and into various generalisations of guarded quantification. A wealth of new proof techniques has been developed as a result of this research. This workshop brings together work on new guarded fragments and on applications of guarded quantification in computer science.

The workshop is held as a part of the 16th European Summer School in Logic, Language and Information (Nancy, 2004). It contains a mixture of invited and contributed talks. The paper by Dietmar Berwanger and Achim Blumensath was originally published in E. Gr  del, W. Thomas and T. Wilke (Eds.): *Automata, Logics, and Infinite Games*, LNCS 2500, pp343-355, 2002, and is reproduced with the kind permission of Springer-Verlag. The paper by Ian Hodkinson was published in *Studia Logica* 72(2):185-197, 2002, and is reproduced with the kind permission of Kluwer Academic Publishers.

Natasha Alechina
(Workshop organiser)

Preliminary programme

9 August Introduction; talks by Johan van Benthem and Ulrike Sattler

10 August

Hans de Nivelle, Automated theorem proving and GF

Lilia Georgieva, Optimizing minimal Herbrand model generation procedures for guarded formulae, description and modal logics

11 August

Dirk Leinders (joint work with Jerzy Tyszkiewicz and Jan Van den Bussche), The semijoin algebra and the guarded fragment

Michael Minock, Managing database incompleteness with the guarded fragment

12 August

Dietmar Berwanger (joint work with Achim Blumensath), Automata for guarded fixed point logics

Antje Nowack, A guarded fragment for abstract state machines

13 August

Ian Hodkinson, Monodic packed fragment with equality is decidable

Ian Pratt, Counting quantifiers and stellar fragment

Table of Contents

Foreword	1
Preliminary program	2
Table of contents	3
Lilia Georgieva, <i>Optimizing minimal Herbrand model generation procedures for guarded formulae, description and modal logics</i>	4
Dirk Leinders, Jerzy Tyszkiewicz and Jan Van den Bussche, <i>The semijoin algebra and the guarded fragment</i>	16
Michael Minock, <i>Managing database incompleteness with the guarded fragment</i>	27
Antje Nowack, <i>A guarded fragment for abstract state machines</i>	46
Dietmar Berwanger and Achim Blumensath, <i>Automata for guarded fixed point logics</i>	66
Ian Pratt, <i>Counting quantifiers and stellar fragment</i>	80
Ian Hodkinson, <i>Monodic packed fragment with equality is decidable</i>	101

Optimising minimal Herbrand model generation procedures for guarded formulae, description and modal logics

Lilia Georgieva

Department of Computer Science, Heriot-Watt University
Edinburgh EH14 4AS, United Kingdom, lilia@macs.hw.ac.uk

Abstract. In this paper we develop efficient minimal Herbrand model generation procedures for decidable by hyperresolution syntactically defined clausal classes, clausal classes rendered by formulae in first-order logic, which contain guarded and loosely guarded formulae and clausal classes rendered by formulae in description and modal logics. The procedures use complement splitting and a restricted application of either model constraint propagation rule or local minimality test and consequently optimise known minimal Herbrand model generation procedures.

1 Introduction

Computing models is a task complementary to refutational theorem proving. Its importance is due to the potential application areas including disproving conjectures in classical theorem proving [11], natural language [5], deductive databases [10, 26] and model-based diagnosis [3].

Characterizing classes of inputs for which termination is guaranteed, defining appropriate formalisms for representing such models, and providing algorithms for working with the resulting model representations is recognized as a challenge in automated deduction [9, 20]. A theorem prover that terminates without finding a refutation proof and outputs a countermodel provides the user with information, suitable for subsequent optimisations.

This is the advantage of hyperresolution decision procedures for satisfiable clausal sets of range restricted clauses [7], i.e. for clauses whose negative literals contain all the variables of the clause. Provided that a hyperresolution derivation terminates, it generates Herbrand models for such classes. An example of a clausal class for which hyperresolution terminates is the class of Horn clauses. The reasoning process corresponds to the standard fixpoint iteration and Herbrand models are generated.

In general hyperresolution does not terminate on non-Horn clauses. There are, however, non-Horn clausal classes that can be decided by hyperresolution, e.g. clausal classes resulting from formulae in classical and non-classical logic [6, 16–18, 25], satisfiable syntactically defined clause classes which allow for the embedding of guarded formulae [14, 9, 19]. In these cases disjunctions resulting from non-Horn clauses are handled by the *splitting* rule, which introduces branching in the forward reasoning process. Splitting is crucial if atoms in disjunctions share variables. However, the *range restrictedness* of clauses ensures that the disjunctions always become ground and instansiated and thus the disjointness of variables is not critical before splitting.

In previous work [12, 14, 15] we have shown that hyperresolution generates Herbrand models for formulae in the guarded fragment of first order logic and syntactically defined clausal classes and have investigated the construction of Herbrand

models from the conclusions of any hyperresolution derivation. We have also analyzed the syntactical properties of the constructed Herbrand models not all of which are minimal. We have proposed three ways of refining hyperresolution so that it is a minimal model generation procedure which constructs only minimal Herbrand models for the considered clausal classes. The refinements are based on hyperresolution, (complement) splitting and either model constraint propagation or local minimality tests. In this paper we continue this line of investigation and show that the minimal Herbrand model generation procedures can be further optimised so that first the application of the model constraint propagation rule is limited to certain open branches only and secondly so that the minimality test uses a constraint which contains strictly fewer literals than the minimal Herbrand model generation procedure developed in [24]. Throughout we employ the approach of Hasegawa et. al [16] and adopt it to our framework.

The optimised minimal Herbrand model generation procedures described in this paper are relevant for multi-modal and description logics, decidable fragments of first-order logics, including the fragment GF1⁻ which is a subfragment of the guarded fragment and syntactically defined clausal classes, among which the clausal class $\mathcal{BU}$, which contains formulae in the guarded and the loosely guarded fragments and for which hyperresolution is a decision procedure and leads to automatical model generation.

The structure of the paper is as follows. Notational conventions and basic definitions are given in Section 2. In Section 3 we discuss approaches for minimal Herbrand model generation by hyperresolution. Optimising minimal Herbrand model generation

The calculus which is used to determine satisfiability of the clause sets considered in this paper is denoted by R^{hyp} . It uses the following expansion rules.

$$\begin{array}{ll} \textbf{Deduce:} & \frac{N}{N \cup \{C\}} \\ & \text{where } C \text{ is a resolvent or factor.} \end{array} \quad \begin{array}{ll} \textbf{Splitting:} & \frac{N \cup \{C_1 \vee C_2\}}{N \cup \{C_1\} \quad | \quad N \cup \{C_2\}} \\ & \text{where } C_1 \text{ and } C_2 \text{ are variable disjoint.} \end{array}$$

The inference rules for hyperresolution and factoring are:

$$\textbf{Hyperresolution:} \quad \frac{C_1 \vee A_1 \quad \dots \quad C_n \vee A_n \quad \neg A_{n+1} \vee \dots \vee \neg A_{2n} \vee D}{(C_1 \vee \dots \vee C_n \vee D)\sigma}$$

where (i) σ is the most general unifier such that $A_i\sigma = A_{n+i}\sigma$ for every i , $1 \leq i \leq n$, and (ii) $C_i \vee A_i$ and D are positive clauses, for every i , $1 \leq i \leq n$. The rightmost premise in the rule is referred to as the *negative* premise and all other premises are referred to as *positive* premises.

$$\textbf{Factoring:} \quad \frac{C \vee A_1 \vee A_2}{(C \vee A_1)\sigma}$$

where σ is the most general unifier of A_1 and A_2 .

Factoring is not required for the completeness of our decision procedure, but it helps avoiding applications of splitting to clauses containing duplicate literals.

A *derivation* in R^{hyp} from a set of clauses N is a finitely branching tree T with root N . The tree is constructed by applications of the expansion rules to the leaves so that factoring, splitting and hyperresolution are applied in this order. It is taken for granted that at no stage in the derivation the clause sets contain duplicate clauses.

Any path $N(= N_0), N_1, \dots$ in a derivation T is called a *closed branch* in T if the clause set $\bigcup_j N_j$ contains the empty clause, otherwise it is called an *open branch*. We call a branch B in a derivation tree *complete* iff no new successor nodes can be added to the endpoint of B by $\mathbf{R}^{\text{hyp}}$. A derivation T is a *refutation* if every path $N(= N_0), N_1, \dots$ in it is a closed branch. A derivation T is a *refutation* if for every path $N(= N_0), N_1, \dots$ the clause set $\bigcup_j N_j$ contains the empty clause. A derivation T from N is called *fair* if for any path $N(= N_0), N_1, \dots$ in T , with *limit* $N_\infty = \bigcup_j \bigcap_{k \geq j} N_k$, it is the case that each clause C that can be deduced from non-redundant premises in N_∞ is contained in some N_j . If $N(= N_0), N_1, \dots$ is a path with limit N_∞ , then we say that N_∞ is saturated up to redundancy.

Theorem 1 ([2]). *Let T be a fair theorem proving derivation from a set N of clauses. Then: (i) N is satisfiable if and only if there exists a path in T with limit N_∞ such that N_∞ is satisfiable. (ii) N is unsatisfiable if and only if for every path $N(= N_0), N_1, \dots$ the clause set $\bigcup_j N_j$ contains the empty clause.*

We denote the set of positive ground unit clauses in a branch B of T by $\mathcal{B}$. Then $\mathcal{B}$ is a Herbrand model for N . We say that $\mathcal{B}$ is a (partial) Herbrand model for N , *represented* by the open branch B . Analogously, $\{\overline{\mathcal{B}}\}$ stands for $\bigcup\{\neg A_i\}, 1 \leq i \leq n$, where the atoms A_i are all positive ground unit atoms such that $\mathcal{B} \models A_i, 1 \leq i \leq n$.

2 Minimal Herbrand model generation by hyperresolution

In this section we provide an outline of the known approaches for minimal Herbrand model generation.

The approach of Bry et al. [7]. A crucial feature of the approach is the *complement splitting rule*. The complement splitting rule prevents the generation of duplicate models and is applied to ground disjunctions. The rule is called *folding down* in [21] and was first implemented in SATCHMO [23]. The rule is defined as follows.

Complement splitting

$$\frac{N \cup \{C \vee D\}}{N \cup \{C, \neg D\} \mid N \cup \{D\}}$$

where $\neg D = \neg A_1 \wedge \dots \neg A_n$, if $D = A_1 \vee \dots \vee A_n$. Complement splitting can also be seen as a combination of the cut rule (applied to D), clause reduction (replacing $C \vee D$ by C in the presence of $\neg D$), and subsumption deletion (removing $C \vee D$ in the presence of D). calculus developed in [12]. Hyperresolution combined with complement splitting is sound and complete [7]. The use of complement splitting ensures that whenever the rule is applied to a clause $C \vee D$, the two branches consider different valuations for D . However, the use of complement splitting alone on clausal classes for which $\mathbf{R}^{\text{hyp}}$ terminates does not ensure that only minimal Herbrand models are generated. In addition, the branches of the derivation tree are investigated in a particular order, namely, the branch for $N \cup \{C, \neg D\}$ is to be investigated before the branch for $N \cup \{D\}$ and the derivation tree is to be built depth-first.

In order to ensure that whenever a complete and open branch B is obtained, none of the remaining branches will generate the same Herbrand model or a model, which is larger for the branch B the procedure proceed as follows.

Let $A_1, \dots, A_n$ be all the positive ground unit clauses in $\mathcal{B}$. Then the clause $\neg A_1 \vee \dots \vee \neg A_n$ is added to the clausal sets on all the branches, which still need to be investigated and consequently in case another branch containing $A_1, \dots, A_n$ is encountered, this branch will be immediately closed by a hyperresolution inference step. A derivation tree T constructed by R^{hyp} , the rule above, and complement splitting satisfies the following properties: (i) Inference rules are applied only to the left-most branch of T , which is not complete. (ii) Suppose $N_1, \dots, N_l$ are the clause sets labeling the leaves of branches T , which are enumerated left-to-right, such that no further inference rules can be applied to N_1 and $A_1, \dots, A_n$ are the unit clauses in N_1 . Then T is replaced by T' where T' is identical to T , except that the leaves $N_2, \dots, N_l$ are replaced by $N_2 \cup \{\neg A_1 \vee \dots \vee \neg A_n\}, \dots, N_l \cup \{\neg A_1 \vee \dots \vee \neg A_n\}$, respectively. The disjunction of negated ground atoms $\neg A_1 \vee \dots \vee \neg A_n$ is called a *model constraint*. The act of replacing $N_2, \dots, N_l$ by $N_2 \cup \{\neg A_1 \vee \dots \vee \neg A_n\}, \dots, N_l \cup \{\neg A_1 \vee \dots \vee \neg A_n\}$ is called *model constraint propagation*.

Provided that the open branches of the derivation tree T , constructed by R^{hyp} do not contain negative literals, whenever a branch is extended by a R^{hyp} step all positive premises are true in the Herbrand interpretation H , defined by the ground literals that are already present in the branch. If H is a minimal model of N there is an open branch B in T , such that $\mathcal{B}$ coincides with the ground atoms in H . Complement splitting ensures that for any two open branches B_i and B_j of the derivation tree $\mathcal{B}_i \not\subseteq \mathcal{B}_j$. The open branches $B_1, \dots, B_i, \dots, B_r$ of T can be enumerated left-to-right so that the set of ground unit clauses in the leftmost branch, i.e. B_1 represents a minimal model of N . Minimal models for range-restricted clauses, on which hyperresolution terminates are preserved under complement splitting [7].

Generalisation of the approach of Bry et al. [12] This approach uses (complement) splitting and model constraint propagation. In contrast to the approach by Bry et al. [7] the parameters branch selection and complement splitting are optional and arbitrary branch selection is possible when constructing a hyperresolution derivation. The model constraint propagation rule is applied as before to a branch $\mathcal{B}$ but is defined differently.

Model constraint propagation: If $\mathcal{B} = A_1, \dots, A_n$ is the Herbrand model of an open, non-finished branch B , which is complete with respect to R^{hyp} , then the rule adds the clause $\overline{\mathcal{B}} = \neg A_1 \vee \dots \vee \neg A_n$ to all leaves of branches in the derivation tree, which are incomplete with respect to $R_{\text{min}}^{\text{hyp}}$ or are marked finished.

Once the rule has been applied, the branch B is marked *finished*. A branch, on which the rule has not been applied is marked as *non-finished*.

As before the clause $\overline{\mathcal{B}}$ is called a *model constraint*. The model constraint propagation rule ensures that no model $\mathcal{B}'$, which is a superset of $\mathcal{B}$ is computed. Because $\mathcal{B}$ may not be a minimal model, it is not enough for the model constraint propagation rule to add a model constraint to branches, which are incomplete with respect to R^{hyp} (i.e. branches, which can be further expanded by the application of an R^{hyp} inference rule). A model constraint is also added to branches, which are complete with respect to R^{hyp} . This ensures that for any branch B' , which would otherwise generate a non-minimal model $\mathcal{B}'$, eventually the model constraint $\overline{\mathcal{B}}$ of a branch B

with $\mathcal{B} \subset \mathcal{B}'$ is propagated to B' . In this case, adding the model constraint $\overline{\mathcal{B}}$ to the leaf of B' changes the status of B' from complete to incomplete, and it is possible to perform a single additional hyperresolution inference, which closes B' . If $\mathcal{B}$ is the set of positive ground unit clauses in the limit of an open branch in a derivation tree from N , $\mathcal{B}$ forms a minimal Herbrand model for N . The procedure generates only minimal Herbrand models for N and it generates all minimal Herbrand models for N only once.

Procedural minimal Herbrand model generation [24] An alternative way of reducing the space requirement of minimal Herbrand model generation for range restricted clauses, clauses rendered by decidable fragments of first-order logic, and syntactically defined clause sets which can be decided by hyperresolution is by following the approach of Niemelä [24] which has been applied and refined in [1, 4]. The approach is based on the following observation.

Given a (finite) set H of positive ground atoms (or unit clauses) define: $\neg H = \{\neg A \mid A \in H\}$ and $\overline{H} = \bigvee_{A \in H} \neg A$. Let N be a set of clauses and U be the set of all atoms over the Herbrand universe of N . Let H be a finite Herbrand model of N . Then H is a minimal Herbrand model of N if and only if $MMT(N, H) = N \cup \neg(U - H) \cup \{\overline{H}\}$ is unsatisfiable. This minimality test is called *groundedness test*.

3 Optimising minimal Herbrand model generation

An issue from our perspective is that all minimal Herbrand model generation approaches described in Section 2 perform unnecessary minimality tests for models that are already minimal. A method which uses *branching lemmas* and is proposed by Hasegawa et al. [16] which we study next enhances their efficiency.

Branching lemmas correspond to application of factorization to minimal model generation [21] and their use is justified by the notion of *proof commitment*.

Proof commitment between two Herbrand models H and H' occurs when a branch extended below a node which contains a literal in H' is closed by a negation of a literal in H . If no proof commitment occurs between a newly generated model and any other model H' that has been previously derived on an open branch of the derivation tree the model H' is guaranteed to be minimal and consequently the minimality test may be omitted.

The method also employs *branching assumptions* which as defined in [16] correspond to an application of the complement splitting rule. As previously stated complement splitting rule guarantees that the leftmost branch of the derivation tree is always minimal [7]. However, Herbrand models generated on other branches are not necessarily minimal. In order to distinguish between branches which are likely to lead to generation of non-minimal Herbrand models, using this approach open branches are divided into *safe* and *warned* [16].

Definition 1. Let L_i be a ground literal in a ground disjunction $L_1 \vee \dots \vee L_n$. The literal L_i is called a *committing disjunct* if a branch expanded below L_i is closed by the branching assumption $[\neg L_j]$ of some right sibling disjunct L_j ($i + 1 \leq j \leq n$). Every right sibling disjunct L_k is called a *committing disjunct from* L_i .

Definition 2. If $L_i \in L_1 \vee \dots \vee L_n$ is not a committing disjunct, $\neg L_i$ is added to to every right sibling disjunct L_j ($i + 1 \leq j \leq n$). $\neg L_i$ is called a branching lemma and is denoted by $[[L_i]]$.

Let H be a Herbrand model obtained in a model generation tree. If the model contains a committed disjunct L_j in $L_1 \vee \dots \vee L_n$ each committing disjunct appearing as a left sibling of L_j is said to be committing disjunct relevant to H . The model H is called a safe model if it contains no committed disjuncts. If H contains a committing disjunct H is called a warned model.

Branching lemmas lead to pruning branches that lead to nonminimal models and to reducing the minimality test [16].

Theorem 2. If a generated on an open branch Herbrand model is safe, H is assured to be minimal and no minimality test is required.

Proof. Assume that a generated model is safe. Then by definition $L_j \in H$ is not a committed disjunct and for each left sibling disjunct L_k ($1 \leq k \leq j - 1$), a model H' which contains L_k satisfies the following: (i) $L_j \notin H'$ under the branching assumption $[\neg L_j]$, (ii) $L_k \notin H$ under the branching assumption $[L_j]$, and (iii) $L_k \notin H$ under the branching lemma $[[\neg L_k]]$. Thus H is not subsumed by H' , i.e. H is a minimal Herbrand model.

If an obtained model H is warned it is not necessary to perform a minimality test on H against all models that have been obtained. Minimality test can be performed only against such models that contain committing disjuncts relevant to H . This is called a restricted minimality test.

Theorem 3. Let T be a derivation tree constructed by hyperresolution from a set of clauses N and let N_1 and N_2 be sibling nodes in T , let L_i be a literal in N_i and let T_i be a subproof tree below N_i ($i=1,2$). If N_2 has a descendant node N_3 labeled with L_1 , then for each model H through a subproof tree T_3 below N_3 , there exists a model H' through T_1 such that $H' \subseteq H$

A factorization dependency relationship on a proof tree is a strict partial ordering $\prec$ relating sibling nodes in the proof tree. A relation $N_1 \prec N_2$ means that searching for minimal model below N_2 is committed to that below N_1 .

Given a proof tree and a factorization dependency relationship $\prec$ on T first select a node N_3 labeled with L_1 and another node N_1 labeled with the same literal such that: (i) N_3 is a descendant of a node N_2 which is a sibling of N_1 , and (ii) $N_2 \not\prec N_1$. Next, close the branch extended to N_3 and modify $\prec$ by adding a relation $N_1 \prec N_2$, then forming the transitive closure of the relation. The proof of N_3 is committed to N_1 . Minimal model generation with factorization is complete, i.e. if a minimal model H of N is constructed by model generation, then H can be constructed by model generation with factorization. Given a satisfiable set of range-restricted clauses N and a derivation tree T constructed by model generation with factorization, any minimal model of N is represented by an open branch of T .

Theorem 4. Let N be a set of clauses and T be a derivation tree from N by model generation with factorization. Let N_1 and N_2 be sibling nodes in T , let T_i be a subproof tree below N_i and H_i be a model through T_i . If $N_2 \not\prec N_1$ then $H_1 \not\subseteq H_2$.

Theorem 5. *Let T be a derivation tree constructed by a depth first left first search and model generation with factorization. If a model H_2 is generated on an open branch to the right of a model H_1 in T then $H_2 \not\subseteq H_1$.*

Corollary 1. *Let N be a satisfiable set of range restricted clauses and T be a derivation tree constructed by model generation with branching assumptions, branching lemmas and restricted minimality test. The open branches of T correspond to minimal Herbrand models of N .*

Let N be a set of range-restricted clauses, T be a proof tree of N by model generation with factorization, and H be a Herbrand model of N . For each literal $L \in H$, let N_L denote a node labeled with L on the branch that corresponds to H . Let H_ρ be the set that satisfies the following condition: for every $L \in H_\rho$ there exists a node N such that $N_L \prec N$. Note that H_ρ is a set of committed disjuncts in H . The shortened negative clause C_{H_ρ} consists of the negations of all literals in H_ρ .

Theorem 6. *Let N be a set of clauses. H is a minimal Herbrand model of N if and only if $N_{H_\rho} = N \cup \overline{H} \cup \{C_{H_\rho}\}$ is unsatisfiable, where $\overline{H} = \bigcup_i \{\neg L_i\} \notin H$.*

4 Hyperresolution as a minimal Herbrand model generation procedure

In this section we list the clausal classes and fragments for which the minimal Herbrand model generation approaches and optimisations described above are applicable.

The usability of hyperresolution as a decision procedure and a minimal Herbrand model generation procedure has been studied in the context of clausal classes rendered by modal and description logics and syntactically defined clause classes. An example is the clausal class, associated with the encoding in clausal form of the extended multi-modal logic $\mathbf{K}_{(m)}(\cap, \cup, \smile)$ [8, 18] and the corresponding reduct of the description logic $\mathcal{ALB}$ without the top role and role negation [17].

The encoding of the multi-modal logic $\mathbf{K}_{(m)}(\cap, \cup, \smile)$ in clausal form consists of the clauses given in Fig. 1 [8], where $\mathcal{P}(s)$ stands for a literal in $\{P_i(s), Q_\varphi(s)\}_{i,\phi}$ and $\mathcal{R}(s, t)$ stands for a literal in $\{R_j(s, t), R_j(t, s), Q_\alpha^{p/n}(s, t), Q_\alpha^{p/n}(t, s)\}_{j,\alpha}$. The symbols Q_α^p and Q_α^n are associated with positive and negative occurrences of a relational subformula α and Q_φ denotes an occurrence of a predicate symbol. The corresponding class of clauses rendered by formulae in $\mathcal{ALB}$ is known as the *class of DL-clauses* [17]. All functional terms in these clause classes are covering and unary. All literals contain either one or two variables, and a literal is ground, whenever it contains a constant symbol.

A clausal class, which is decidable by ordered resolution is the clausal class $\mathbf{DL}^*$ [8]. It consists of clauses which have the following properties: (i) all literals are unary or binary; (ii) there is no nesting of function symbols; (iii) every functional term in C contains all the variables of C ; (iv) every binary literal even if it has no functional terms contains all variables of C . Syntactically defined clausal classes that are decidable by hyperresolution are investigated in [14, 9, 19] and include the

$\mathcal{P}(\bar{a})$	
$\neg Q_\varphi(x) \vee \neg P_i(x)$	if $\varphi = \neg p_i$
$\neg Q_\varphi(x) \vee \neg \mathcal{R}(x, y)[\vee \mathcal{P}(y)]$	if $\varphi = [\alpha]\phi$ $[\varphi = [\alpha]\perp]$
$\left. \begin{array}{l} \neg Q_\varphi(x) \vee \mathcal{R}(x, f(x)) \\ \neg Q_\varphi(x) \vee \mathcal{P}(f(x)) \end{array} \right\}$	if $\varphi = \langle \alpha \rangle \phi$
$\left. \begin{array}{l} \neg Q_\varphi(x) \vee \mathcal{P}(x) \\ \neg Q_\varphi(x) \vee \mathcal{P}(x) \end{array} \right\}$	if $\varphi = \phi \wedge \psi$
$\neg Q_\varphi(x) \vee \mathcal{P}(x) \vee \mathcal{P}(x)$	if $\varphi = \phi \vee \psi$
$\neg Q_\alpha^p(x, y) \vee \mathcal{R}(x, y)[\vee \mathcal{R}(x, y)]$	if $\alpha = \beta_1 \wedge [\vee]\beta_2$ has pos. polarity
$Q_\alpha^n(x, y) \vee \neg \mathcal{R}(x, y)[\vee \neg \mathcal{R}(x, y)]$	if $\alpha = \beta_1 \wedge [\vee]\beta_2$ has neg. polarity

Fig. 1. Schematic clausal form for $K_{(m)}(\cap, \cup, \neg)$.

classes $\mathcal{PVD}$, $\mathcal{KPOD}$ and $\mathcal{BU}$. The approaches are also applicable to the first-order fragment GF1^- [22] and its extensions on first-order level [15].

Grouping is a global condition of predicate symbols in GF1^- and all occurrences of a predicate symbol in a guard position must satisfy the given grouping of the predicate symbol. Each predicate symbol P is uniquely associated with a pair (i, j) of non-negative integers, such that if the arity of P is n then $i + j = n$. This pair is called the *grouping* of the predicate symbol. Sometimes the grouping (i, j) of a predicate symbol P will be made explicit by writing $P^{(i,j)}$. The notion of grouping is extended to literals in the following way. A literal L is said to satisfy the *grouping condition with respect to the sequences $\bar{x}$ and $\bar{s}$* , if $L = (\neg)P^{(i,j)}(\bar{x}, \bar{s})$ or $L = (\neg)P^{(j,i)}(\bar{s}, \bar{x})$, where $\bar{x}$ is an i -sequence of variables, and $\bar{s}$ is either a j -sequence of variables disjoint from $\bar{x}$ or a j -sequence of terms of the form $f(\bar{z})$ where $\bar{z} \subseteq \bar{x}$, and $\bar{x}$ is non-empty.

The set of formulae in GF1^- is defined to be the smallest set satisfying the following conditions:

- (i) $\top$ and $\perp$ are GF1^- formulae,
- (ii) if P is an n -ary predicate symbol and $\bar{x}$ is a sequence of n variables, then $P(\bar{x})$ is a GF1^- formula,
- (iii) if ϕ and ψ are GF1^- formulae then so are $\neg\phi$, $\phi \wedge \psi$, $\phi \vee \psi$, and
- (iv) if $\phi(\bar{y})$ is a GF1^- formula, $G^{(i,j)}$ is a predicate symbol with grouping (i, j) , and $\bar{x}, \bar{y}$ are non-empty variable sequences of length i and j with no variables in common, then the following formulae are GF1^- formulae.

$$\begin{array}{ll} \exists \bar{y}(G^{(i,j)}(\bar{x}, \bar{y}) \wedge \phi(\bar{y})) & \forall \bar{y}(G^{(i,j)}(\bar{x}, \bar{y}) \rightarrow \phi(\bar{y})) \\ \exists \bar{x}(G^{(i,j)}(\bar{x}, \bar{y}) \wedge \phi(\bar{x})) & \forall \bar{x}(G^{(i,j)}(\bar{x}, \bar{y}) \rightarrow \phi(\bar{x})) \end{array}$$

Note how the role of $\bar{x}$ and $\bar{y}$ may be interchanged in the guard. Occurrences of atoms in guard positions such that (iv) is satisfied are said to *satisfy the grouping restriction*.

Examples of GF1^- formulae are the following.

$$\begin{array}{l} q(x, y) \wedge \exists x, y(r_1(z, z, x, y) \wedge p(x, y)), \quad p(x, y) \wedge p(y, x), \\ \forall xy(r_2(x, y, z) \rightarrow (p(x, y) \wedge \exists z(r_2(x, y, z) \wedge q(z, z)))) \end{array}$$

The grouping of the predicate symbols r_1 and r_2 is $(2, 2)$ and $(2, 1)$, respectively, while for the remaining predicate symbols the grouping is immaterial because they are not on guard positions. All free variables in GF1^- formulae are implicitly existentially quantified. Note that making the existential quantification of x and y explicit in the second example would result in a formula which is not in GF1^- (it would still be a guarded formula though). Other examples of guarded formulae which do not belong to GF1^- are $\forall xy(q(x, x, y) \rightarrow \perp)$ and $\forall xy(p(x, y) \rightarrow \exists zp(y, z))$. The first formula is not in GF1^- .

The subformula $\perp$ guarded by $q(x, x, y)$ in the first formula contains no free variables and therefore the condition that the bounded variables x and y occur in the guarded formula is not observed. The second formula is closed. The formula $\forall xy(p(x, y, z) \rightarrow \exists zp(y, x, z) \wedge r(z))$ also does not belong to GF1^- .

The language of $\mathcal{BU}$ is that of first-order clausal logic. Additionally, each predicate symbol is associated with a unique grouping. Repetitions of variables and terms in any of the sequences are allowed.

Furthermore, an acyclic relation $\succ_d$, called an *acyclic dependency relation*, is defined over the predicate symbols. Let $\succ_d^+$ be the transitive closure of $\succ_d$. Then $\succ_d^+$ is an ordering on predicate symbols. This ordering extends to atoms, literals and clauses by the following definitions. Given two literals $L_1 = (\neg)P_1(\bar{s})$ and $L_2 = (\neg)P_2(\bar{t})$, $L_1 \succ_D L_2$ iff $P_1 \succ_d^+ P_2$. The multiset extension of the ordering $\succ_D$, also denoted by $\succ_D$, defines an ordering on ground clauses. The acyclicity of $\succ_d$ implies that $\succ_D$ is also acyclic.

Given a finite signature Σ such that (i) any predicate symbol has a unique grouping, and (ii) there is an acyclic dependency relationship $\succ_d$ on the predicate symbols in Σ , we define the *class* $\mathcal{BU}$ of clausal sets over Σ as follows. A clausal set N belongs to $\mathcal{BU}$ if any clause C in N satisfies one of the three conditions below as well as the following. If C is a non-ground and non-positive clause then C is required to contain a strictly $\succ_D$ -maximal literal, which is negative and non-functional. This literal is called the *main* literal of the clause. The predicate symbol P of the main literal must either have the grouping $(0, i)$ or $(i, 0)$, where i is the arity of P .

Condition 1: C is a non-positive, non-ground and non-functional clause and the following is true.

- (a) The union of the variables of the negative part can be partitioned into two disjoint subsets X and Y , at least one of which is non-empty.
- (b) For every literal L in C , either the variables of L are (i) subsets of X , or (ii) subsets of Y , or (iii) there are non-empty sequences $\bar{x}, \bar{y}$, such that $\bar{x} \subseteq X$, $\bar{y} \subseteq Y$ and L satisfies the grouping condition with respect to $\bar{x}$ and $\bar{y}$.
- (c) Either the main literal contains all the variables of the clause, or it contains all the variables from one of the sets X and Y , and there is a negative literal L whose arguments satisfy (b.iii) and which contains all the variables from Y if the main literal contains all the variables of X , or all the variables from X if the main literal contains all the variables of Y .

Condition 2: C is a non-positive and non-ground functional clause and the following is true.

- (a) The main literal of C contains all the variables of C .

- (b) Every other literal L in C satisfies the grouping condition with respect to two disjoint sequences of variables $\bar{x}$ and $\bar{y}$, or with respect to two sequences $\bar{x}$ and $\overline{f_{\bar{x}}(\bar{u}_i)}$, where $\bar{x}$ is a sequence of variables and $\overline{f_{\bar{x}}(\bar{u}_i)}$ is a sequence of terms $f_i(\bar{u}_i)$ such that $\bar{u}_i \subseteq \bar{x}$.

Condition 3: C is a positive ground unit clause, its arguments are constants and its predicate symbol has grouping $(0, i)$ or $(i, 0)$.

The following are examples of $\mathcal{BU}$ clause sets:

$$\begin{aligned} & \{R(a, b), P(x, f(x, y)) \vee \neg R(x, y)\} \\ & \{P(a, b), \neg P(x, y) \vee Q(x, x, y, f(x, x))\} \\ & \{ Q_3(a, a_1, a_2), \\ & \quad \neg Q_3(x, y, z) \vee \neg Q_2(x) \vee P(x, x, y, z), \\ & \quad \neg Q_3(x, y, z) \vee Q_2(x) \vee Q_1(y, y, z) \vee Q_0(z, y), \\ & \quad \neg Q_2(x) \vee \neg P(x, x', y', z') \vee Q_1(x', y', z') \vee Q_0(x, x'), \\ & \quad \neg Q_1(x', y', z') \vee Q_0(y', z'), \\ & \quad \neg Q_0(y', z') \vee P(y', y', z', g(z', y')) \} \end{aligned}$$

Under the assumptions that (i) $R \succ_d P$, (ii) R has grouping $(0, 2)$ and P has grouping $(1, 1)$ the first set belongs to $\mathcal{BU}$. The second set belongs to $\mathcal{BU}$ provided $P \succ_d Q$, the grouping associated with the predicate symbol P is either $(0, 2)$ or $(2, 0)$, and the grouping associated with the predicate symbol Q is $(3, 1)$. For the third set to belong to $\mathcal{BU}$ the ordering among the predicate symbols must be $Q_3 \succ_d Q_2 \succ_d Q_1 \succ_d Q_0 \succ_d P$, P must have the grouping $(3, 1)$, Q_3 the grouping $(0, 3)$ or $(3, 0)$, Q_2 the grouping $(0, 1)$ or $(1, 0)$, any grouping (i, j) such that $i + j = 3$ can be associated with Q_1 , and the grouping $(0, 2)$ or $(2, 0)$ must be associated with Q_0 . The following clause sets do not belong to $\mathcal{BU}$.

$$\begin{aligned} & \{R(a, b), P(x, f(x, y), y) \vee \neg R(x, y)\} \\ & \{P(a), \neg P(x) \vee P(f(x))\} \end{aligned}$$

The arguments of the atom $P(x, f(x, y), y)$ in the first clause set cannot be partitioned into two disjoint sequences of variables and terms so that they satisfy the grouping condition. The second clause in the second set does not contain a strictly maximal negative literal, with respect to any acyclic dependency relation. In general, this excludes clauses where the predicate symbol of the main literal occurs both positively and negatively. Thus the transitivity clause and the symmetry clause do not belong to any clausal set in $\mathcal{BU}$. Also the reflexivity clause and the seriality clause are excluded from $\mathcal{BU}$ clause sets, because every non-ground clause must contain a negative main literal. Thus, other than irreflexivity $\neg R(x, x)$ and properties which can be expressed by negative unit clause, for example, seriality of the complement of a relation $\neg R(x, f(x))$, none of the standard properties of relations can be formulated in $\mathcal{BU}$.

A main motivation for studying classes like GF1^- and $\mathcal{BU}$ is that a variety of expressive modal and description logics can be embedded into them. Expressive modal and description logics have found applications in such varied areas as, for example, verification, programme analysis, knowledge representation, deductive data bases and the semantic web.

5 Conclusion

In this paper we showed that minimal Herbrand model generation procedures for decidable by hyperresolution clausal classes can be optimised. The optimisations are achieved by restricted applications of either model constraint propagation rule or local minimality test and apply the approach of Hasegawa et al. [16] to methods for minimal Herbrand model generation developed in [7, 13, 24].

References

1. C. Aravindan and P. Baumgartner. Theorem proving techniques for view deletion in databases. *Journal of Symbolic Computation*, 29(2):119–147, 2000.
2. L. Bachmair, H. Ganzinger, and U. Waldmann. Superposition with simplification as a decision procedure for the monadic class with equality. In G. Gottlob, A. Leitsch, and D. Mundici, editors, *Proceedings of KGC'93*, volume 713 of *Lecture Notes in Computer Science*, pages 83–96. Springer, 1993.
3. P. Baumgartner, P. Fröhlich, U. Furbach, and W. Nejdl. Semantically guided theorem proving for diagnosis applications. In A. R. Ralescu and J. G. Shanahan, editors, *Proceedings of International Joint Conference Artificial Intelligence*, pages 460–465. Morgan Kaufmann, 1997.
4. P. Baumgartner, J. Horton, and B. Spencer. Merge path improvements for minimal model hyper tableaux. In H. M. de Swart, editor, *Proceedings of Analytic Tableaux and Related Methods*, volume 1617 of *Lecture Notes in Artificial Intelligence*, pages 60–76. Springer, 1999.
5. P. Blackburn, J. Bos, M. Kohlhase, and H. de Nivelle. Inference and computational semantics. In *Proceedings of International Workshop on Computational Semantics*. Tilburg, The Netherlands, 1999.
6. F. Bry and S. Torge. A deduction method complete for refutation and finite satisfiability. In J. Dix, F. del Cerro, and U. Furbach, editors, *Proceedings of Journées Européennes sur la Logique en Intelligence Artificielle*, volume 1498 of *Lecture Notes in Artificial Intelligence*, pages 1–17. Springer, 1998.
7. F. Bry and A. Yahya. Positive unit hyperresolution tableaux for minimal model generation. *Journal of Automated Reasoning*, 25(1):35–82, 2000.
8. H. de Nivelle, R. A. Schmidt, and U. Hustadt. Resolution-based methods for modal logics. *Logic Journal of IGPL*, 8(3):265–292, 2000.
9. C. Fermüller, A. Leitsch, U. Hustadt, and T. Tammet. Resolution decision procedures. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 25, pages 1791–1849. Elsevier, 2001.
10. J. A. Fernandez and J. Minker. Bottom-up evaluation of hierarchical disjunctive deductive databases. In K. Furukawa, editor, *Proceedings of International Conference on Logic Programming*, pages 660–675. MIT Press, 1991.
11. U. Furbach, M. Khn, and F. Stolzenburg. Model-guided proof debugging. Technical Report RR-6/98, Universität Koblenz-Landau, Institut für Informatik, 1998.
12. L. Georgieva, U. Hustadt, and R. A. Schmidt. Computational space efficiency and minimal model generation for guarded formulae. In *Proceedings of Logic Programming and Non-monotonic Reasoning*, volume 2250 of *Lecture Notes in Artificial Intelligence*, pages 85–99. Springer, 2001.
13. L. Georgieva, U. Hustadt, and R. A. Schmidt. Computational space efficiency and minimal model generation for guarded formulae. In R. Nieuwenhuis and A. Voronkov, editors, *Proceedings of Logic Programming and Automated Reasoning*, volume 2250 of *Lecture Notes in Artificial Intelligence*, pages 85–99. Springer, 2001.
14. L. Georgieva, U. Hustadt, and R. A. Schmidt. A new clausal class decidable by hyperresolution. In A. Voronkov, editor, *Proceedings of Conference on Automated Deduction*, volume 2392 of *Lecture Notes in Artificial Intelligence*, pages 260–274. Springer, 2002.
15. L. Georgieva, U. Hustadt, and R. A. Schmidt. Hyperresolution for guarded formulae. To appear in *Journal of Symbolic Computation*, 2004.
16. Hasegawa, H. R., Fujita, and M. Koshimura. Efficient minimal model generation using branching lemmas. In C. Kirchner and H. Kirchner, editors, *Proceedings of Conference on Automated Deduction*, volume 1421 of *Lecture Notes in Artificial Intelligence*, pages 184–199. Springer, 2000.
17. U. Hustadt and R. A. Schmidt. Issues of decidability for description logics in the framework of resolution. In R. Caferra and G. Salzer, editors, *Automated Deduction in Classical and Non-Classical Logics*, volume 1761 of *Lecture Notes in Artificial Intelligence*, pages 191–205. Springer, 2000.

18. U. Hustadt and R. A. Schmidt. Using resolution for testing modal satisfiability and building models. *Journal of Automated Reasoning*, 28(2):205–232, 2002.
19. A. Leitsch. Deciding clause classes by semantic clash resolution. *Fundamenta Informaticae*, 18:163–182, 1993.
20. A. Leitsch, T. Tammet, and N. Zamov. *Resolution methods for the decision problem*. Springer, 1993.
21. R. Letz, K. Mayr, and Ch. Goller. Controlled integration of the cut rule into connection tableau calculi. *Journal of Automated Reasoning*, 13(3):297–337, 1994.
22. C. Lutz, U. Sattler, and S. Tobies. A suggestion of an n -ary description logic. In *Proceedings of Description Logics Workshop*, pages 81–85, 1999.
23. R. Manthey and F. Bry. SATCHMO: A theorem prover implemented in prolog. In E. Overbeek and R. Lusk, editors, *Proceedings of International Conference on Automated Deduction*, volume 310 of *Lecture Notes in Computer Science*, pages 415–434. Springer, 1988.
24. I. Niemelä. A tableau calculus for minimal model reasoning. In *Proceedings of Analytic Tableaux and Related Methods*, volume 1071 of *Lecture Notes in Artificial Intelligence*, pages 278–294. Springer, 1996.
25. S. Torge. *Überprüfung der Erfüllbarkeit im Endlichen: Ein Verfahren und seine Anwendung*. PhD thesis, Herbert Utz Verlag, München, 1998.
26. A. Yahya. Minimal model generation for refined answering of generalized queries in disjunctive deductive databases. In J. Dix, U. Furbach, and A. Nerode, editors, *Proceedings of Logic Programming and Non-monotonic Reasoning*, volume 1265 of *Lecture Notes in Computer Science*, pages 325–341. Springer, 1997.

The semijoin algebra and the guarded fragment

Dirk Leinders
Jerzy Tyszkiewicz *
Jan Van den Bussche

Abstract

The semijoin algebra is the variant of the relational algebra obtained by replacing the join operator by the semijoin operator. We discuss some interesting connections between the semijoin algebra and the guarded fragment of first-order logic. We also provide an Ehrenfeucht-Fraïssé game, characterizing the discerning power of the semijoin algebra. This game gives a method for showing that certain queries are not expressible in the semijoin algebra.

1 Introduction

Semijoins are very important in the field of database query processing. While computing project-join queries in general is NP-complete in the size of the query and the database, this can be done in polynomial time when the database schema is acyclic [12], a property known to be equivalent to the existence of a semijoin program [3]. Semijoins are often used as part of a query pre-processing phase where dangling tuples are eliminated. Another interesting property is that the size of a relation resulting from a semijoin is always linear in the size of the input. Therefore, a query processor will try to use semijoins as often as possible when generating a query plan for a given query (a technique known as “pushing projections” [7]). Also in distributed query processing, semijoins have great importance, because when a database is distributed across several sites, they can help avoid the shipment of many unneeded tuples.

Because of its practical importance, we would like to have a clear knowledge of the capabilities and the limitations of semijoins. For example, Bernstein, Chiu and Goodman [4, 5] have characterized the conjunctive queries computable by semijoin programs. In this paper, we consider the much larger class of queries computable in the variant of the relational algebra obtained by replacing the join operator by the semijoin operator. We call this the semijoin algebra (SA). A join of two relations combines all tuples satisfying a given condition, called

*This author has been partially supported by the European Community Research Training Network “Games and Automata for Synthesis and Validation” (GAMES), contract HPRN-CT-2002-00283.

the join condition. A semijoin differs from a join in the sense that it selects only those tuples in the first relation that participate in the join. The semijoin algebra is a fragment of the relational algebra, which is known to be equivalent to first-order logic (called relational calculus in database theory [1]).

Interestingly, there is a fragment of first-order logic very similar to the semijoin algebra: it is the so called “guarded fragment” (GF) [2, 8, 9, 10], which has been studied in the field of modal logic. This is interesting because the motivations to study this fragment came purely from the field of logic and had nothing to do with database query processing. Indeed, the purpose was to extend propositional modal logic to the predicate level, retaining the good properties of modal logic, such as the finite model property. An important tool in the study of the expressive power of the GF is the notion of “guarded bisimulation”, which provides a characterization of the discerning power of the GF.

We will show that when we allow only equalities to appear in the semijoin conditions, the semijoin algebra has essentially the same expressive power as the guarded fragment. When also nonequalities or other predicates are allowed, the semijoin algebra becomes more powerful. We will define a generalization of guarded bisimulation, in the form of an Ehrenfeucht-Fraïssé game, that characterizes the discerning power of the semijoin algebra. We will use this tool to show that certain queries are not expressible in SA.

2 Preliminaries

In this section, we give formal definitions of the semijoin algebra and the guarded fragment.

From the outset, we assume a universe $\mathbb{U}$ of basic data values, over which a number of predicates are defined. These predicates can be combined into quantifier-free first-order formulas, which are used in selection and semijoin conditions. The names of these predicates and their arities are collected in the vocabulary Ω . The equality predicate ($=$) is always in Ω . A database schema is a finite set $\mathbf{S}$ of relation names, each associated with its arity. $\mathbf{S}$ is disjoint from Ω . A database D over $\mathbf{S}$ is an assignment of a finite relation $D(R) \subseteq \mathbb{U}^n$ to each $R \in \mathbf{S}$, where n is the arity of R .

Proviso. When φ stands for a first-order formula, then $\varphi(x_1, \dots, x_k)$ indicates that all free variables of φ are among $x_1, \dots, x_k$.

First, we define the Semijoin Algebra.

Definition 1 (Semijoin algebra, SA). Let $\mathbf{S}$ be a database schema. Syntax and semantics of the Semijoin Algebra is inductively defined as follows:

1. Each relation $R \in \mathbf{S}$ is a semijoin algebra expression.
2. If $E_1, E_2 \in \text{SA}$ have arity n , then also $E_1 \cup E_2$, $E_1 - E_2$ belong to SA and are of arity n .
3. If $E \in \text{SA}$ has arity n and $X \subseteq \{1, \dots, n\}$, then $\pi_X(E)$ belongs to SA and is of arity $\#X$.

4. If $E_1, E_2 \in \text{SA}$ have arities n and m , respectively, and $\theta_1(x_1, \dots, x_n)$ and $\theta_2(x_1, \dots, x_n, y_1, \dots, y_m)$ are quantifier-free formulas over Ω , then also $\sigma_{\theta_1}(E_1)$ and $E_1 \bowtie_{\theta_2} E_2$ belong to SA and are of arity n .

The semantics of the projection, the selection and the semijoin operator are as follows: $\pi_X(E) := \{(a_i)_{i \in X} \mid (a_1, \dots, a_n) \in E\}$, $\sigma_{\theta_1}(E) := \{(a_1, \dots, a_n) \in E \mid \theta_1(a_1, \dots, a_n) \text{ holds}\}$, $E_1 \bowtie_{\theta_2} E_2 := \{(a_1, \dots, a_n) \in E_1 \mid \exists (b_1, \dots, b_m) \in E_2, \theta_2(a_1, \dots, a_n, b_1, \dots, b_m) \text{ holds}\}$. The semantics of the other operators are well known.

Now, we recall the definition of the guarded fragment.

Definition 2 (Guarded fragment, GF). Let $\mathbf{S}$ be a database schema.

1. All quantifier-free first-order formulas over $\mathbf{S}$ are formulas of GF.
2. If φ and ψ are formulas of GF, then so are $\neg\varphi$, $\varphi \vee \psi$, $\varphi \wedge \psi$, $\varphi \rightarrow \psi$ and $\varphi \leftrightarrow \psi$.
3. If $\varphi(\bar{x}, \bar{y})$ is a formula of GF and $\alpha(\bar{x}, \bar{y})$ is an atomic formula such that all free variables of φ do actually occur in α then $\exists \bar{y}(\alpha(\bar{x}, \bar{y}) \wedge \varphi(\bar{x}, \bar{y}))$ is a formula of GF.

As the guarded fragment is a fragment of first-order logic, the semantics of GF is that of first-order logic, interpreted over the active domain of the database [1].

3 Semijoin algebra versus guarded fragment

In this section, $\Omega = \{=\}$ consists only of the equality predicate. Suppose furthermore that we only allow conjunctions of equalities to be used in the semijoin conditions; selection conditions can be arbitrary quantifier-free formulas over Ω . We will denote the semijoin algebra with this restriction on the semijoin conditions by $\text{SA}^=$. Before we prove that $\text{SA}^=$ is subsumed by GF, we need a lemma.

Lemma 3. *For every $\text{SA}^=$ expression E of arity k , for every database A and for every tuple $\bar{z} = (z_1, \dots, z_k)$ in $E(A)$, there exists R in $\mathbf{S}$, an injective function $f : \{1, \dots, k\} \rightarrow \{1, \dots, \text{arity}(R)\}$, and a tuple $\bar{t}$ in $A(R)$ such that $\bigwedge_{i=1}^k z_i = t_{f(i)}$.*

Proof. By structural induction on expression E . □

Theorem 4. *For every $\text{SA}^=$ expression E of arity k , there exists a GF formula φ_E such that for every database D , $E(D) = \{\bar{d} \in D \mid \varphi_E(\bar{d})\}$.*

Proof. The proof is by structural induction on E .

- if E is R , then $\varphi_E(x_1, \dots, x_k) := R(x_1, \dots, x_k)$.
- if E is $E_1 \cup E_2$, then $\varphi_E(x_1, \dots, x_k) := \varphi_{E_1}(x_1, \dots, x_k) \vee \varphi_{E_2}(x_1, \dots, x_k)$.

- if E is $E_1 - E_2$, then $\varphi_E(x_1, \dots, x_k) := \varphi_{E_1}(x_1, \dots, x_k) \wedge \neg \varphi_{E_2}(x_1, \dots, x_k)$.
- if E is $\sigma_\theta(E_1)$, then $\varphi_E(x_1, \dots, x_k) := \varphi_{E_1}(x_1, \dots, x_k) \wedge \theta(x_1, \dots, x_k)$.
- if E is $\pi_{i_1, \dots, i_k}(E_1)$ with E_1 of arity n , then, by induction, $\varphi_{E_1}(z_1, \dots, z_n)$ defines all tuples in $E_1(D)$. By Lemma 3, $\varphi_{E_1}(\bar{z})$ is equivalent to the formula obtained by replacing in $\psi :=$

$$\bigvee_{R \in \mathbf{S}} \bigvee_{\substack{f: \{1, \dots, n\} \rightarrow \\ \{1, \dots, \text{arity}(R)\}}} \exists (t_j)_{j \in Q} (R(\bar{t}) \wedge \varphi_{E_1}(t_{f(1)}, \dots, t_{f(n)}))$$

each $t_{f(i)}$ by z_i , $i = 1, \dots, n$. In this formula, Q is a shorthand for the set $\{1, \dots, \text{arity}(R)\} - f(\{1, \dots, n\})$. Formula φ_E should now only select components $i_1, \dots, i_k$ out of this formula. To this end, we modify ψ such that in each disjunct it quantifies over $(t_j)_{j \in Q'}$ with $Q' = \{1, \dots, \text{arity}(R)\} - f(\{i_1, \dots, i_k\})$ and in each disjunct $t_{f(i_l)}$ is replaced by x_l , $l = 1, \dots, k$. Now $\varphi_E(x_1, \dots, x_k)$ is obtained.

- if E is $E_1 \bowtie_\theta E_2$ with $\theta = \bigwedge_{l=1}^s x_{i_l} = y_{j_l}$ and E_2 of arity n , then, by induction, $\varphi_{E_1}(x_1, \dots, x_k)$ and $\varphi_{E_2}(z_1, \dots, z_n)$ define all tuples in $E_1(D)$ and $E_2(D)$ respectively. By Lemma 3, $\varphi_E(x_1, \dots, x_k)$ is obtained by replacing in formula $\chi :=$

$$\phi_{E_1}(x_1, \dots, x_k) \wedge \bigvee_{R \in \mathbf{S}} \bigvee_{\substack{f: \{1, \dots, n\} \rightarrow \\ \{1, \dots, \text{arity}(R)\}}} \exists (t_j)_{j \in Q''} (R(\bar{t}) \wedge \varphi_{E_2}(t_{f(1)}, \dots, t_{f(n)}))$$

each $t_{f(j_l)}$ by x_{i_l} , $l = 1, \dots, s$. Note that condition θ is enforced by repetition of variables x_{i_l} . In this formula, $Q'' = \{1, \dots, \text{arity}(R)\} - f(\{j_1, \dots, j_s\})$.

□

By the decidability of GF, we obtain:

Corollary 5. *Satisfiability of $SA^=$ expressions is decidable.*

With decidability of SA expressions, we always mean finite satisfiability, because a database is finite by definition.

The literal converse statement of Theorem 4 is not true, because the guarded fragment contains all quantifier-free first-order formulas, so that one can express arbitrary cartesian products in it, such as $\{(x, y) \mid S_1(x) \wedge S_2(y)\}$. Cartesian products, of course, can not be expressed in the semijoin algebra. Nevertheless, the result of any GF query restricted to a single relation by a semijoin is always expressible in $SA^=$:

Theorem 6. *For every GF formula $\varphi(x_1, \dots, x_k)$, for every relation R (with arity n), for every injective function $f : \{1, \dots, k\} \rightarrow \{1, \dots, n\}$, the query $\{\bar{x} \mid \varphi(\bar{x})\} \bowtie_\theta R$ in which θ is $\bigwedge_{i=1}^k x_i = y_{f(i)}$, is expressible in $SA^=$.*

Proof. By structural induction on φ , we construct the desired semijoin expression $E_{\varphi,k}^{f,R}$.

- if $\varphi(x_1, \dots, x_k)$ is $T(x_{i_1}, \dots, x_{i_l})$ then $E_{\varphi,k}^{f,R} := \pi_{f(1), \dots, f(k)}(R) \bowtie_{\theta} T$, where θ is $(x_{i_1} = y_1) \wedge (x_{i_2} = y_2) \wedge \dots \wedge (x_{i_l} = y_l)$;
- if $\varphi(x_1, \dots, x_k)$ is $(x_i = x_j)$ then $E_{\varphi,k}^{f,R} := \sigma_{i=j}(\pi_{f(1), \dots, f(k)}(R))$;
- if $\varphi(x_1, \dots, x_k)$ is $\psi(x_1, \dots, x_k) \vee \xi(x_1, \dots, x_k)$ then $E_{\varphi,k}^{f,R} := E_{\psi,k}^{f,R} \cup E_{\xi,k}^{f,R}$;
- if $\varphi(x_1, \dots, x_k)$ is $\neg\psi(x_1, \dots, x_k)$ then $E_{\varphi,k}^{f,R} := \pi_{f(1), \dots, f(k)}(R) - E_{\psi,k}^{f,R}$;
- suppose $\varphi(x_1, \dots, x_k)$ is $\exists \bar{z}(\alpha(\bar{x}, \bar{z}) \wedge \psi(\bar{x}, \bar{z}))$, where α is atomic with relation name T . Let $x_{i_1}, \dots, x_{i_r}$ be the different occurrences of variables among $x_1, \dots, x_k$ in α . Now, $E_{\varphi,k}^{f,R} := \pi_{f(1), \dots, f(k)}(R) \bowtie_{\theta} E_{\psi, r+l}^{g,T}$ where θ is $(x_{i_1} = y_1) \wedge (x_{i_2} = y_2) \wedge \dots \wedge (x_{i_r} = y_r)$ and g is the function that maps $j \in \{1, \dots, r\}$ to the position of x_{i_j} in α and that maps $j \in \{r+1, \dots, r+l\}$ to the position of z_{j-r} in α .

□

Taking $k = 0$ and R equal to any nonempty relation in the above theorem, we obtain:

Corollary 7. *Over the class of nonempty databases GF sentences and 0-ary $SA^=$ expressions have equal expressive power.*

Here, a database is said to be empty if all its relations are empty.

Let us now allow arbitrary semijoin conditions (still over equality only). Specifically, nonequalities are now allowed. We will denote the semijoin algebra over $\Omega = \{=\}$ by $SA^{\neq}$. Then, GF no longer subsumes $SA^{\neq}$. A counterexample is the query that asks whether there are at least two distinct elements in a single unary relation S . This is expressible in $SA^{\neq}$ as $S \bowtie_{x_1 \neq y_1} S$, but is not expressible in GF. Indeed, a set with a single element is guarded bisimilar to a set with two elements [2, 10].

Unfortunately, these nonequalities in semijoin conditions make SA undecidable.

Theorem 8. *Satisfiability of $SA^{\neq}$ expressions is undecidable.*

Proof. Grädel [8, Theorem 5.8] shows that GF with functionality statements in the form of functional[D], saying that the binary relation D is the graph of a partial function, is a conservative reduction class. Since functional[D] is expressible in $SA^{\neq}$ as $D \bowtie_{x_1=y_1 \wedge x_2 \neq y_2} D = \emptyset$, it follows that $SA^{\neq}$ is undecidable. □

In the next section, we will generalize guarded bisimulation to the semijoin algebra, with arbitrary quantifier-free formulas over Ω as semijoin conditions.

4 An Ehrenfeucht-Fraïssé game for the semijoin algebra

In this section, we describe an Ehrenfeucht-Fraïssé game that characterizes the discerning power of the semijoin algebra.

Let A and B be two databases over the same schema $\mathbf{S}$. The *semijoin game* on these databases is played by two players, called the spoiler and the duplicator. They, in turn, choose tuples from the tuple spaces T_A and T_B , which are defined as follows: $T_A := \bigcup_{R \in \mathbf{S}} \bigcup \{ \pi_X(A(R)) \mid X \subseteq \{1, \dots, \text{arity}(R)\} \}$, and T_B is defined analogously. So, the players can pick tuples from the databases and projections of these.

At each stage in the game, there is a tuple $\bar{a} \in T_A$ and a tuple $\bar{b} \in T_B$. We will denote such a configuration by $(A, \bar{a}; B, \bar{b})$. The conditions for the duplicator to win the game with 0 rounds are:

1. $\forall R \in \mathbf{S}, \forall X \subseteq \{1, \dots, \text{arity}(R)\} : \bar{a} \in \pi_X(A(R)) \Leftrightarrow \bar{b} \in \pi_X(B(R))$
2. for every atomic formula (equivalently, for every quantifier-free formula) θ over Ω , $\theta(\bar{a})$ holds iff $\theta(\bar{b})$ holds.

In the game with $m \geq 1$ rounds, the spoiler will be the first one to make a move. Therefore, he first chooses a database (A or B). Then he picks a tuple in T_A or in T_B respectively. The duplicator then has to make an “analogous” move in the other tuple space. When the duplicator can hold this for m times, no matter what moves the spoiler takes, we say that the duplicator wins the m -round semijoin game on A and B . The “analogous” moves for the duplicator are formally defined as legal answers in the next definition.

Definition 9 (legal answer). Suppose that at a certain moment in the semijoin game, the configuration is $(A, \bar{a}; B, \bar{b})$. If the spoiler takes a tuple $\bar{c} \in T_A$ in his next move, then the tuples $\bar{d} \in T_B$, for which the following conditions hold, are legal answers for the duplicator:

1. $\forall R \in \mathbf{S}, \forall X \subseteq \{1, \dots, \text{arity}(R)\} : \bar{d} \in \pi_X(B(R)) \Leftrightarrow \bar{c} \in \pi_X(A(R))$
2. for every atomic formula θ over Ω , $\theta(\bar{a}, \bar{c})$ holds iff $\theta(\bar{b}, \bar{d})$ holds.

If the spoiler takes a tuple $\bar{d} \in T_B$, the legal answers $\bar{c} \in T_A$ are defined identically.

In the following, we denote the semijoin game with initial configuration $(A, \bar{a}; B, \bar{b})$ and that consists of m rounds, by $G_m(A, \bar{a}; B, \bar{b})$.

We first state and prove

Proposition 10. *If the duplicator wins $G_m(A, \bar{a}; B, \bar{b})$, then for each semijoin expression E with $\leq m$ nested semijoins and projections, we have $\bar{a} \in E(A) \Leftrightarrow \bar{b} \in E(B)$.*

Proof. We prove this by induction on m . The base case $m = 0$ is clear. Now consider the case $m > 0$. Suppose that $\bar{a} \in E_1 \times_\theta E_2(A)$ but $\bar{b} \notin E_1 \times_\theta E_2(B)$. Then $\bar{a} \in E_1(A)$ and $\exists \bar{c} \in E_2(A) : \theta(\bar{a}, \bar{c})$, and either (*) $\bar{b} \notin E_1(B)$ or (**) $\neg \exists \bar{d} \in E_2(B) : \theta(\bar{b}, \bar{d})$. In situation (*), $\bar{a}$ and $\bar{b}$ are distinguished by an expression with $m-1$ semijoins or projections, so the spoiler has a winning strategy; in situation (**), the spoiler has a winning strategy by choosing this $\bar{c} \in E_2(A)$ with $\theta(\bar{a}, \bar{c})$, because each legal answer of the duplicator $\bar{d}$ has $\theta(\bar{b}, \bar{d})$ and therefore $\bar{d} \notin E_2(B)$. So, the spoiler now has a winning strategy in the game $G_{m-1}(A, \bar{c}; B, \bar{d})$. In case a projection distinguishes $\bar{a}$ and $\bar{b}$, a similar winning strategy for the spoiler exists. In case $\bar{a}$ and $\bar{b}$ are distinguished by an expression that is neither a semijoin, nor a projection, there is a simpler expression that distinguishes them, so the result follows by structural induction. $\square$

We now come to the main theorem of this section. This theorem concerns the game $G_\infty(A, \bar{a}; B, \bar{b})$, which we also abbreviate as $G(A, \bar{a}; B, \bar{b})$. We say that the duplicator wins $G(A, \bar{a}; B, \bar{b})$ if the spoiler has no winning strategy. This means that the duplicator can keep on playing forever, choosing legal answers for every move of the spoiler.

Theorem 11. *The duplicator wins $G(A, \bar{a}; B, \bar{b})$ if and only if for each semijoin expression E , we have $\bar{a} \in E(A) \Leftrightarrow \bar{b} \in E(B)$.*

Proof. The ‘only if’ direction of the proof follows directly from Proposition 10, because if the duplicator wins $G(A, \bar{a}; B, \bar{b})$, he wins $G_m(A, \bar{a}; B, \bar{b})$ for every $m \geq 0$. So, $\bar{a}$ and $\bar{b}$ are indistinguishable through all semijoin expressions. For the ‘if’ direction, it is sufficient to prove that if the duplicator loses, $\bar{a}$ and $\bar{b}$ are distinguishable. We therefore construct, by induction, a semijoin expression $E_{\bar{a}}^r$ such that (i) $\bar{a} \in E_{\bar{a}}^r(A)$, and (ii) $\bar{b} \in E_{\bar{a}}^r(B)$ iff the duplicator wins $G_r(A, \bar{a}; B, \bar{b})$. We define $E_{\bar{a}}^0$ as

$$\sigma_{\theta_{\bar{a}}} \left(\bigcap_{R \in \mathbf{S}} \bigcap_{\{X \subseteq Z \mid \bar{a} \in \pi_X(A(R))\}} \pi_X(R) \right) - \bigcup_{R \in \mathbf{S}} \bigcup_{\{X \subseteq Z \mid \bar{a} \notin \pi_X(A(R))\}} \pi_X(R)$$

In this expression, Z is a shorthand for $\{1, \dots, \text{arity}(R)\}$ and $\theta_{\bar{a}}$ is the *atomic type* of $\bar{a}$ over Ω , i.e., the conjunction of all atomic and negated atomic formulas over Ω that are true of $\bar{a}$.

We now construct $E_{\bar{a}}^r$ in terms of $E_{\bar{a}}^{r-1}$:

$$\bigcap_{\bar{c} \in T_A} (E_{\bar{a}}^0 \times_{\theta_{\bar{a}, \bar{c}}} E_{\bar{c}}^{r-1}) \cap (E_{\bar{a}}^0 - \bigcup_{j=1}^s \bigcup_{\theta} (E_{\bar{a}}^0 \times_{\theta} \bigcap_{\substack{\bar{c} \in T_A \\ \theta(\bar{a}, \bar{c})}} (E_{\bar{c}}^{r-1})^{\text{compl}}))$$

In this expression, $\theta_{\bar{a}, \bar{c}}$ is the atomic type of $\bar{a}$ and $\bar{c}$ over Ω ; s is the maximal arity of a relation in $\mathbf{S}$; θ ranges over all atomic Ω -types of two tuples, one with the arity of $\bar{a}$, and one with arity j . The notation E^{compl} , for an expression of arity k , is a shorthand for

$$\left(\bigcup_{R \in \mathbf{S}} \bigcup_{\substack{X \subseteq \{1, \dots, \text{arity}(R)\} \\ \#X = k}} \pi_X(R) \right) - E$$

□

5 Queries inexpressible in the semijoin algebra

Grädel [8] already showed that transitivity is not expressible in the guarded fragment. We will now show that transitivity is still inexpressible in the more powerful semijoin algebra.

Theorem 12. *Transitivity is inexpressible in the semijoin algebra.*

Proof. We will give two databases A and B over the schema $\mathbf{S}$ containing a single relation R , that are indistinguishable by semijoin expressions, and with the property that R is transitive in A and not in B . These databases are shown graphically in Figure 1.

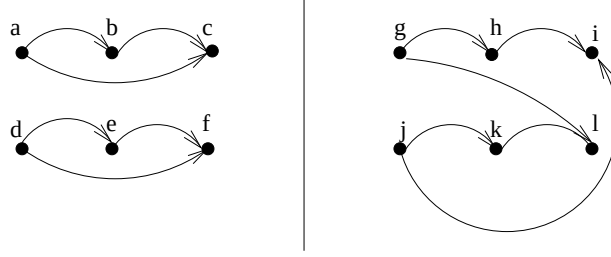


Figure 1: Databases A (left) and B (right) that imply inexpressibility of transitivity in the semijoin algebra

In this figure the edges represent the relation R . A moment's inspection reveals that the duplicator has a winning strategy in the semijoin game $G(A, \langle \rangle; B, \langle \rangle)$. For the sake of completeness we give here the formal strategy. We do this by using the following bijections from tuple space T_A to T_B .

$f : T_A \rightarrow T_B$		$g : T_A \rightarrow T_B$	
$a \mapsto g$	$ab \mapsto gh$	$a \mapsto j$	$ab \mapsto jk$
$b \mapsto h$	$bc \mapsto hi$	$b \mapsto k$	$bc \mapsto kl$
$c \mapsto i$	$de \mapsto jk$	$c \mapsto l$	$de \mapsto gh$
$d \mapsto j$	$ef \mapsto kl$	$d \mapsto g$	$ef \mapsto hi$
$e \mapsto k$	$ac \mapsto gl$	$e \mapsto h$	$ac \mapsto ji$
$f \mapsto l$	$df \mapsto ji$	$f \mapsto i$	$df \mapsto gl$

When the spoiler makes his first move, the duplicator has a legal answer by taking the image or pre-image of the spoiler's chosen tuple under bijection f . The duplicator now continues answering each spoiler move by applying f or f^{-1} to the chosen tuple, until:

- in configuration $(A, ac; B, gl)$ the spoiler chooses bc or kl , or
- in configuration $(A, bc; B, hi)$ the spoiler chooses ac or ji , or

- in configuration $(A, df; B, ji)$ the spoiler chooses ef or hi , or
- in configuration $(A, ef; B, kl)$ the spoiler chooses df or gl .

In either case, the duplicator answers with the tuple obtained from applying g or g^{-1} to the chosen tuple, and from then, he follows strategy function g . Following g , he switches back to strategy function f whenever:

- in configuration $(A, ac; B, ji)$ the spoiler chooses bc or hi , or
- in configuration $(A, bc; B, kl)$ the spoiler chooses ac or gl , or
- in configuration $(A, df; B, gl)$ the spoiler chooses ef or kl , or
- in configuration $(A, ef; B, hi)$ the spoiler chooses df or ji .

□

Another example of a query inexpressible in the semijoin algebra is the following:

Theorem 13. *The query $R = \pi_1(R) \times \pi_2(R)$ about a binary relation R is inexpressible in the semijoin algebra.*

Proof. In Figure 2, two databases A and B are shown where A satisfies the query and B does not. The duplicator has a winning strategy in the semijoin game $G(A, \langle \rangle; B, \langle \rangle)$. □

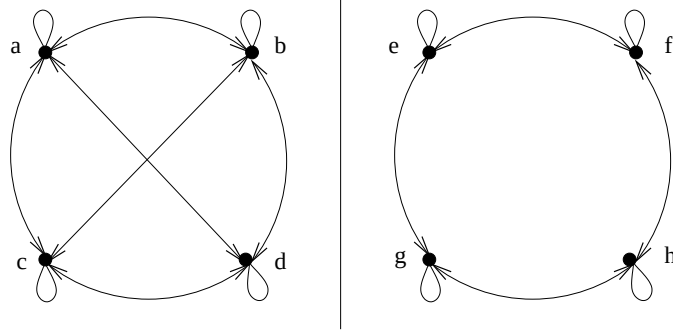


Figure 2: Databases A (left) and B (right) that imply inexpressibility of the query of Theorem 13.

6 Impact of order

In this section, we investigate the impact of order. On ordered databases (where Ω now also contains a total order on the domain), the query that asks if there

are at least k elements in a unary relation S becomes expressible as $\text{at_least}(k)$, which is inductively defined as follows:

$$\begin{cases} \text{at_least}(1) &:= S \\ \text{at_least}(k) &:= S \ltimes_{x_1 < y_1} (\text{at_least}(k-1)) \end{cases}$$

Note that this query is independent of the chosen order. This parallels the situation in first-order logic, where there also exists an order-invariant query that is expressible with but inexpressible without order ([1, Exercise 17.27] and [6, Proposition 2.5.6]).

Transitivity remains inexpressible in the semijoin algebra even on ordered databases. Consider the following databases A and B over a single binary relation R : $A(R)$ is the union of X , Y and Z , where $X = \{1, \dots, m\} \times \{2m+1\}$, $Y = \{2m+1\} \times \{m+1, \dots, 2m\}$, and $Z = \{1, \dots, m\} \times \{m+1, \dots, 2m\}$; $B(R) = A(R) - \{(\frac{m+1}{2}, m + \frac{m+1}{2})\}$. Clearly, R is transitive in A , but not in B . We have shown elsewhere [11] that when $m = 2n+1$, the duplicator has a winning strategy in the n -round semijoin game $G_n(A, \langle \rangle; B, \langle \rangle)$. By Proposition 10, transitivity is not expressible in SA with order.

References

- [1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases*. Addison-Wesley, 1995.
- [2] H. Andreka, I. Nemeti, and J. van Benthem. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27(3):217–274, 1998.
- [3] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30(3):479–513, 1983.
- [4] P.A. Bernstein and D.W. Chiu. Using semi-joins to solve relational queries. *Journal of the ACM*, 28(1):25–40, 1981.
- [5] P.A. Bernstein and N. Goodman. Power of natural semijoins. *SIAM Journal on Computing*, 10(4):751–771, 1981.
- [6] H.-D. Ebbinghaus and J. Flum. *Finite model theory*. Springer, 1999.
- [7] H. Garcia-Molina, J.D. Ullman, and J. Widom. *Database systems: the complete book*. Prentice Hall, 2000.
- [8] E. Grädel. On the restraining power of guards. *Journal of Symbolic Logic*, 64(4):1719–1742, 1999.
- [9] E. Grädel, C. Hirsch, and M. Otto. Back and forth between guarded and modal logics. In *Proc. 15th IEEE Symp. on Logic in Computer Science*. IEEE Computer Society Press, 2000.

- [10] C. Hirsch. *Guarded logics: Algorithms and bisimulation*. PhD thesis, RWTH Aachen, 2002.
- [11] D. Leinders, J. Tyszkiewicz, and J. Van den Bussche. On the expressive power of semijoin queries. *arXiv cs.DB/0308014*. To appear in Information Processing Letters.
- [12] M. Yannakakis. Algorithms for acyclic database schemes. In *Proc. of Intl. Conf. on Very Large Data Bases*, pages 82–94. IEEE Press, 1981.

Managing Database Incompleteness with the Guarded Fragment

Michael J. Minock

Department of Computing Science

The University of Umeå, Sweden 90187

Phone: +46 90 786 6398 FAX: +46 90 786 6126

Email: *mjm@cs.umu.se*

Abstract

Traditional databases do not explicitly represent the portions over their schemas for which they are sound and complete. This paper proposes a method by which very fine grained meta-data descriptions may be attached to relational data sources to describe, among other things, the portions of various tables for which the database has complete data. Because of the formal properties of the meta-data description language, itself expressible within the guarded fragment of first-order logic, intensional descriptions of the coverage offered to a specific query may be generated. The techniques discussed in this paper are general and may apply to many scenarios where fine-grained meta-data descriptions are associated with tuple sets.

1 Introduction

Often one is prepared to make very specific declarations of soundness and completeness in a database. For example I am willing to say that my database has correct records for all of the commercial airports in Sweden. I may continue and declare that my database has all the schedule information for all of the flights to the Swedish airport with code ‘UME’ on a specific date. In any case these declarations of local soundness and completeness are fine-grained views¹ and are asserted as a dynamic part of populating or modifying the database.

The problem is that these declarations should be merged and managed so that when the database is queried, a coherent description of the database’s completeness (or incompleteness) may be provided with respect to the specific query. Thus when the user asks for all the airport tuples in Sweden, Norway or Denmark they should be informed that their answer is complete for Sweden and not so for the remaining countries.

¹Hereafter let us assume that such views are sound. The notion of complete in this case corresponds to the closed world assumption being applicable over the view.

More abstractly, consider that we have a set of n views: $v_1, \dots, v_n$ that describe the portion of the schema for which the database has sound and complete coverage. The problem is, given a query q , to calculate the portion of q that may be answered with *certainty*² using the views. The answer to this problem is a disjoint partition of the original query q into two queries q_{certain} and $q_{\text{uncertain}}$ where q_{certain} is the portion of the query that may be answered with certainty and $q_{\text{uncertain}}$ is the remainder that may not be answered with certainty over the views.

In relation to the general problem of answering queries using views (see [14]), the problem here is to obtain a maximally contained rewriting of a query using a set of closed-world views. Of note the language in which views (or queries) are expressed is well-beyond [6] the class of conjunctive queries [7]. The key to obtaining a solution to this problem, is the limitation of projection to whole schema tuples [16] and the restriction of tuple calculus formulas to a decidable class that may be translated to the guarded fragment of first-order logic [2].

The plan of this paper is as follows. Section 2 covers basic definitions and introduces the notion of *schema tuple queries* which give us a convenient way to describe sets of tuples within database tables. Section 3 presents the guarded schema tuple query language $\mathcal{L}_{GT}$ and its extension to full multi-typed disjunction, Q_{GT} . Additionally section 3 shows that one can calculate syntactic difference and intersection as well as decide query containment for queries expressed within Q_{GT} . Section 4 discusses some of the formal limitations of $\mathcal{L}_{GT}$ (and Q_{GT}) and some weak methods that might remedy them. Section 5 presents a solution to the problem of deriving maximally contained rewritings where the query and view descriptions are expressed within the language $\mathcal{L}_{GT}$ and the certain answer description is expressed within the language Q_{GT} . Section 6 discusses this work in relation to related work. Section 7 concludes this paper and outlines some of the prospects for future work in this area. The proofs are given in the appendix.

2 Basic Notions

Let us assume the existence of three disjoint sets: $\mathcal{U}$, the *universal domain of atomic values*, $\mathcal{P}$, *predicate names* and $\mathcal{A}$, *attribute names*. Without loss of generality, let us assume that $\mathcal{U}$ is a countably infinite, totally ordered set. Let U be a distinct symbol representing the type of $\mathcal{U}$. A *relation schema* R is a sequence $\langle A_1 : U, \dots, A_n : U \rangle$ where $n \geq 1$ is called the *arity* of R and all $A_i \in \mathcal{A}$ are distinct attribute names. A *database schema* D is a sequence $\langle P_1 : R_1, \dots, P_m : R_m \rangle$, where $m \geq 1$, P_i 's are distinct predicate names and R_i 's are relation schemas. A *relation instance* r of R with arity n is a finite subset of $\mathcal{U}^n$. A *database instance* d of D is a sequence $\langle P_1 : r_1, \dots, P_m : r_m \rangle$, where r_i is a relation instance of R_i for $i \in [1..m]$. Hereafter we assume the database schema is fixed and thus drop reference to D .

²The notion of certain answers here is that of certain answers in [1] under closed-world views. Specifically an answer τ to query q is certain, if it is an answer to q over all databases where $v_1, \dots, v_n$ are sound and complete views. See the formal definition of certain answers in section 5.

2.1 Tuple Relational Calculus

The tuple relational calculus is a ‘syntactic sugar’ over first order logic that lets us range variables over tuples of a relation, rather than over constants within a fixed argument position of a relation. Tuple calculus is widely used in database research and was introduced by Codd [9].

Let Z be the set of all tuple variables and for $z \in Z$ let $z.a$ be the *component reference* to attribute a of the relation over which z ranges. We say that such component references are *meaningful* if a is indeed an attribute of the relation over which z ranges. Let θ denote one of the operators ($=, >, <, \geq, \leq$ or $\neq$), let ϵ denote a set membership operator ($\in$ or $\notin$).

Let us inductively define the *tuple relational formulas*. The *atomic tuple formulas* are:

1. **Range Conditions:** $P(z)$, where P is a predicate name and $z \in Z$.
2. **Simple Conditions:** $X\theta Y$, where X is a component reference and $Y \in \mathcal{U}$.
3. **Join Conditions:** $X\theta Y$, where both X and Y are component references.
4. **Set Conditions:** $X\epsilon Y$, where X is a component reference and Y is a set of constants drawn from $\mathcal{U}$. (e.g. $x.A \in \{y_1, \dots, y_n\}$)

All atomic formulas are tuple relational formulas and if F_1 and F_2 are tuple relational formula, where F_1 has some free variable z , then $F_1 \wedge F_2$, $F_1 \vee F_2$, $\neg F_1$, $(\exists z)F_1$ and $(\forall z)F_1$ are also tuple relational formulas. The set $Free(\alpha) \subseteq Z$ denotes the free tuple variables of the formula α . We say a condition is *meaningful* if all of its component references are meaningful. We say that a formula is *safe* if all of its tuple variables range over a fixed number of relations.

2.2 Integrity Constraints

As is common, primary keys are expressed through functional dependencies and functional dependencies are expressed as universally quantified tuple formulas. Specifically the functional dependency $W \rightarrow A$ over the relation P where W is a set of m attributes and A is a single attribute of P is expressed as the universally quantified formula: $(\forall x)(\forall y)(P(x) \wedge P(y) \wedge y.w_1 = x.w_1 \wedge \dots \wedge y.w_m = x.w_m \Rightarrow y.a = x.a)$. We shall use the symbol F to denote the set of all of the functional dependencies that hold over the database.

Foreign key constraints are simply a special case of inclusion dependencies. An inclusion dependency from the arbitrary sequence of attributes $a_1, \dots, a_m$ in relation P to the arbitrary sequence of attributes $b_1, \dots, b_m$ in P' is expressed as the formula: $(\forall x)(P(x) \Rightarrow (\exists y)(P'(y) \wedge x.a_1 = y.b_1 \wedge \dots \wedge x.a_m = y.b_m))$. We shall use the symbol I to denote all of the inclusion dependencies that hold over the database.

Example 1 (A relational database schema):

City(name, country, latitude, longitude)
Airport(code, country, latitude, longitude)
Flight(fnum, carrier, from_{Airport}, to_{Airport})
Schedule(flight_{Flight}, date, depTime, arrTime)

where the primary keys are underlined and the foreign keys are subscripted by the relation to which they refer.

2.3 Schema Tuple Queries

So much for relational orthodoxy, we now introduce the notion of typed schema tuples and schema tuple queries.

Definition 1 (Schema tuples)

The *schema tuple* τ of the database instance d is the pair $\langle P_i : \mu \rangle$, where $1 \leq i \leq m$ and μ is a single tuple in r_i .

Example 2 (Two example schema tuples)

$\tau_1 = \langle \text{Airport} : 'ULM', 'Germany', 47, 13 \rangle$

$\tau_2 = \langle \text{City} : 'ULM', 'Germany', 47, 13 \rangle$

We say that $\text{type}(\tau) = P_i$ and we say the components of τ are the components of μ . The schema tuple τ_1 is equal to the schema tuple τ_2 if and only if $\text{type}(\tau_1) = \text{type}(\tau_2)$ and they match on all components. Thus $\tau_1 \neq \tau_2$ even though they have equal corresponding components.

Definition 2 (Schema tuple queries)

A *schema tuple query* is an expression of the form $\{x|\alpha\}$, where α is a tuple relational formula over the single free tuple variable x .

When we write the expression $\{x|\alpha\}$ we normally assume that the expression α is over the free tuple variable x . For the schema tuple τ , $\tau \in \{x|\alpha\}$ if $\{\tau/x\}(\alpha)$ where $\{t/x\}\alpha$ means to substitute the term t in place of x in α . Thus a *schema tuple query* is simply the intensional description of a schema tuple set. The actual tuples within this set are the extensional answers to the schema tuple query.

3 Guarded Schema Tuple Query Languages

We shall now turn our attention to an interesting sub-language of the language of tuple relational formulas.

3.1 $\mathcal{L}_{GT}$ and its decidability

We shall inductively define the set of *guarded components*. Note that in this definition, we assume that the guarded component ψ over the variable y may be written as $s \cdot (\exists y)(P(y) \wedge \phi)$ where s is the sign (empty or $\neg$), $P(y)$ is the guard and ϕ is the *body* of the component.

Definition 3 (*Guarded components*)

1. The formula $(\exists y)(P(y) \wedge \phi)$ is a guarded component where ϕ is a conjunction of meaningful set and simple conditions limited to the operators $=$ and $\neq$ over y .
2. If ψ and ψ' are guarded components³, then $(\exists y)(P(y) \wedge \phi \wedge s' \cdot (\exists y')(P'(y') \wedge y.a = y'.b \wedge \phi'))$ is a guarded component⁴ where $y.a = y'.b$ is a meaningful equality join between the attribute a of P and the attribute b of P' .
3. If ψ and ϕ are guarded components, then $\psi \wedge \phi$ and $\neg\psi$ are guarded components.

Note that the guarded component ψ may be written as $(\exists y)(P(y) \wedge \phi_0 \wedge \phi_1 \wedge \dots \wedge \phi_n)$, where ϕ_i is the i -th conjunct of ϕ . We say that the guarded component ϕ is *normal* if it is written so that all the simple and set conditions over y are grouped in ϕ_0 and each ϕ_i for $0 < i \leq n$ involves exactly 1 normal child-component.

Definition 4 (*The language $\mathcal{L}_{GT}$*)

The language $\mathcal{L}_{GT}$ consists of all formulas of the form,

$$P(x) \wedge (\bigwedge_{i=0}^n \phi_i)$$

obtained from a normal guarded component over the variable x .

Thus the language $\mathcal{L}_{GT}$ is obtained by simply removing the existential quantifier at the head of a normal guarded component. Note that queries built using $\mathcal{L}_{GT}$ are *non-cyclic*. A query is said to be *self-joining* if the queries descendent components have a variable that ranges over the same relation that the free variable x ranges over. Finally we speak of the *width* of a query as the depth of the deepest descendent of the head component of the query. Of the following queries, only q_3 is self joining and no query has a width greater than 3.

Example 3 (*Queries and views built using $\mathcal{L}_{GT}$*)

³Written $s \cdot (\exists y)(P(y) \wedge \phi)$ and $s' \cdot (\exists y')(P'(y') \wedge \phi')$ respectively.

⁴ ψ' is said to be a *child-component* of ψ in this case. Likewise, ψ is said to be the *parent* of ψ' . This naturally leads to the corresponding notions *descendent* (*ancestor*) components of a given component.

v₁ : “The airports in Sweden”
 $\{x | \text{Airport}(x) \wedge x.\text{country} = \text{“Sweden”}\}$

q₁ : “The airports in Scandinavia”
 $\{x | \text{Airport}(x) \wedge x.\text{country} \in \{\text{“Sweden”}, \text{“Norway”}, \text{“Denmark”}\}\}$

v₂ : “Flights within Sweden”
 $\{x | \text{Flight}(x) \wedge$
 $(\exists y_1)(\text{Airport}(y_1) \wedge y_1.\text{code} = x.\text{to} \wedge y_1.\text{country} = \text{“Sweden”}) \wedge$
 $(\exists y_2)(\text{Airport}(y_2) \wedge y_2.\text{code} = x.\text{from} \wedge y_2.\text{country} = \text{“Sweden”})\}$

q₂ : “Flights to Umeå, not from Arlanda”
 $\{x | \text{Flight}(x) \wedge$
 $(\exists y_1)(\text{Airport}(y_1) \wedge$
 $y_1.\text{code} = x.\text{to} \wedge y_1.\text{code} = \text{“UME”} \wedge y_1.\text{country} = \text{“Sweden”}) \wedge$
 $\neg(\exists y_2)(\text{Airport}(y_2) \wedge$
 $y_2.\text{code} = x.\text{from} \wedge y_2.\text{code} = \text{“ARN”} \wedge y_2.\text{country} = \text{“Sweden”})\}$

v₃ : “Schedules for domestic flights to Umeå in first 2 days of December, 2003”
 $\{x | \text{Schedule}(x) \wedge$
 $x.\text{date} \in \{\text{“20031201”}, \text{“20031202”}\}$
 $(\exists y_1)(\text{Flight}(y_1) \wedge x.\text{flight} = y_1.\text{fnum} \wedge$
 $(\exists y_2)(\text{Airport}(y_2) \wedge y_1.\text{from} = y_2.\text{code} \wedge y_2.\text{country} = \text{“Sweden”}) \wedge$
 $(\exists y_3)(\text{Airport}(y_3) \wedge y_1.\text{to} = y_3.\text{code} \wedge y_3.\text{country} = \text{“Sweden”} \wedge$
 $y_3.\text{code} = \text{“UME”})\}$

q₃ : “Swedish airports that have flights to every other airport in Sweden”
 $\{x | \text{Airport}(x) \wedge$
 $x.\text{country} = \text{“Sweden”} \wedge$
 $\neg(\exists y_1)(\text{Airport}(y_1) \wedge y_1.\text{country} = \text{“Sweden”} \wedge$
 $\neg(\exists y_2)(\text{Flight}(y_2) \wedge y_2.\text{from} = x.\text{code} \wedge y_2.\text{to} = y_1.\text{code}))\}$

Note that in q_3 we assume that all airports have a flight to themselves.

Theorem 1 ($\mathcal{L}_{GT}$ is decidable for satisfiability)

For all $\ell \in \mathcal{L}_{GT}$ over the free variable x , we may decide if there exists a database instance d , satisfying the inclusion dependencies I , for which $\{\tau/x\}\ell$ where τ is a schema tuple in d .

3.2 The Language Q_{GT}

We define the language Q_{GT} as the closure of $\mathcal{L}_{GT}$ over disjunction where the free variable within each disjunct may range over a different relation.

Definition 5 (*The language Q_{GT}*)

$q \in Q_{GT}$ if q is written as a finite expression $\ell_1 \vee \dots \vee \ell_k$ where each $\ell \in \mathcal{L}_{GT}$ and ℓ is over the free tuple variable x .

Theorem 2 (*Q_{GT} is decidable for satisfiability*)

For all $q \in Q_{GT}$ over the free variable x , we may determine if there exists a database instance d , satisfying the inclusion dependencies I , for which $\{\tau/x\}q$ where τ is a schema tuple in d .

3.3 Syntactic intersection and difference

We may describe the set difference (or intersection) of two queries built over Q_{GT} with a third query built over Q_{GT} .

Theorem 3 (*Syntactic intersection of Q_{GT} queries*)

Let $q_1 \in Q_{GT}$ and $q_2 \in Q_{GT}$. Then there is a $q_3 \in Q_{GT}$ such that for all database instances $\{x|q_1\} \cap \{x|q_2\} = \{x|q_3\}$.

Theorem 4 (*Syntactic difference of Q_{GT} queries*)

Let $q_1 \in Q_{GT}$ and $q_2 \in Q_{GT}$. Then there is a $q_3 \in Q_{GT}$ such that for all database instances $\{x|q_1\} - \{x|q_2\} = \{x|q_3\}$.

3.4 Query containment

Theorem 5 (*$\subseteq, =$ and disjointness over Q_{GT}*)

If $q_1 \in Q_{GT}$, $q_2 \in Q_{GT}$ and I are the set of inclusion dependencies, then one may decide if:

1. $\{x|q_1\} \subseteq \{x|q_2\}$
2. $\{x|q_1\} = \{x|q_2\}$
3. $\{x|q_1\} \cap \{x|q_2\} = \emptyset$.

are necessarily true over the set of all database instances for which I holds.

4 Weak Properties

There are two issues that remain formally unresolved in this work: *enforcing functional dependencies* and *simplifying queries*. In both cases we comment on the formal difficulties as well as on weaker methods to practically address these issues.

4.1 Enforcing Functional Dependencies

Unfortunately there are limitations when considering decidability of $\mathcal{L}_{GT}$ under functional dependencies. The guarded fragment is not decidable in the company of assertion statements [12]. Thus the following query is deemed to be satisfiable:

$$\{x | \text{Flight}(x) \wedge (\exists y_1)(\text{Airport}(y_1) \wedge x.\text{from} = y_1.\text{code} \wedge y_1.\text{country} = \text{"Sweden"}) (\exists y_2)(\text{Airport}(y_2) \wedge x.\text{from} = y_2.\text{code} \wedge y_2.\text{country} = \text{"Norway"})\}$$

The approach here is to rewrite expressions in $\mathcal{L}_{GT}$ into enhanced expressions that capture at least some of the relevant functional dependencies. Thus for the functional dependency $W \rightarrow A$ over the relation P , a simple case of this is achieved by rewriting any component $(\exists y)(P(y) \wedge y.a = c)$ where $c \in \mathcal{U}$, to $(\exists y)(P(y) \wedge y.a = c \wedge \neg(\exists y')(P(y') \wedge y.w_1 = y'.w_1 \wedge \dots \wedge y.w_m = y'.w_m \wedge y'.a \neq c))$. Clearly this does not cause any problems because the language $\mathcal{L}_{GT}$ stays closed under such a transformation.

Thus based on the functional dependency $\text{code} \rightarrow \text{country}$ holding over Airport we transform the query from above to:

$$\{x | \text{Flight}(x) \wedge (\exists y_1)(\text{Airport}(y_1) \wedge x.\text{from} = y_1.\text{code} \wedge y_1.\text{country} = \text{"Sweden"}) \neg(\exists y_3)(\text{Airport}(y_3) \wedge y_1.\text{code} = y_3.\text{code} \wedge y_3.\text{country} \neq \text{"Sweden"}) (\exists y_2)(\text{Airport}(y_2) \wedge x.\text{from} = y_2.\text{code} \wedge y_2.\text{country} = \text{"Norway"}) \neg(\exists y_4)(\text{Airport}(y_4) \wedge y_2.\text{code} = y_4.\text{code} \wedge y_4.\text{country} \neq \text{"Norway"})\}$$

In this case the functional dependency is reflected and the query is correctly deemed to be unsatisfiable. This technique, by itself, still lacks strength. Consider the example query:

$$\{x | \text{Schedule}(x) \wedge (\exists y_1)(\text{Flight}(y_1) \wedge x.\text{flight} = y_1.\text{fnum} \wedge (\exists y_2)(\text{Airport}(y_2) \wedge y_1.\text{from} = y_2.\text{code} \wedge y_2.\text{country} = \text{"Sweden"})) (\exists y_3)(\text{Flight}(y_3) \wedge x.\text{flight} = y_3.\text{fnum} \wedge (\exists y_4)(\text{Airport}(y_4) \wedge y_3.\text{from} = y_4.\text{code} \wedge y_4.\text{country} = \text{"Norway"}))\}$$

In this query the above technique is insufficient because the functional dependencies over Flight never get a chance to impact the decision. What we need is an extended technique to push constants throughout the query based on functional dependencies. To get a clearer view of this process, consider that the child relationship between guarded components defines a path through the schema that uses either foreign key to primary key (*foreign key reference*) or primary key to foreign key (*primary key referenced by*) steps. Let us assume the existence of a function that maps a given primary key constant k to a corresponding foreign key constant within the same table. Such a function is defined for every pair of primary/foreign

key attributes of a table. We denote this function on the relation R between the primary key K and the foreign key attribute A as $FK_{K,A}^R(\mathcal{U}^n) \rightarrow \mathcal{U}$ where n is the number of attributes in K .

If we Skolemize the translation of our query in FOL, then (at least) all of the domain variables corresponding to the free variable x will have Skolem constants⁵. We may then use these FK functions to push constants through foreign key references. Interestingly this technique works no matter what the sign of the quantifier and stands as an orthogonal technique to standard Skolemization. The process is guaranteed to generate a finite Herbrand universe because the queries are non-cyclic and thus the application of the FK function will be finite. If this technique is used along with the rewriting technique above then the problem query above is deemed unsatisfiable.

One can imagine an even more extended version of this approach where each descendent component of a query is individually tested as a separate satisfiability problem, complete with rewriting and FK based constant propagation. Though of practical importance, it is unlikely that such an approach will ever fully resolve the formal difficulties of enforcing functional dependencies over $\mathcal{L}_{GT}$. Further evidence for this limitation comes from the undecidability of inferring additional inclusion or functional dependencies from a given set of functional and inclusion dependencies [17][8]. Recent progress [15] has been made to tease out those cases where the interaction is limited, and it remains a goal to incorporate such results into the approach described here.

4.2 Simplification

Given an expression $q \in Q_{GT}$ we say that q' is a *equivalent rewriting* of q if there is a finite sequence of semantically equivalent formulas $q_1, \dots, q_m$ such that q_1 is q and q_m is q' and each q_{i+1} is obtained from q_i by applying a *rewrite rule*. The rewrite rules that we use include the routine rewrite rules [19] and several special cases designed around the syntactic form of Q_{GT} . Of interest we also include rewrite rules that test if two distinct existential variables can be merged or if one may drop variables, components or conditions wholesale. In such cases, an explicit equivalence test is issued to insure that a semantically equivalent query has been written.

In summary, we have a set of sound, but not necessarily complete set of rewrite rules for queries in Q_{GT} . We employ these rewrites to search for *shorter* (in terms of the number of conditions) equivalent rewritings of a query. In practice we are able to simplify a wide class of queries within reasonable time.

⁵Note that the unique names assumption does not hold over such constants.

5 Reporting Completeness

We now shall solve a general case of the problem posed in the introduction. Namely for the query q and the n views $v_1, \dots, v_n$ expressing sound and complete coverage, we seek to partition the query into the two disjoint pieces q_{certain} and $q_{\text{uncertain}}$ where the portion q_{certain} may be evaluated over the database to retrieve all available certain answers to the query. Let us define certain answers in this context.

Definition 6 (*Certain answers*)

Given a set of sound and complete views $v_1, \dots, v_n$, each written $\{x_i | v_i\}$, a tuple τ is a *certain answer* to the query $\{x | \alpha\}$, if and only if $\tau \in \{x | \alpha\}$ for all database states d consistent with the extensions of $v_1, \dots, v_n$.

Naturally for the problem here we assume that $\alpha \in \mathcal{L}_{GT}$ and $v_i \in \mathcal{L}_{GT}$. The problem is thus to show a procedure to obtain a schema tuple query $\{x | \sigma\}$ that obtains only the certain answers to $\{x | \alpha\}$ given the sound and complete views.

5.1 Obtaining certain query descriptions from a single view

We observe that it is not necessarily the case that *any* tuple from a view whose description semantically intersects with the query description, will actually be a certain answer to the query. Formally, for the i -th view, if $\tau \in \{x_i | \{x_i/x\}\ell \wedge v_i\}$ then it is not necessarily true that τ is a certain answer to $\{x | \ell\}$.

For intuition consider that a schema tuple view written in $\mathcal{L}_{GT}$, on its own, is not able to evaluate conditions on other relations⁶. So, when considering a query posed over a single view, we just as well might *truncate* the query so that it is *definable* over the view. Formally query truncation is carried out by simply *dropping* all non-free variables of the query formula ℓ . Dropping a variable means that the variable and any corresponding components using the variable are removed. The result of this process is written $\{x | \text{drop}(\ell)\}$ and it always results in a more general query.

Theorem 6 (*Certainty test over single views*)

τ is a certain answer to the not self-joining query $\{x | \ell\}$ posed over the single view $\{x' | v\}$ if and only if for all database instances d , $\tau \in \{x' | \{x'/x\}\text{drop}(\ell) \wedge v\}$ and $\{x' | \{x'/x\}\text{drop}(\ell) \wedge v\} = \{x' | \{x'/x\}\ell \wedge v\}$.

5.2 Obtaining certain query descriptions over a set of views

In the actual algorithm we search out the most general view combinations that return certain answers to the query. We call each view combination a *plan*. Specifically a plan is a rewrite of ℓ where variables are singly matched with views from the given set of n views. The manner in which this is achieved is straightforward. A

⁶Self-joining queries are an exception to this statement.

search tree is built where each node contains a plan. The original query formula ℓ is the plan within the root node of a search tree. Nodes are expanded, by replacing an unmatched variable from the original ℓ with a view description from the set of n views. Formally, assume that the component $(\exists y)(P(y) \wedge \phi_0 \wedge \phi_1 \wedge \dots \wedge \phi_n)$ occurs within the plan ℓ' and v_i is $P(x_i) \wedge \phi'$. The match between y and v_i is reflected in the rewritten component $(\exists x_i)(P(x_i) \wedge \{x_i/y\}\phi_0 \wedge \{x_i/y\}\phi_1 \wedge \dots \wedge \{x_i/y\}\phi_n \wedge \phi')$. We say that $\ell'' \in \mathcal{L}_{GT}$ is obtained from ℓ' where such a component has been rewritten. At the level of the search tree, we say the node with the plan ℓ'' is obtained from the node with the plan ℓ' by matching the variable y within ℓ' to the view v_i .

Variable matching occurs top-down in the plan moving from parent to child components in ℓ . The termination test for node expansion is if the node meets one of the following three conditions:

1. The node's plan is unsatisfiable.
2. The node's plan, with unmatched variables dropped, can be used to give certain answers to the query.
3. There are no remaining unmatched variables within the node's plan.

This process results in a finite tree of $O((nm)^m)$ nodes where n is the number of views and m is the number of tuple variables in ℓ . This tree is traversed and the set of certain answer generating plans are obtained. These plans are pooled into an expression in Q_{GT} that intensionally describes the certain answers. This expression may be evaluated over the database to generate the actual certain answers, or it may be simplified and presented as a summary description⁷.

Theorem 7 (*Certain answers over a set of views*)

Given a set of sound and complete views $v_1, \dots, v_n$, each written as $\{x_i|v_i\}$ where $v_i \in \mathcal{L}_{GT}$ and the non self-joining query $\{x|\ell\}$, the certain answers to $\{x|\ell\}$ may be written as the expression $\{x|\sigma\}$ where $\sigma \in Q_{GT}$.

5.3 Examples of certainty reporting

Here we see the ideas of reporting completeness in action. These examples draw upon expressions from example 3 as well as the following additional inputs:

v₄ : “Schedule information for all domestic Swedish flights for November 30 and December 1, 2003”

$\{x|Schedule(x) \wedge$
 $x.date \in \{“20031130”, “20031201”\} \wedge$

⁷It is possible that such descriptions include impossible states that violate the functional dependencies of the database. Thankfully, by definition, such descriptions are never answer yielding. Moreover if the techniques of section 4.1 are used, many empty answer reports will not occur in practice.

$$\begin{aligned}
&(\exists y_1)(Flight(y_1) \wedge x.flight = y_1.fnum \wedge \\
&(\exists y_2)(Airport(y_2) \wedge y_1.from = y_2.code \wedge y_2.country = \text{"Sweden"}) \wedge \\
&(\exists y_3)(Airport(y_3) \wedge y_1.to = y_3.code \wedge y_3.country = \text{"Sweden"}))\}
\end{aligned}$$

q₄ : “Schedule information for flights arriving in Umeå or Luleå, Sweden”

$$\begin{aligned}
&\{x | Schedule(x) \wedge \\
&(\exists y_1)(Flight(y_1) \wedge x.flight = y_1.fnum \wedge \\
&(\exists y_2)(Airport(y_2) \wedge y_1.to = y_2.code \wedge y_2.code \in \{\text{"UME"}, \text{"LUL"}\} \wedge \\
&y_2.country = \text{"Sweden"}))\}
\end{aligned}$$

Example 4 (Example outputs)

Assuming that the set of views are v_1, v_2, v_3 and v_4 from above, query $q_{1_certain}$ is exactly the form of v_1 , for query q_2 :

$$\begin{aligned}
&\mathbf{q_{2_certain}} : \\
&\{x | Flight(x) \wedge \\
&(\exists y_1)(Airport(y_1) \wedge \\
&y_1.code = x.to \wedge y_1.code = \text{"UME"} \wedge y_1.country = \text{"Sweden"}) \wedge \\
&\neg(\exists y_2)(Airport(y_2) \wedge \\
&y_2.code = x.from \wedge y_2.code = \text{"ARN"} \wedge y_2.country = \text{"Sweden"}) \\
&(\exists y_3)(\mathbf{Airport(y_3)} \wedge \mathbf{y_3.code = x.from} \wedge \mathbf{y_3.country = \text{"Sweden"}}))\}
\end{aligned}$$

q_3 may be answered completely and for query q_4 :

$$\begin{aligned}
&\mathbf{q_{4_certain}} : \\
&\{x | Schedule(x) \wedge \\
&\mathbf{x.date} \in \{\text{"20031130"}, \text{"20031201"}, \text{"20031202"}\} \wedge \\
&(\exists y_1)(Flight(y_1) \wedge x.flight = y_1.fnum \wedge \\
&(\exists y_2)(Airport(y_2) \wedge y_1.to = y_2.code \wedge y_2.code \in \{\text{"UME"}, \text{"LUL"}\} \wedge \\
&y_2.country = \text{"Sweden"})) \\
&\vee \\
&Schedule(x) \wedge \\
&\mathbf{x.date} \in \{\text{"20031201"}, \text{"20031202"}\} \wedge \\
&(\exists y_1)(Flight(y_1) \wedge x.flight = y_1.fnum \wedge \\
&(\exists y_2)(Airport(y_2) \wedge y_1.to = y_2.code \wedge \mathbf{y_2.code = \text{"UME"}} \wedge \\
&y_2.country = \text{"Sweden"}))\}
\end{aligned}$$

Material in bold states what additional conditions are necessary for certainty given that the views v_1, v_2, v_3 and v_4 are sound and complete.

5.4 Simplification of a set of views

As a final issue, we consider that one would like to compile view descriptions off-line into more compact semantically equivalent forms to speed up on-line processing. This is simply the minimization process of 4.2 with the added constraint that any rewriting of the set of views $v_1, \dots, v_n$ to $v'_1, \dots, v'_m$ must be semantically equivalent and *must able to give certain answers to each input view* $v_1, \dots, v_n$.

6 Related Work

The problem addressed in this work relates to the two problems of deriving maximally contained rewritings of a query over a set of views [14] and the problem of obtaining certain answers [1][13] over closed-world views. In fact, we combine these problems into the single problem of deriving a certain rewriting of a query, given a set of closed world views.

The derivation of the certain answer description, is greatly simplified by the schema tuple query assumption. To the author's knowledge no prior work has invoked the 'schema tuple query' restriction explicitly. Certainly one could imagine a regime in which the schema tuple restriction would be enforced over non-recursive Datalog. In such a case Q_{GT} would be correspond to the class of non-recursive Datalog programs with negation in their rule bodies.

Though it may seem to be an odd restriction, there are two principle reasons why the schema tuple restriction is made: The first reason is that set difference is fully defined over schema tuple query answer sets⁸. Since the atoms of sets are whole tuples, we can speak of taking the 'typed' difference between two tuple sets. In fact we can derive such query differences syntactically without having to materialize answer sets (see theorems 3 and 4). We may also speak of taking differences between tuple sets of heterogeneous types. If, however, we relax the schema tuple query restriction and permit projections and thus non-typed relations, then we face the prospect of taking differences between non-union compatible, non-typed relations. The second reason for the schema tuple restriction centers around the perspicuity of intensional descriptions. Because tables correspond roughly to the nouns and verbs of a domain, descriptions should center around collections of such objects. Unrestricted projection would probably so complicate query expressions that the task of generating crisp, understandable natural language descriptions would be doomed.

The limitations of the schema tuple restriction may usually be overcome by considering a virtual, highly decomposed version of the schema. Alternatively one may enable projection and simple aggregation as an activity on the periphery of a core schema tuple expression processor. Finally it should be noted that there is a very direct correspondence between queries specified in $\mathcal{L}_{GT}$ and standard SQL. Queries in $\mathcal{L}_{GT}$ mirror the SQL of the form:

```
SELECT *  
FROM R AS x  
WHERE  
  [NOT] EXISTS (sub-query) AND ... ;
```

⁸Decidable description logics [10] of the type *ALC* and beyond also share this property ($A(x) - B(x)$ may be represented as $A(x) \sqcap \neg B(x)$). Description logics typically use unary and binary predicates which formalize traditional semantic network role/filler systems. As such they are interesting fragments of logic over predicates of at most two variables [4]. For an in depth discussion of this limitation with respect to databases see [16].

where the *sub-query* is of the same form. Naturally simple and join conditions may be added to such queries and there may be more than one sub-query.

The language $\mathcal{L}_{GT}$ is related to the language described in [16], when the author was unaware of the guarded fragment [2] of first-order logic. This prior language, now named $\mathcal{L}_{SBT}$ relied on the decidability of the $\exists^*\forall^*$ fragment [3]. Interestingly this fragment is not normally decidable under complementation ($\neg\exists^*\forall^*\rho \equiv \forall^*\exists^*\neg\rho$), yet $\mathcal{L}_{SBT}$ is. The reason that $\mathcal{L}_{SBT}$ stays closed under complementation is intimately connected with the aforementioned schema tuple query assumption. In any case it will be interesting to compare the practical and theoretical trade-offs between $\mathcal{L}_{SBT}$ and $\mathcal{L}_{GT}$.

It should finally be mentioned that the practical problem addressed in this work is very similar in spirit to the problem confronted by the PANORAMA system [18]. PANORAMA allows users to attach *properties* to views and then decides which other views inherit such properties. PANORAMA's view definition language is strictly conjunctive [7] and the set of properties include *soundness*, *completeness* and *emptiness*, but it is an open, extensible set. PANORAMA associates a meta-schema with the actual schema where the meta-schema encodes the view definitions with associated properties. Queries are processed over both the base relations as well as the meta relations. The normal extensional answer is augmented with a set of view/property pairs that cover the extensional answers. The technique is efficient, yet incomplete. That is not all relevant covering views will be reported in the meta-answer. Motro terms his approach *algebraic* and contrasts it with the logical based work in *semantic query optimization* [5].

7 Conclusions and future work

The main result of this paper is to show that if sound and complete materialized views are described using the language $\mathcal{L}_{GT}$ and the user's query is specified using the language $\mathcal{L}_{GT}$ and is not self-joining, then one may generate intensional descriptions (within the language Q_{GT}) of the certain answers of the query over the views. Moreover one may directly apply the resulting query over the materialized views to obtain such certain answers.

The languages $\mathcal{L}_{GT}$ and Q_{GT} may be translated into the highly expressive guarded fragment of first order logic [2][12]. We envision the application of these results in a number of areas including, but not limited to, data warehousing, semantic query optimization and cooperative query answering.

8 Acknowledgments

I thank Steve Hegner for his comments on an earlier draft of this paper. I also thank the anonymous reviewers that pointed out several flaws in my initial definition of $\mathcal{L}_{GT}$.

9 Bibliography

References

- [1] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *Sym. on Principles of Database Systems*, pages 254–263, 1998.
- [2] H. Andreka, J. van Benthem, and I. Nemeti. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27:217–274, 1998.
- [3] P. Bernays and M. Schönfinkel. Zum entscheidungsproblem der mathematischen logik. *M.A.*, 99:342–372, 1928.
- [4] A. Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 82:353–367, 1996.
- [5] U. Chakravarthy, J. Grant, and J. Minker. Logic-based approach to semantic query optimization. *ACM Trans. Database Syst.*, 15(2):162–207, 1990.
- [6] A. Chandra and D. Harel. Structure and complexity of relational queries. *Journal of Computer Systems and Sciences*, 25(1):99–128, 1982.
- [7] A. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proc. 9 of the ACM Sym. on the Theory of Computing*, pages 77–90., 1977.
- [8] A. Chandra and M. Vardi. The implication problem for functional dependencies and inclusion dependencies is undecidable. *SIAM Journal Comput.*, 14(3):671–677, 1985.
- [9] E. Codd. Relational completeness of data base sublanguages. In R. Rustin, editor, *Database Systems*, pages 33–64. Prentice-Hall, 1972.
- [10] F. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in description logics. In G. Brewka, editor, *Studies in Logic, Language and Information*, pages 193–238, 1996.
- [11] H. Ganzinger and H. de Nivelle. A superposition decision procedure for the guarded fragment with equality. In *LICS*, pages 295–304, 1999.
- [12] E. Grädel. On the restraining power of guards. *Symbolic Logic*, 64:1719–1742, 1999.
- [13] G. Grahne and A. Mendelzon. Tableau techniques for querying information sources through global schemas. In *proc. of ICDT*, pages 332–347, 1999.

- [14] A. Halevy. Answering queries using views: A survey. *VLDB Journal* 10(4): 270-294, 2001.
- [15] M. Levene and G. Loizou. How to prevent interaction of functional and inclusion dependencies. *Information Processing Letters*, 71:115–125, 1999.
- [16] M. Minock. Knowledge representation using schema tuple queries. In *KRDB '03*. IEEE Computer Society Press, 2003.
- [17] J.C. Mitchell. The implication problem for functional and inclusion dependencies. *Informatica and Control*, 56:154–173, 1983.
- [18] A. Motro. Panorama: A database system that annotates its answers to queries with their properties. *Intell. Inf. Syst.*, 7(1):51–73, 1996.
- [19] J. Ullman. *Principles of Database and Knowledge-Base Systems*. Computer Science Press, Rockville, Maryland, 1989.

10 Appendix

Proof of Theorem 1:

Let L be a first order language. The *atomic formulas* (or atoms) of L are of the usual forms: $v_1 = v_2$ for domain variables v_1, v_2 and $P(v_1, \dots, v_n)$ for the n -ary relation symbol $P \in \mathcal{P}$.

Definition 7 (Guarded formula [2])

The *guarded formulas* of L are defined inductively as follows.

1. An atomic formula is a guarded formula
2. If ϕ, ψ are guarded formulas, then $\phi \wedge \psi$ and $\neg\phi$ are guarded formulas.
3. Let $\bar{v}$ be a finite, non-empty sequence of domain variables, ψ a guarded formula and G an atom such that $Free(\psi) \subseteq Free(G)$. Then $(\exists \bar{v})(G \wedge \psi)$ is a guarded formula where G is called the *guard* of the quantifier.

Sentences within the guarded fragment with equality are decidable for satisfiability, have the finite model property and have the tree model property [2][12]. If we can translate any expression $\ell \in \mathcal{L}_{GT}$ as well as the inclusions dependencies I into the guarded fragment, then theorem 1 will be proven.

Translation of $\ell \in \mathcal{L}_{GT}$ to the guarded fragment

Because we test for satisfiability, the free variable of ℓ is considered existentially quantified: $\ell' = (\exists x)(\ell)$. This re-frames the problem to that of showing

that guarded components of the tuple calculus, defined inductively in definition 3 may always be translated from the ‘syntactic sugar’ of the tuple calculus into the guarded fragment of first order logic.

In what follows, the notation $\bar{y}$ corresponds to expansion of tuple variable $\underline{y}$ to a set of domain variables. These variables range over the first order predicate $\bar{P}$ which corresponds to P in the tuple calculus. When necessary, we write the individual variables in the first order case. That is we write, $(\exists y_1), \dots, (\exists y_n)(\bar{P}(y_1, \dots, y_n) \wedge \dots)$. Constants are handled through introducing special unary ‘constant relations’ for each constant appearing in a query. Additionally we enforce mutual exclusion between such constant relations, for example $(\forall v)(Norway(v) \Rightarrow \neg Sweden(v))$

It suffices to show that each resulting guarded component in the inductive definition presented in definition 3 may be translated into a first order formula of the guarded fragment.

1. The tuple formula $(\exists \underline{y})(P(\underline{y}) \wedge \phi)$ is translated to the formula $(\exists \bar{y})(\bar{P}(\bar{y}) \wedge \bar{\phi})$ where $\bar{\phi}$ corresponds to the first order expansion of all the conditions of the tuple formula ϕ . In performing this translation, set operators ($\in$ and $\notin$) are rewritten using $=$ and $\neq$ and all simple conditions are expressed through ranging domain variables over introduced constant relations. The resulting formula rests within the guarded fragment.
2. $(\exists \underline{y})(P(\underline{y}) \wedge \phi \wedge s' \cdot (\exists \underline{y}')(P'(\underline{y}') \wedge y.a = y'.b \wedge \phi'))$ is a guarded component where $y.a = y'.b$ is a meaningful equality join between the i -th attribute of P and the j -th attribute of P' . The translation of this component into the guarded fragment is: $(\exists y_1), \dots, (\exists y_i), \dots, (\exists y_n) (\bar{P}(\bar{y}) \wedge \bar{\phi} \wedge s' \cdot (\exists y'_1), \dots, (\exists y'_{j-1}) (\exists y'_{j+1}), \dots, (\exists y'_{n'}) (\bar{P}'(\bar{y}'_1, \dots, \bar{y}'_{j-1}, y_i, \bar{y}'_{j+1}, \dots, \bar{y}'_{n'}) \wedge \bar{\phi}'))$
3. Conjunction and negation of guarded components are merely translated down to the first order fragment, where they are permitted.

Translation of inclusion dependencies to the guarded fragment

Each inclusion dependency in I , is written in tuple calculus as $(\forall x)(P(x) \Rightarrow (\exists y)(P'(y) \wedge x.a_1 = y.b_1 \wedge \dots \wedge x.a_m = y.b_m))$. Such expression may be translated to the guarded fragment as $\neg(\exists \bar{x})(\bar{P}(\bar{x}) \wedge \neg(\exists \bar{y}')(\bar{P}'(\bar{y}' \cup \bar{x}'))$ where $\bar{y}'$ are the variables in $\bar{y}$ which are not joined on and $\bar{x}'$ are the variables in $\bar{x}$ which are joined on where it is assumed that such variables will be properly ordered in $\bar{P}'(\bar{y}' \cup \bar{x}')$. This resulting expression is within the first order guarded fragment.

The translation of the guarded component plus the translation of the inclusion dependencies are conjoined together and the resulting formula is decided for satisfiability [11]. $\square$

Proof of Theorem 2:

Proof:

A satisfiability test may be conducted for each disjunct of $q \in Q_{GT}$. Since the number of disjuncts is finite, based on theorem 1 this process will terminate with a correct answer. $\square$

Proof of Theorem 3:

A simple consequence of associativity and distributivity of $\wedge$ over $\vee$. $\square$

Proof of Theorem 4:

A simple consequence of DeMorgan's law, associativity and distributivity of $\wedge$ over $\vee$. $\square$

Proof of Theorem 5:

1 is true iff $\{x|q_1\} - \{x|q_2\} = \emptyset$ under the constraints I . This may be decided based on theorems 2 and 4. The decidability of 2 follows directly from the ability to decide predicate 1. The decidability of 3 follows directly from the theorems 2 and 3. $\square$

Proof of Theorem 6:

If τ is a certain answer to $\{x|\ell\}$ then $\tau \in \{x|\ell\}$ for all database states d consistent with the extension $\{x'|v\}$. Clearly, therefore, $\tau \in \{x'|v\}$. More specifically $\tau \in \{x'|\{x'/x\}\ell \wedge v\}$. Now, since any possible state of affairs might exist for relations other than the single relation which v ranges over, there are always states that could rule a tuple in or out based on whether ℓ 's arbitrary sub-components are satisfied. Thus for an answer to be certain, that is unable to be ruled in or out based on states outside of v , the sub-components of ℓ must not play a role. Thus $\tau \in \{x'|\{x'/x\}drop(\ell) \wedge v\}$. Since it is the case that $\tau \in \{x'|\{x'/x\}\ell \wedge v\}$, and dropping components may only generalize a query, it must be that $\{x'|\{x'/x\}drop(\ell) \wedge v\} = \{x'|\{x'/x\}\ell \wedge v\}$.

If $\{x'|\{x'/x\}drop(\ell) \wedge v\} = \{x'|\{x'/x\}\ell \wedge v\}$ for all database states and if $\tau \in \{x'|\{x'/x\}drop(\ell) \wedge v\}$ then $\tau \in \{x'|\{x'/x\}\ell \wedge v\}$. Therefore $\tau \in \{x|\ell\}$ and $\tau \in \{x'|v\}$ since, by definition all database states include $\{x'|v\}$. Therefore τ is a certain answer. $\square$

Proof of Theorem 7:

Naturally we take σ to be the expression derived using the algorithm in section 5.2.

If $\tau \in \{x|\sigma\}$ then τ is a certain answer to $\{x|\ell\}$ given the sound and complete views $v_1, \dots, v_n$. This statement is true, because σ is written only using views from $v_1, \dots, v_n$ and simple and set conditions that may be evaluated over such views. Since the views are sound and complete, there is no special database instance d that would rule τ out of the answer set. Thus τ is certain.

If τ is a certain answer to $\{x|\ell\}$ given the sound and complete views $v_1, \dots, v_n$, then $\tau \in \{x|\sigma\}$. It is harder to establish this statement. Assume that τ is a certain answer to $\{x|\ell\}$ given the sound and complete views $v_1, \dots, v_n$. Now suppose that $\tau \notin \{x|\sigma\}$. This means that $\tau \in \{x|\alpha\}$, where α is satisfiable and $\alpha \wedge \sigma$ is not sat-

isfiable. It is clear that α must be written only using views from $v_1, \dots, v_n$ using simple and set conditions that may be evaluated over such views – otherwise some possible state outside of $v_1, \dots, v_n$ could rule τ outside of $\{x|\ell\}$ and thus violate the assumption that τ is certain. Now the crux of the issue is whether α may be written in a more expressive language that would rule τ as a certain answer of $\{x|\ell\}$. But, since $\ell \in \mathcal{L}_{GT}$, any such enhancement of expressivity could not be expressed in ℓ . Given this, to make an impact on what gets ruled in or out of the answer set of ℓ , α must not be in a language that is more expressive than $\mathcal{L}_{GT}$. We may thus assume that $\alpha \in \mathcal{L}_{GT}$. Now given that ℓ employs only so many constants in its components and that a certain answer generating view, as a consequent of theorem 6, must use a tighter predication for its existential variables than ℓ , there are only a finite number of possible re-writings of the n views into an expression of some width of k or less. In fact the algorithm of section 5.2 calculates just this expression for the k set to the width of the widest disjunct of σ . The only possibility left is that α is a rewrite greater than width k that expresses a case of certainty that is not covered by σ . But since all additional predications cause further specialization, this is not possible and thus $\{x|\alpha\} \subseteq \{x|\sigma\}$. This is a contradiction of the assumptions above and thus $\tau \in \{x|\sigma\}$. $\square$

A Guarded Fragment for Abstract State Machines

Antje Nowack

Mathematische Grundlagen der Informatik,
RWTH Aachen, D-52056 Aachen, Germany,
email: nowack@informatik.rwth-aachen.de

Abstract. Abstract State Machines (ASMs) provide a formal method for transparent design and specification of complex dynamic systems. They combine advantages of informal and formal methods. Applications motivate a number of computability and decidability problems connected to ASMs. Such problems result for example from the area of verifying properties of ASMs. Their high expressive power leads rather directly to undecidability respectively uncomputability results for most interesting problems in the case of unrestricted ASMs. Consequently, it is rather natural to ask whether there exist expressive classes of ASMs for which we can prove positive decidability and computability results. In this work, we introduce such a class of ASMs. The idea of this class is the same as for the guarded fragment of first-order logic. We analyse the expressive power of this class and prove that it is stronger than Datalog LITE and the guarded fragment of first-order fixed point logic. Some decidability and computability results have been proven in earlier works.

1 Introduction

Abstract State Machines (ASMs) have been introduced as an attempt to bridge the gap between formal models of computation and practical specification methods. The result is a formal method for transparent design and specification of complex dynamic systems. ASMs combine advantages of informal methods (understandability, executability) with the advantages of formal methods (precision and applicability of mathematical methods and results).

On the one hand, the ASM method is a formal method. There is a precise semantics and this allows the application of mathematical methods to analyse ASMs. In particular, these are often results from mathematical logic as ASMs use classical mathematical structures to describe states of a computation.

Anyhow, ASMs are easily understandable. ASM specifications are easy to read and to write. And this is one of the most important (pre)conditions for a broad acceptance in practice and a resulting application. The reason for the understandability is that ASMs have a very simple syntax that is quite similar to pseudo-code. And indeed, ASMs are used in practice to specify hardware and software systems. Reports on this can be found e.g. in [3].

Furthermore, a number of computability and decidability problems is connected to ASMs mostly motivated by applications. Examples for decidability problems are different variants of the verification problem. The general verification problem - for example - deals with the question whether all computations of an ASM satisfy a property given in some formalism (usually, some temporal logic). An example for computability problems is the computation of a minimal slice. A slice of an ASM consists of a subset of the rules of the original ASM. On a specified part of the states, this slice behaves in exactly the same way as the original ASM. The motivation for computing a minimal slice of an ASM comes from debugging. If an error is observed, the programmer tries to extract that part of the ASM which is responsible for the error. This part might be much smaller than the original ASM.

On the one hand, ASMs are a very powerful model of computation and this is one of their important advantages. In order to witness this, it has been proven that well-defined classes of ASMs simulate sequential, parallel and quantum algorithms on their natural abstraction level (see [11], [2], [8]).

On the other hand, the high expressive power leads rather directly to undecidability resp. uncomputability results for most interesting problems in the case of unrestricted ASMs. For instance, the general verification problem is not decidable and a minimal slice is not computable.

Consequently, it is rather natural to ask whether there exist expressive classes of ASMs for which we can prove positive decidability and computability results. In this work, we introduce such a class of ASMs. Some decidability and computability results have already been proven in other works (see [12], [13]). We analyse the expressive power of this class and prove that it is stronger than Datalog LITE and the guarded fragment of first-order fixed point logic.

Besides the above aspects, we expect our results to increase the understanding of Datalog LITE and the guarded fragment of first-order fixed point logic.

Outline. First, we give a short introduction to ASMs. After briefly considering some decision and computation problems, we introduce the class of guarded (deterministic) ASMs denoted by $\text{GF}(\mathcal{D})$. In the remainder of this work, we prove that $\text{GF}(\mathcal{D})$ is stronger than Datalog LITE and the guarded fragment of first-order fixed point logic.

2 A Brief Background on Abstract State Machines

In this work, $\mathcal{D}$ denotes the class of deterministic ASMs. Readers that are common with ASMs please note that we do not allow the use import (and do not consider it in the remainder). As we restrict our considerations to these ASMs, we give only a short introduction to the class $\mathcal{D}$ and do not present further constructs. For detailed information on ASMs (formerly called evolving algebras), see e.g. [10].

The basic idea of ASMs is the following. The states of an ASM are finite first-

order structures. An ASM is defined via a program that determines the transitions between the states. The basic instructions of programs are update rules of the form $u := v$ where u and v are terms over a given vocabulary. Further types of rules have the form **if** φ **then** R **endif** and **forall** $\bar{x} : \varphi$ **do** R **endforall** where R is a rule and φ a first-order formula. Their semantics is the intuitive one. The rule inside an if-clause is executed if the guard φ evaluates to true. The inner rule of a forall-rule is executed in parallel for all tuples of elements satisfying the guard φ of the forall-rule. A program is a sequence of such rules. In one step of the program, they are all executed in parallel. The program is executed repeatedly. This yields a sequence of structures called a run. At first, a run is infinite as an ASM program can always be applied to an appropriate structure. But we can cut off a run starting from a certain point in case that there are no more changes in the states. We say that the ASM halts.

Note that in the above informal explanation and in the subsequent formal definition, the notion of parallel execution of rules or updates is meant in the sense of real parallelism. The rules or updates are executed simultaneously and this is *not* meant e.g. in the sense of arbitrary interleaving. The updates happen at exactly the same time. And this is the reason why two updates or rules might contradict each other. If e.g. the same item should be set to true and to false at the same time, then this is a contradiction and can not be carried out in a canonical way. We will later address this problem again and present the solution chosen in the context of ASM.

Formally, the syntax of ASM rules defining an ASM from $\mathcal{D}$ is given in figure 1. The notion of term is defined as usual. As we consider relations as special functions (i.e., functions that can only develop to true or to false), the outermost symbol of a term can also be a relation symbol. Terms whose outermost symbol is a relation symbol are called boolean-valued. Consequently, such terms can only evaluate to true or to false.

An ASM rule R defines for every state $\mathcal{A}$ and every variable assignment ξ a set of updates resp. an update set $US(R, \mathcal{A}, \xi)$ (see figure 2).

State. A state of an ASM is a finite first-order structure over the same vocabulary as the program defining the ASM.

Update. An update of a state $\mathcal{A}$ is a tuple $\beta = (f, \bar{a}, b)$ where f is a function symbol of arity r , $\bar{a}$ is an r -tuple of elements from the domain of $\mathcal{A}$ and b is an element from the domain of $\mathcal{A}$. Executing this update means to change the interpretation of f such that the value of f at $\bar{a}$ becomes b and the interpretation of f for the other tuples is left as before. Note that if f is a relation symbol then b must be contained in the set $\{\text{true}, \text{false}\}$.

Executing in parallel all elements of $US(\Pi, \mathcal{A}, \xi)$ leads to the successor state of $\mathcal{A}$ with respect to Π . A problem arises if two updates in an update set contradict each other.

Let $\mathcal{T}$ be a finite vocabulary (i.e., a finite set of relation and function symbols). The class of ASM rules over $\mathcal{T}$ is defined by induction as follows.

Skip Skip is an ASM rule.

Update rule For all terms s over $\mathcal{T}$ and all terms t over $\mathcal{T}$, the assignment $s := t$ is a rule. If s is a boolean-valued term then t has also to be boolean-valued.

Conditional rule If g is an FO-formula over the vocabulary $\mathcal{T}$ and R is a rule (over the vocabulary $\mathcal{T}$) then **if** g **then** R **endif** is a rule (with guard g).

Block If R_1 and R_2 are rules then **do-in-parallel** R_1 R_2 **enddo** is a rule.

Parallelism Let $\bar{z}$ be a tuple of variables, R a rule, φ an FO-formula over the vocabulary $\mathcal{T}$. Then **forall** $\bar{z} : \varphi$ R **endforall** is a rule.

Free and bounded variables are defined as usual with **forall** $\bar{x} : \varphi$ binding $\bar{x}$. A program is a rule without free variables.

Fig. 1. Syntax of ASM rules over the finite vocabulary $\mathcal{T}$

Contradictory. A deterministic update set U is contradictory if it contains a pair of updates $(f, \bar{a}, b)$ and $(f, \bar{a}, c)$ where $b \neq c$; otherwise U is consistent.

To execute a consistent update set $\{(f_1, \bar{a}_1, b_1), \dots, (f_k, \bar{a}_k, b_k)\}$, set $f\bar{a}_1, \dots, f\bar{a}_k$ to $b_1, \dots, b_k$ respectively and leave the rest of the interpretations as they are.

To execute a contradictory deterministic update set, do nothing; the new state is identical to the old one.

Executing Π once for a state $\mathcal{A}$ leads to the successor of $\mathcal{A}$.

Successor. For a deterministic program Π , $\mathcal{A} \vdash_{\Pi} \mathcal{A}'$ holds if $\text{US}(\Pi, \mathcal{A}, \xi)$ transforms the state $\mathcal{A}$ into the state $\mathcal{A}'$. The state $\mathcal{A}'$ is a successor of $\mathcal{A}$ with respect to Π .

The iterated execution of an ASM Π leads to a run of Π .

Run. Let ρ be a sequence $(\mathcal{A}_i)_{i < \lambda}$ of states where λ ranges over the natural numbers and the first infinite ordinal ω . Then ρ is a run of P on I if $\mathcal{A}_i \vdash_P \mathcal{A}_{i+1}$ for all i with $i + 1 < \lambda$ and $\mathcal{A}_0 = I$.

Query. An ASM query is a pair (Π, Q) consisting of an ASM program Π and a designated relation symbol Q . With every structure $\mathcal{A}$ on which the computation of Π eventually halts (i.e., no more changes occur in the states), the query (Π, Q) associates the result $(\Pi, Q)[\mathcal{A}]$, i.e., the interpretation of Q as computed by Π on input $\mathcal{A}$.

The evaluation of a deterministic rule R at a state $\mathcal{A}$ under a variable assignment ξ produces a deterministic update set $U = \text{US}(R, \mathcal{A}, \xi)$. To execute R , execute the update set that is defined by induction on the rule:

- $\text{US}(\text{Skip}, \mathcal{A}, \xi) = \emptyset$
- $\text{US}(f(t_1, \dots, t_r) := t_0, \mathcal{A}, \xi) = \{((f, \llbracket t_1 \rrbracket_\xi^{\mathcal{A}}, \dots, \llbracket t_r \rrbracket_\xi^{\mathcal{A}}), \llbracket t_0 \rrbracket_\xi^{\mathcal{A}})\}$
- $\text{US}(\text{if } g \text{ then } R' \text{ endif}, \mathcal{A}, \xi) = \begin{cases} \text{US}(R', \mathcal{A}, \xi) & \text{if } \mathcal{A}, \xi \models g \\ \emptyset & \text{else} \end{cases}$
- $\text{US}(\text{do-in-parallel } R_1 \ R_2 \ \text{enddo}, \mathcal{A}, \xi) = \text{US}(R_1, \mathcal{A}, \xi) \cup \text{US}(R_2, \mathcal{A}, \xi)$
- $\text{US}(\text{forall } \bar{z} : \varphi \ R' \ \text{endforall}, \mathcal{A}, \xi) = \bigcup_{\bar{a} \in M} \text{US}(R', \mathcal{A}, \xi[\bar{z} \rightarrow \bar{a}])$

For a term t , $\llbracket t \rrbracket_\xi^{\mathcal{A}}$ denotes the value of t in $\mathcal{A}$ with ξ .

Fig. 2. Semantics of ASM rules

3 Decidability and Computability Problems

As already mentioned in the introduction, there is a number of decidability and computability problems connected to ASMs. In this section, we consider two examples. They are both motivated by the application of ASMs in software engineering as a specification formalism. As writing formal specifications is usually not an error free process, we are rather early concerned with the task of finding errors in a specification. There are two basic approaches to detect errors. The first approach is to verify properties automatically. It is clear that this is not possible in many cases as the question whether an ASM satisfies a property is undecidable in the unrestricted version. The second approach is to test ASMs. The executability yields this possibility. We do not consider testing itself but a technique that might be combined with testing (but can also be combined with verification). This technique is called slicing.

The verification problem. The aim of verifying ASMs is to prove that the behaviour of an ASM specification coincides with the desired one. There are mainly two possible meanings for the question whether “the behaviour of an ASM specification coincides with the desired one”:

1. Do all computations of an ASM satisfy a property? (general verification problem)
2. Does the computation of an ASM with a given initial structure satisfy a property? (restricted verification problem)

While the restricted verification problem is decidable for $\mathcal{D}$ and formulae from temporal first-order logic, the general verification problem is undecidable in this case.

The reason for the decidability of the restricted verification problem is the following. As the domain of the states does not change in a run of an ASM from

$\mathcal{D}$, every run of an ASM in $\mathcal{D}$ contains only a finite number of different states. An upper bound for this number can be immediately computed from the initial state. It suffices to consider the initial part of the run up to a length corresponding to this upper bound in order to decide whether a run satisfies a property expressed in linear temporal first-order logic. So, the restricted verification problem is comparable to a model-checking problem with finite space.

The reason for the undecidability of the general verification is that the (finite) satisfiability problem for first-order logic can be immediately reduced to the general verification problem. Nevertheless, it is possible to find decidable instances. One of them consists of a well-defined fragment of the class $\text{GF}(\mathcal{D})$ of ASMs (see [12]).

Slicing. In order to debug or to modify an ASM specification, we are most often not interested in its complete behaviour but only in a specified part of it. Consider for instance the case of debugging. When an error is observed, the developer tries to extract that part of the specification which is responsible for the erroneous behaviour. This set of rules might be much smaller than the original one and therefore, it might be much easier to catch and correct the error(s) if he knows this set.

This observation motivates the concept of slicing. By applying this technique, one extracts rules from an ASM program that are relevant for the behaviour with respect to the specified parts of the states. The result is a slice which is a syntactically correct ASM again. The smaller such a slice is the better.

Consider for example an ASM involving a relation R . If we are only interested in the evaluation of R then we can cut off all rules from the ASM that do not influence it.

Although a minimal slice is not computable for arbitrary ASMs from $\mathcal{D}$, it is computable for ASMs from the fragment $\text{GF}(\mathcal{D})$ of $\mathcal{D}$ (see [13]) which we consider in the following.

4 The Guarded Fragment

Both problems presented in the preceding section are strongly connected to satisfiability problems in mathematical logic. In order to demonstrate this connection, we consider the ASM defined by the program $\Pi := \text{if } \varphi \text{ then } \text{res} := \text{true} \text{ endif}$ where φ is an arbitrary first-order formula not involving the nullary relation symbol res .

Every computation of Π satisfies the temporal formula $(X \text{ res})$ (stating that res holds in the next state) iff $\neg\varphi$ is not finitely satisfiable.

Similarly, computing a minimal slice for an ASM from $\mathcal{D}$ is directly connected to the satisfiability problem for first-order logic. **Skip** is a minimal slice of Π relative to the evaluation of res iff φ is not satisfiable. Furthermore, **Skip** is the only such minimal slice of Π in this case.

Consequently, the undecidability of the finite satisfiability problem for first-order logic FO induces the undecidability of the general verification problem and non-computability of a minimal slice for ASMs from $\mathcal{D}$.

A further witness for the connection to mathematical logic is that many proofs of (un)decidability resp. (non-)computability results are reductions of the considered problems to some (finite) satisfiability problem or vice versa.

The above observations suggest to consider decidable instances of the (finite) satisfiability problem and to try to obtain decidable resp. computable instances of decision resp. computation problems for ASMs by introducing similar restrictions for ASMs. In the past, a number of concepts has been presented in order to obtain decidable instances of the (finite) satisfiability problem. Examples are the so-called prefix classes, restrictions on the vocabularies or the two-variable fragment. In most cases, a transfer of these concepts is not suitable as the corresponding restrictions on the ASMs are very strong.

In [1], an approach to decidable instances of the satisfiability problem for first-order logic has been presented that is essentially different from the ones mentioned above. It originates from an investigation of the reason for the good model-theoretic properties of propositional modal logic (ML). H. Andréka, J. van Benthem and I. Németi have considered the translation from ML to first-order logic a bit closer and detected that in the modal fragment, all quantifications are guarded by some atomic formula. This observation then lead to the definition of the guarded fragment of first-order logic.

Definition 1 *Let $\mathcal{Y}$ be a vocabulary containing only constant and relation symbols. The guarded fragment GF of first-order logic is defined inductively as follows:*

1. *Every relational atomic formula (possibly an equation) belongs to GF.*
2. *GF is closed under propositional connectives $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$.*
3. *If $\psi(\bar{x}, \bar{y})$ is a formula in GF then $\exists \bar{y}(\alpha(\bar{x}, \bar{y}) \wedge \psi(\bar{x}, \bar{y}))$ and $\forall \bar{y}(\alpha(\bar{x}, \bar{y}) \rightarrow \psi(\bar{x}, \bar{y}))$ belong to GF provided that $\text{free}(\psi) \subseteq \text{free}(\alpha) = \bar{x} \cup \bar{y}$ and $\alpha(\bar{x}, \bar{y})$ is a relational atomic formula.*

Here, $\text{free}(\psi)$ means the set of all free variables of ψ .

Based on a number of results (see [1], [5]), Andréka, van Benthem, and Németi assume that the guarded nature of quantification is the main reason for the good model-theoretic and algorithmic properties of modal logic. However, most of the interesting properties are still expressible in GF. The following definition transfers the idea of GF to the ASMs from $\mathcal{D}$.

Definition 2 *The guarded fragment $\text{GF}(\mathcal{D})$ of the class $\mathcal{D}$ of ASMs is the subset of $\mathcal{D}$ such that every element Π of $\text{GF}(\mathcal{D})$ uses only relation and constant symbols and satisfies the following conditions:*

- *the lefthand side and the righthand side of any update rule are boolean-valued terms*
- *every guard of an if-clause in Π is in the guarded fragment of first-order logic*

- every forall-rule has the form **forall** $\bar{x} : \alpha(\bar{x}, \bar{y})$ R **endforall** where $\alpha(\bar{x}, \bar{y})$ is an atomic formula and $\text{free}(R) \subseteq \text{free}(\alpha) = \{\bar{x}, \bar{y}\}$

Note that the definition of the guarded fragment of $\mathcal{D}$ can be generalized to any class $\mathcal{C}$ of ASMs (also nondeterministic).

5 Basic properties of Guarded ASMs

In section 3, we have already addressed some properties of $\text{GF}(\mathcal{D})$ concerning decidability respectively computability. In this section, we consider some basic properties with regard to the expressive power. The aim of this section is to increase the understanding of $\text{GF}(\mathcal{D})$.

To formulate some basic properties of guarded ASMs, we give the definition of the Gaifman graph of a structure in advance. The Gaifman graph of a structure $\mathcal{A}$ is an undirected graph with the same domain as $\mathcal{A}$. There is an edge between two elements of the domain iff they coexist in some atomic fact in $\mathcal{A}$.

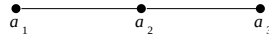
Definition 3 *The Gaifman graph of a relational structure $\mathcal{A}$ (with universe A) is the undirected graph $G(\mathcal{A}) = (A, E^{\mathcal{A}})$ where*

$$E^{\mathcal{A}} = \{(a, a') : a \neq a', \text{ there exists a ground atomic formula } \alpha(b_1, \dots, b_n) \text{ such that } a, a' \in \{b_1, \dots, b_n\}\}$$

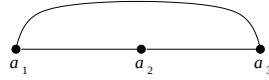
An immediate consequence of the definition of $\text{GF}(\mathcal{D})$ is that we can not access pairs of elements that are not connected via an edge in the Gaifman graph. This fact implies the following lemma.

Lemma 4 *Let $\Pi \in \text{GF}(\mathcal{D})$ not using constants, $\mathcal{A}$ be a state of Π , and $G_{\mathcal{A}} = (V_{\mathcal{A}}, E^{\mathcal{A}})$ be its Gaifman graph. Furthermore, let $\mathcal{B}$ be the state of Π with $\mathcal{A} \vdash_{\Pi} \mathcal{B}$ and let $G_{\mathcal{B}} = (V_{\mathcal{B}}, E^{\mathcal{B}})$ be its Gaifman graph. Then $E^{\mathcal{B}} \subseteq E^{\mathcal{A}}$ holds.*

As a consequence of this lemma, we obtain some non-computability results. E.g., the transitive closure is not computable by an ASM from $\text{GF}(\mathcal{D})$. In order to prove this, consider the class of undirected graphs. In this case, the Gaifman graph of a state $\mathcal{A}$ is obtained from $\mathcal{A}$ by removing all self-referring edges. The Gaifman graph $G_{\mathcal{A}} = (V_{\mathcal{A}}, E^{\mathcal{A}})$ of $\mathcal{A}$ given by



is essentially the state itself. The transitive closure of the above graph is



Again, the Gaifman graph $G_{\mathcal{B}} = (V_{\mathcal{B}}, E^{\mathcal{B}})$ is essentially the state $\mathcal{B}$ itself. But $E^{\mathcal{B}} \not\subseteq E^{\mathcal{A}}$ as $(a_1, a_3) \notin E^{\mathcal{A}}$ (but $(a_1, a_3) \in E^{\mathcal{B}}$).

To disprove the converse of lemma 4 we use the notion of equivalence of ASM in the following sense.

Definition 5 Two (deterministic) ASM-programs Π_1 and Π_2 (over the same vocabulary) are equivalent if for all states $\mathcal{A}$ and $\mathcal{B}$ (of Π_1 and Π_2) the following holds. $\mathcal{A} \vdash_{\Pi_1} \mathcal{B}$ if, and only if, $\mathcal{A} \vdash_{\Pi_2} \mathcal{B}$.

Lemma 6 There exists a relational ASM $\Pi \in \mathcal{D}$ that is not equivalent to an ASM from $\text{GF}(\mathcal{D})$ and for all states $\mathcal{A}$ and $\mathcal{B}$ of Π with Gaifman graphs $G_{\mathcal{A}} = (V_{\mathcal{A}}, E^{\mathcal{A}})$ and $G_{\mathcal{B}} = (V_{\mathcal{B}}, E^{\mathcal{B}})$ the following holds. If $\mathcal{A} \vdash_{\Pi} \mathcal{B}$ then $E^{\mathcal{B}} \subseteq E^{\mathcal{A}}$.

Proof. In order to prove the lemma, we give a simple example of an ASM $\Pi \in \mathcal{D}$ over the vocabulary $\{E/2, P/0\}$ (E is a binary relation symbol and P is a nullary relation symbol). Consider the ASM defined by

if $\forall xyz (Exy \wedge Eyz \rightarrow Exz)$ **then** $P := \text{false}$ **endif**

As we have only negative updates in Π , for all states $\mathcal{A}$ and $\mathcal{B}$ with Gaifman graphs $G_{\mathcal{A}} = (V_{\mathcal{A}}, E^{\mathcal{A}})$ and $G_{\mathcal{B}} = (V_{\mathcal{B}}, E^{\mathcal{B}})$ the following holds. If $\mathcal{A} \vdash_{\Pi} \mathcal{B}$ then $E^{\mathcal{B}} \subseteq E^{\mathcal{A}}$.

Now, we prove that there is no ASM $\Pi' \in \text{GF}(\mathcal{D})$ that is equivalent to Π . The proof is done by contradiction. Assume that there exists an ASM $\Pi' \in \text{GF}(\mathcal{D})$ that is equivalent to Π .

We first demonstrate how we can rewrite Π' such that we obtain an equivalent ASM from $\text{GF}(\mathcal{D})$ of the form **if** ψ **then** $P := \text{false}$ **endif**.

As Π' is equivalent to Π , Π' cannot change other relations than P and P is nullary. Consequently, if Π' contains an update rule with another lefthand side than P then this update rule is never executed. Therefore, we can remove this update rule from Π' resp. replace it by **Skip**.

Furthermore, every update rule of the form $P := t$ where t is an arbitrary boolean-valued term can be replaced by the parallel composition of **if** t **then** $P := \text{true}$ **endif** and **if** $\neg t$ **then** $P := \text{false}$ **endif**. Again, as Π' is equivalent to Π and Π updates P to true in no case, the first conditional rule in the above parallel composition can be removed. Therefore, we obtain **if** $\neg t$ **then** $P := \text{false}$ **endif** instead.

As a first consequence of the above observations we can assume that every update rule appearing in Π' is equal to $P := \text{false}$. Another consequence of these observations is that we can assume that no variable appears in an update rule of Π' . Hence, the successive application of the equivalence preserving rewriting rules given in figure 3 to Π' leads to an ASM from $\text{GF}(\mathcal{D})$ that is equivalent to Π' and thus to Π . If these rules are applied until no more of them is applicable then we obtain an ASM from $\text{GF}(\mathcal{D})$ of the form **if** ψ **then** $P := \text{false}$ **endif**.

As this ASM is equivalent to Π , ψ is equivalent to $\forall xyz (Exy \wedge Eyz \rightarrow Exz)$ and therefore, ψ expresses that E is transitive. As the above ASM is in $\text{GF}(\mathcal{D})$, ψ is in GF . Consequently, transitivity is expressible in GF . This is a contradiction as it is a well-known fact that transitivity is not expressible in GF . $\square$

Note that this proof is essentially a reduction to the respective property of GF and therefore, this is a further witness for the strong connection between ASMs and logic.

original rule	equivalent rule
forall $\bar{x} : \alpha(\bar{x}, \bar{y})$ do if $\varphi(\bar{x})$ then $P := \text{false}$ endif endforall	if $\exists \bar{x}(\alpha(\bar{x}, \bar{y}) \wedge \varphi(\bar{x}))$ then $P := \text{false}$ endif
if φ then if ψ then $\tilde{I}$ endif endif	if $\varphi \wedge \psi$ then $\tilde{I}$ endif
do-in-parallel if φ then $P := \text{false}$ endif if ψ then $P := \text{false}$ endif enddo	if $\varphi \vee \psi$ then $P := \text{false}$ endif

Fig. 3. Rewriting rules applied in the proof of lemma 6

Though we have seen some limits of $\text{GF}(\mathcal{D})$ in this section, many interesting properties are nevertheless definable in $\text{GF}(\mathcal{D})$. We demonstrate that in the remainder of this work.

6 Guarded ASMs are Stronger than Datalog LITE

In this section, we compare $\text{GF}(\mathcal{D})$ to Datalog LITE (see [4]). Datalog LITE is a deductive query language with a linear time model checking algorithm. It is a guarded version of stratified Datalog additionally allowing a limited form of universal quantification in rule bodies.

Despite linear time evaluation, Datalog LITE is highly expressive. In [4], it has been proven that Datalog LITE is equivalent in expressive power to alternation-free guarded fixed point logic, a natural fragment of the guarded fixed point logic. Well-known logical formalisms such as propositional multi-modal logic, computation tree logic (CTL), the alternation free μ -calculus, each correspond to well-defined and syntactically simple fragments of Datalog LITE.

In the next section, we prove that the restriction to the alternation-free fragment is not needed for comparing $\text{GF}(\mathcal{D})$ to guarded fixed-point logic. This restriction has been necessary for the comparison of Datalog LITE to guarded fixed-point logic. Furthermore, we even obtain a strict result in the case of $\text{GF}(\mathcal{D})$. I.e., $\text{GF}(\mathcal{D})$ is stronger than guarded (first-order) fixed point logic. Though this proposition already implies that $\text{GF}(\mathcal{D})$ is stronger than Datalog LITE, we prove it elementarily. We expect to obtain a better insight to the connection between Datalog LITE and $\text{GF}(\mathcal{D})$ in this way as the proof is much more straightforward.

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 $Ru_1 \dots u_s$, and $B_1, \dots, B_r$, the body of the rule, is a collection of literals (i.e., atomic formulae or negated atomic formulae) of the form $Sv_1 \dots v_t$ or $\neg Sv_1 \dots v_t$ where $u_1, \dots, u_s, v_1, \dots, v_t$ are variables. The relation 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 predicate occurs negated in the body of any rule. The predicates that appear only in the bodies of the rules are called extensional or input predicates. Given a relational structure $\mathcal{A}$ over the vocabulary of the input predicates, the program computes, via the usual fixed point semantics, an interpretation for the head predicates. A Datalog query is a pair (Π, Q) consisting of a Datalog program Π and a designated head predicate Q of Π . With every structure $\mathcal{A}$, the query (Π, Q) associates the result $(\Pi, Q)[\mathcal{A}]$, ie., the interpretation of Q as computed by Π on input $\mathcal{A}$.

A stratified Datalog program is a sequence $\Pi = (\Pi_0, \dots, \Pi_r)$ of basic Datalog programs which are 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 extensional predicate only in higher strata Π_j for $j > i$. In particular, this means that

1. if a head predicate of stratum Π_j occurs positively in the body of a rule of a stratum Π_i , then $j \leq i$, and
2. if a head predicate of stratum Π_j occurs negatively in the body of a rule of stratum Π_i , then $j < i$.

The semantics of a stratified program is defined stratum per stratum. The extensional predicates of a stratum Π_i are either extensional in the entire program Π or are head predicates of a lower stratum. Hence, once the lower strata are valuated, we can compute the interpretation of the head predicates of Π_i as in the case of basic Datalog programs.

A Datalog rule is monadic if each of its literals has at most one free variable. Note that this does not necessarily imply that only unary predicates are used. We permit the appearance of literals $\neg S v_1 \dots v_k$ with $k > 1$ that may contain repeated occurrences of a single variable. A Datalog rule is guarded if its body contains a positive atom (the guard of the rule) in which all free variables of the rule occur.

A generalized literal G is an expression of the form $\forall y_1 \dots y_n (\alpha \rightarrow \beta)$ where α and β are atoms, and $\text{free}(\beta) \subseteq \text{free}(\alpha)$. The free variables $\text{free}(G)$ of G are given by $\text{free}(\alpha) - \{y_1, \dots, y_n\}$.

A Datalog LITE program is a stratified Datalog program whose rules are either monadic or guarded and which contains (unnegated) generalized literals. The notions of guardedness and stratification are extended to generalized literals.

The proposition of the following lemma essentially says that $\text{GF}(\mathcal{D})$ is strictly more expressive than Datalog LITE. The notion of ASM query allows to compare ASMs and Datalog LITE directly. An ASM query (Π_{ASM}, Q) is equivalent to a Datalog LITE query $(\Pi_{\text{Datalog}}, Q)$ if, and only if, for every state $\mathcal{A}$ (over an appropriate vocabulary) and every tuple $\bar{a}$ of elements from the domain of $\mathcal{A}$

$$\bar{a} \in (\Pi_{\text{ASM}}, Q)[\mathcal{A}] \text{ if, and only if, } \bar{a} \in (\Pi_{\text{Datalog}}, Q)[\mathcal{A}]$$

Consequently, we consider equivalence with respect to the result of a computation. Another possibility would be to compare programs stepwisely as we did

it for two ASMs. This means that we consider each state in a computation of an ASM and compare it to the corresponding state in the computation of the Datalog LITE program. The ASM and the Datalog LITE program are equivalent in the case where the states are identical in each step of every computation. But as we compare $\text{GF}(\mathcal{D})$ to a logic fragment (namely, the guarded fragment of first-order fixed-point logic) in the next section where we do not have this notion of equivalence, we restrict to equivalence of queries in this case. This means that we consider equivalence with respect to the result of a computation.

Lemma 7 *Every Datalog LITE query is equivalent to a $\text{GF}(\mathcal{D})$ query. Conversely, there exists a $\text{GF}(\mathcal{D})$ query for which there is no equivalent Datalog LITE query.*

Proof. First, we show that $\text{GF}(\mathcal{D})$ is at least as strong as Datalog LITE. The idea of this part is to translate Datalog LITE programs to ASMs from $\text{GF}(\mathcal{D})$. We construct this translation stepwisely by starting with translating Datalog rules. Here, we have to distinguish between the two types of rules, namely monadic and guarded ones. We then proceed with the translation of basic Datalog programs. Essentially, the translation of a basic Datalog program is the parallel composition of the translations of the contained rules. Finally, we construct from these components a translation for a Datalog LITE program. The stratification requires an additional construct. At the end, we prove strictness via an example consisting of an ASM for which there exists no corresponding Datalog LITE program.

The first step in the translation is to map every Datalog LITE rule R to an ASM rule $a(R) \in \text{GF}(\mathcal{D})$. If R is monadic then it has the form $Px \leftarrow B$ where B is a sequence of literals each with at most one variable. Let S_x^R be the set of all literals in B with free variable x . Then $a(R)$ is the rule

forall $x : \text{true}$ **do**
 if $\bigwedge_{\beta \in S_x^R} \beta \wedge \bigwedge_{y \in \text{free}(R) - \{x\}} \left(\exists y \bigwedge_{\beta \in S_y^R} \beta \right)$ **then** $Px := \text{true}$ **endif**
endforall

Else R is a guarded rule. Let g be the guard of R with $\text{free}(g) = \{\bar{x}\}$, α be its head and B its body. Then $a(R)$ is the rule

forall $\bar{x} : g$ **do**
 if $\bigwedge_{\beta \text{ appears in } B} \beta$ **then** $\alpha := \text{true}$ **endif**
endforall

where $\bar{x}$ is a tuple of variables containing exactly the free variables of g .

This completes the translation of single rule. In the next step, we define the translation of a Datalog program from the translations of the contained rules.

A basic Datalog program D consisting of the rules $R_1, \dots, R_n$ is translated to **do-in-parallel** $a(R_1) \dots a(R_n)$ **enddo**.

Let $D = (D_0, \dots, D_r)$ be a Datalog LITE program whose strata are basic Datalog programs. In addition to those relation and constant symbols from the vocabulary of D , the vocabulary of the resulting ASM contains r nullary boolean constant symbols $\text{computed}_0, \dots, \text{computed}_{r-1}$ marking whether the computation of the i -th stratum is completed. Initially, $\text{computed}_0, \dots, \text{computed}_{r-1}$ are all interpreted by false. D is translated to the ASM defined by

```

do-in-parallel
  if  $\neg \text{computed}_0$  then
    do-in-parallel
       $a(D_0)$ 
      if  $\varphi_0$  then  $\text{computed}_0 := \text{true}$  endif
    enddo
  endif
   $A_1$ 
   $\vdots$ 
   $A_r$ 
enddo

```

where $A_i, i \in \{1, \dots, r\}$ abbreviates

```

if  $\text{computed}_{i-1} \wedge \neg \text{computed}_i$  then
  do-in-parallel
     $a(D_i)$ 
    if  $\varphi_i$  then  $\text{computed}_i := \text{true}$  endif
  enddo
endif

```

and the formulae $\varphi_i, i \in \{0, \dots, r\}$, intuitively expresses that the computation of the i -th stratum is completed. Formally, the formula $\varphi_i, i \in \{1, \dots, r\}$ is given by

$$\bigwedge_{R \in \text{mon}(D_i)} \forall x \left[\left(\bigwedge_{\beta \in S_x^R} \beta \wedge \bigwedge_{y \in \text{free}(R) - \{x\}} \left(\exists y \bigwedge_{\beta \in S_y^R} \beta \right) \right) \rightarrow Px \right] \wedge \\
\bigwedge_{\substack{\alpha \leftarrow B \in \text{guarded}(D_i) \text{ with guard } g \\ \text{free}(g) = \{x_1, \dots, x_k\}}} \forall x_1 \dots x_k \left[g \rightarrow \left(\left(\bigwedge_{\beta \text{ is in } B \text{ but not } g} \beta \right) \rightarrow \alpha \right) \right]$$

where $\text{mon}(D_i)$ denotes the set of monadic rules in D_i and $\text{guarded}(D_i)$ is the set of guarded rules in D_i .

In the rest of this proof, we show that $\text{GF}(\mathcal{D})$ is *strictly* stronger than Datalog LITE. I.e., there exists a $\text{GF}(\mathcal{D})$ query for which there is no equivalent Datalog LITE query. In order to prove strictness, we use the following proposition that has been proved in [4].

Let Π be a Datalog LITE program with head predicates $T_1, \dots, T_r$, and let $T_1^{\mathcal{A}}, \dots, T_r^{\mathcal{A}}$ be the relations computed by Π on an input structure $\mathcal{A}$. The Gaifman graph of the expanded structure $(\mathcal{A}, T_1^{\mathcal{A}}, \dots, T_r^{\mathcal{A}})$ coincides with the Gaifman graph of $\mathcal{A}$.

There exists an ASM $\Pi \in \text{GF}(\mathcal{D})$ whose vocabulary does not contain constant symbols and a state $\mathcal{A}$ of Π such that the Gaifman graph of the state $\mathcal{B}$ with $\mathcal{A} \vdash_{\Pi} \mathcal{B}$ and the Gaifman graph of $\mathcal{A}$ do not coincide. A witness is the ASM $\Pi \in \text{GF}(\mathcal{D})$ defined by **forall** $xy : \text{Exy}$ **do** $\text{Exy} := \text{false}$ **endforall**. $\square$

7 Guarded ASMs are Stronger than Guarded Fixed Point Logic

The most important extension of propositional modal logic (ML) is the modal μ -calculus. It extends ML by least and greatest fixed points. This motivates to extend GF also by least and greatest fixed points leading to the guarded fixed point logic μGF .

μGF is the second mechanism to which we compare $\text{GF}(\mathcal{D})$. It has first been considered in [9]. Further considerations can be found in [6] and [7].

Definition 8 *Guarded fixed point logic μGF is obtained by adding to GF the following rules for constructing fixed point formulae.*

Let W be a k -ary relation variable and $\bar{x} = x_1 \dots x_k$ be a tuples of distinct variables. Further, let $\psi(W, \bar{x})$ be a guarded formula where W appears only positively and not in guards. Moreover, we require that all the free variables of $\psi(W, \bar{x})$ are contained in $\bar{x}$. For such a formula $\psi(W, \bar{x})$, we can construct the formulae

$$\text{lfp}_{\bar{x}, W} \psi(\bar{x}) \text{ and } \text{gfp}_{\bar{x}, W} \psi(\bar{x})$$

The semantics of the fixed point formulae is the following. Given a structure $\mathcal{A}$ and a valuation χ for the free second-order variables in ψ , other than W , the formula $\psi(W, \bar{x})$ defines an operator on k -ary relations $W \subseteq A^k$, namely

$$\psi^{\mathcal{A}, \chi}(W) := \{\bar{a} \in A^k : \mathcal{A}, \chi \models \psi(W, \bar{a})\}$$

As W occurs only positively in ψ , this operator is inductive. Therefore, it has a least fixed point $\text{lfp}(\psi^{\mathcal{A}, \chi})$ and a greatest fixed point $\text{gfp}(\psi^{\mathcal{A}, \chi})$. The semantics of least fixed point formulae is defined by

$$\mathcal{A}, \chi \models [\text{lfp}(\psi^{\mathcal{A}, \chi})] (\bar{a}) \text{ iff } \bar{a} \in \text{lfp}(\psi^{\mathcal{A}, \chi})$$

and

$$\mathcal{A}, \chi \models [\text{gfp}(\psi^{\mathcal{A}, \chi})] (\bar{a}) \text{ iff } \bar{a} \in \text{gfp}(\psi^{\mathcal{A}, \chi})$$

W is the fixed point variable of $[\text{gfp } W \bar{x}. \psi](\bar{x})$ resp. $[\text{lfp } W \bar{x}. \psi](\bar{x})$.

In the remainder of this section, we compare the expressive power of $\text{GF}(\mathcal{D})$ to the expressive power of the guarded fixed point logic.

In [4], it has been proven that every alternation-free sentence in μGF is equivalent to a Datalog LITE query. Conversely, every Datalog LITE query is equivalent to an alternation-free formula in μGF . This proposition and lemma 7 imply the following corollary.

Corollary 9 *Every alternation-free sentence in μGF is equivalent to a $\text{GF}(\mathcal{D})$ query. Conversely, there exists a $\text{GF}(\mathcal{D})$ query for which there is no equivalent alternation free formula in μGF .*

But this lemma can even be sharpened. Namely, it does not only hold for the alternation-free fragment of μGF but for the complete guarded fixed point logic.

The intuitive reason why we can abandon the restriction of alternation-freedom is rather simple. A Datalog LITE program is arranged in strata which can only be processed in a top-down order. Once a stratum has been left, it is not possible to return to it again. This restriction is the main reason for the necessity of alternation-freedom. For $\text{GF}(\mathcal{D})$, we do not have any restriction on the order of executing parts of an ASM and can abandon alternation-freedom.

In order to shorten the proof given in the remainder of this work, we give the following definition in advance.

Definition 10 *Let $\Pi \in \mathcal{D}$ be an ASM and $\mathcal{A}$ be a state of Π .*

1. *Let $(\mathcal{A}_i)_{i \in \mathbb{N}} = \mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \dots$ be the run of Π on $\mathcal{A} = \mathcal{A}_0$. Assume that there exists an $i \in \mathbb{N}$ such that $\mathcal{A}_i$ coincides with $\mathcal{A}_{i+1}$. Let n be the smallest such number. Then $\mathcal{A}_n^*$ denotes the state $\mathcal{A}_n$.*
2. *Furthermore, let $\bar{a}$ be a tuple of elements from the domain of $\mathcal{A}$ and R be a relation symbol whose arity corresponds to the length of $\bar{a}$. Then we say that $\bar{a} \in (\Pi, R)[\mathcal{A}]$ holds iff $\mathcal{A}_n^* \models R\bar{a}$.*

Note that an index as referred to in the above definition does not exist in every run of every ASM in $\mathcal{D}$. But in the following considerations we compare the expressive power of $\text{GF}(\mathcal{D})$ and μGF and because of the inductivity of the operators defined by the formulae in the fixed point constructs, we can restrict our considerations to ASMs where we can find such an index for all runs.

The notion of Datalog LITE query respectively ASM query provides the possibility to compare Datalog LITE respectively ASMs with μGF formulae. Essentially, the vocabulary of the Datalog LITE program respectively the ASM Π is supposed to contain a distinguished relation H that is initially empty. Then the query (Π, H) and the formula φ are said to be equivalent in the case that for every state $\mathcal{A}$ and every tuple $\bar{a}$ of elements from the domain of $\mathcal{A}$

$$\bar{a} \in (\Pi, H)[\mathcal{A}] \text{ iff } \mathcal{A} \models \varphi(\bar{a})$$

holds. Consequently, when the computation halts, the relation H contains exactly those tuples that satisfy the formula φ .

Theorem 11 *Every sentence in μGF is equivalent to a $\text{GF}(\mathcal{D})$ query. Conversely, there exists a $\text{GF}(\mathcal{D})$ query for which there is no equivalent formula in μGF .*

Proof. First, we prove that every sentence in μGF is equivalent to a $\text{GF}(\mathcal{D})$ query. This is done via constructing for an arbitrary μGF -sentence an equivalent $\text{GF}(\mathcal{D})$ -query.

W.l.o.g., we can assume that different quantifications use distinct variables. We can establish this form via simple renamings. If we do not assume this, we would have to distinguish in the following syntactically equal subformulae that occur in different places.

Let ψ be a μGF sentence, and φ a subformula of ψ . If φ is not a sentence then let α_φ be the guard of the innermost quantifier dominating φ . Obviously, $\text{free}(\varphi) \subseteq \text{free}(\alpha_\varphi)$. In the case where φ is a sentence, we can set $\alpha_\varphi := \text{true}$.

We now associate with every subformula φ of ψ an ASM Π_φ and a relation H_φ where H_φ has the same arity as φ . Let $\mathcal{A}$ be the state that results from $\mathcal{A}$ by extending the vocabulary with the new relation symbols introduced in the construction and initializing these relations with the empty set (resp. by false if they are nullary). Then our construction ensures that

$$\bar{a} \in (\Pi_\varphi, H_\varphi)[\mathcal{A}] \text{ iff } \mathcal{A} \models \varphi(\bar{a})$$

for every structure resp. state $\mathcal{A}$ and every tuple $\bar{a}$ that is guarded by α_φ . In the case that φ is a sentence (in particular for $\varphi = \psi$), this means that (Π_φ, H_φ) evaluates to true on $\mathcal{A}$ iff $\mathcal{A} \models \varphi(\bar{a})$.

Furthermore, we associate with every subformula φ of ψ , that is non-atomic or uses a fixed point variable, a relation computed_φ . This relation has the same arity as φ and indicates for a tuple $\bar{a}$ of elements from the domain of the considered structure (with given valuations of the relation variables) whether the computation of the valuation of φ has already been completed (for the tuple $\bar{a}$). For atomic formulae φ not involving a fixed-point variable, $\text{computed}_\varphi \bar{x}$ has to be replaced by true in the following.

In order to simplify the construction, we eliminate the use of gfp by using the duality for greatest and least fixed points:

$$\text{gfp}_{\bar{x}, W} \xi(\bar{x}, W) \equiv \neg \text{lfp}_{\bar{x}, W} \neg \xi(\bar{x}, \neg W)$$

where $\xi(\bar{x}, \neg W)$ is obtained by replacing subformulae $W\bar{t}$ by $\neg W\bar{t}$ in $\xi(\bar{x}, W)$.

Now, we define for every μGF -sentence φ an ASM $\Pi_\varphi \in \text{GF}(\mathcal{D})$ that is equivalent to φ in the already defined sense. Π_φ is defined by induction on φ . We partially omit **do-in-parallel** if it is understood, e.g. inside a forall-rule.

If φ is an atomic formula then Π_φ consists of the single rule

```

forall  $\bar{x} : \varphi(\bar{x})$  do
   $H_\varphi \bar{x} := \text{true}$ 
   $\text{computed}_\varphi \bar{x} := \text{true}$ 
endforall

```

If φ does not use a fixed-point variable then H_φ is initialized by the empty relation. Consequently, H_φ contains exactly those tuples of elements satisfying φ after executing the above updates.

Else, the valuation of φ may change. But we know that all fixed point operators are inductive. Therefore, it is not possible that a tuple is first contained in a relation interpreting a fixed point variable and at some point it is removed except that the fixed point formula with the involved fixed point variable has to be evaluated from new (starting with the empty relation).

If $\varphi = \nu \wedge \vartheta$ then Π_φ is the following rule where $\bar{x}'$ and $\bar{x}''$ are subtuples of $\bar{x}$.

```

do-in-parallel
   $\Pi_\nu$ 
   $\Pi_\vartheta$ 
  forall  $\bar{x}, \bar{y}: \alpha_\varphi(\bar{x}, \bar{y})$  do
    if  $\text{computed}_\nu \bar{x}' \wedge \text{computed}_\vartheta \bar{x}''$  then
      if  $H_\nu \bar{x}' \wedge H_\vartheta \bar{x}''$  then  $H_\varphi \bar{x} := \text{true}$  endif
       $\text{computed}_\varphi \bar{x} := \text{true}$ 
    endif
  endforall
enddo

```

In order to compute the value of a conjunction of two formulae we require that the valuations of both formulae are already known resp. computed. The interpretation of $H_{\nu \wedge \vartheta}$ is the conjunction of the relations H_ν and H_ϑ .

If $\varphi = \neg \vartheta$ then Π_φ is the following rule where $\bar{x}'$ is a subtuple of $\bar{x}$.

```

do-in-parallel
   $\Pi_\vartheta$ 
  forall  $\bar{x}, \bar{y}: \alpha_\varphi(\bar{x}, \bar{y})$  do
    if  $\text{computed}_\vartheta \bar{x}$  then
      if  $\neg H_\vartheta \bar{x}'$  then  $H_\varphi \bar{x} := \text{true}$ 
      else  $H_\varphi \bar{x} := \text{false}$  endif
       $\text{computed}_\varphi \bar{x} := \text{true}$ 
    endif
  endforall
enddo

```

H_ϑ is complemented on the tuples of interest. The set of tuples in which we are interested is defined by α_ϑ . In order to complement H_ϑ , this relation has to be computed on this set of tuples.

If $\varphi = \forall \bar{y}(\alpha_\vartheta \rightarrow \vartheta)$ then Π_φ is the following rule where $\bar{x}'$ is a subtuple of $\bar{x}$, $\bar{x}''$ is a subtuple of $\bar{x}'$ and $\bar{z}'$ is a subtuple of $\bar{z}$

```

do-in-parallel
  forall  $\bar{x}, \bar{y}: \alpha_\varphi(\bar{x}, \bar{y})$  do
     $\Pi_\vartheta$ 
    if  $\forall \bar{z}(\alpha_\vartheta(\bar{x}', \bar{z}) \rightarrow \text{computed}_\vartheta(\bar{x}'', \bar{z}'))$  then
      if  $\forall \bar{z}(\alpha_\vartheta(\bar{x}', \bar{z}) \rightarrow H_\vartheta(\bar{x}'', \bar{z}'))$  then  $H_\varphi \bar{x} := \text{true}$  endif
    endif
  endforall

```

```

        computedφ $\bar{x}' := \text{true}$ 
      endif
    endforall
  enddo

```

When the value of the formula φ is determined, the valuation of ϑ has to be known for all important variable assignments. These are indicated by α_ϑ .

If $\varphi = [\text{lfp}_{W, \bar{x}} \vartheta(W, \bar{x})](\bar{x})$ then, by the induction hypothesis, there is an ASM from $\text{GF}(\mathcal{D})$ computing the update operator defined by $\vartheta(W, \bar{x})$. Let $\xi_1, \dots, \xi_m$ be those subformulae of φ involving the fixed point variable W which do not have a fixed point constructor outermost. Furthermore, let $\rho_1, \dots, \rho_n$ be the other subformulae of φ involving W . That is, those subformulae which contain W and have a fixed point constructor outermost. Let Π'_φ be the following rule.

```

do-in-parallel
   $\Pi_\vartheta$ 
  forall  $\bar{x} : \alpha_\varphi(\bar{x}, \bar{y})$  do
    if computedφ $\bar{x}$  then // computation with current values completed
      if  $\neg(W\bar{x} \leftrightarrow \vartheta(\bar{x}))$  then // fixed point not yet reached
        do-in-parallel
          if  $\vartheta$  then  $W\bar{x} := \text{true}$  endif // update of fixed point variable

          // Compute the valuation of those formulae that use the fixed
          // point variable  $W$ . The valuation of these formulae might have
          // changed as  $W$  has changed. If such a formula has a fixed
          // point variable constructor outermost then the respective
          // fixed point has to be computed new. Until this computation is
          // not finished (i.e., the fixed point is not yet reached) the value
          // of the respectively computed is false for all important tuples.

          forall  $\bar{u}_{\xi_1} : H_{\xi_1}(\bar{u}_{\xi_1})$  do  $H_{\xi_1}(\bar{u}_{\xi_1}) := \text{false}$  endforall
          //  $\xi_i$  has to be computed again with new interpretation of  $W$ 
          forall  $\bar{z}_{\xi_1} : \alpha_{\xi_1}(\bar{x}^{\xi_1}, \bar{z}_{\xi_1})$  do computedξ1 $(\bar{x}^{\xi_1}, \bar{z}'_{\xi_1}) := \text{false}$  endforall
          :
          forall  $\bar{u}_{\xi_m} : H_{\xi_m}(\bar{u}_{\xi_m})$  do  $H_{\xi_m}(\bar{u}_{\xi_m}) := \text{false}$  endforall
          forall  $\bar{z}_{\xi_m} : \alpha_{\xi_m}(\bar{x}^{\xi_m}, \bar{z}_{\xi_m})$  do computedξm $(\bar{x}^{\xi_m}, \bar{z}'_{\xi_m}) := \text{false}$  endforall
          forall  $\bar{z}_{\rho_1} : \alpha_{\rho_1}(\bar{x}^{\rho_1}, \bar{z}_{\rho_1})$  do computedρ1 $(\bar{x}^{\rho_1}, \bar{z}'_{\rho_1}) := \text{false}$  endforall
          :
          forall  $\bar{z}_{\rho_n} : \alpha_{\rho_n}(\bar{x}^{\rho_n}, \bar{z}_{\rho_n})$  do computedρn $(\bar{x}^{\rho_n}, \bar{z}'_{\rho_n}) := \text{false}$  endforall
        enddo
      endif
    endforall
  if  $\forall \bar{x} (\alpha_\varphi(\bar{x}, \bar{y}) \rightarrow (W(\bar{x}) \leftrightarrow \vartheta(\bar{x})))$  then

```

```

forall  $\bar{x} : \alpha_\varphi(\bar{x}, y)$  do  $\text{computed}_\varphi(\bar{x}) := \text{true}$  endforall
endif
enddo

```

where $\bar{x}^{\xi_i}$ and $\bar{x}^{\rho_j}$ ($i \in \{1, \dots, m\}$, $j \in \{1, \dots, n\}$) are subtuples of $\bar{x}$, $\bar{x}^{\xi_i}$ is a subtuple of $\bar{x}^{\xi_i}$ ($i \in \{1, \dots, m\}$), and $\bar{x}^{\rho_j}$ is a subtuple of $\bar{x}^{\rho_j}$ ($j \in \{1, \dots, n\}$). Furthermore $\bar{z}'_{\xi_i}$ is a subtuple of $\bar{z}_{\xi_i}$ ($i \in \{1, \dots, m\}$) and $\bar{z}'_{\rho_j}$ is a subtuple of $\bar{z}_{\rho_j}$ ($j \in \{1, \dots, n\}$).

Π_φ results from Π'_φ by replacing all occurrences of W and H_ϑ by H_φ . Π_φ simulates the computation of the least fixed point stepwisely. The comments given in the ASM explain its work.

After the construction of Π_φ as described above, we have to remove the updates of computed_α for all atomic formulae α not using a fixed point variable resp. we have to replace it by **Skip**. Remember that for atomic formulae φ not using a fixed point variable, $\text{computed}_\varphi \bar{x}$ has to be replaced by **true**.

Initial states are those which satisfy the following three formulae.

$$\begin{aligned}
& \bigwedge_{\xi \text{ is a subformula of } \varphi} \forall \bar{x}_\xi (\neg H_\xi \bar{x}_\xi), \\
& \bigwedge_{\xi \text{ is a subformula of } \varphi} \forall \bar{x}_\xi (\neg \text{computed}_\xi \bar{x}_\xi), \\
& \bigwedge_{\substack{V \text{ is a relation variable} \\ \text{appearing in } \varphi}} \forall \bar{y}_V (\neg V \bar{y}_V).
\end{aligned}$$

This completes the construction of the $\text{GF}(\mathcal{D})$ -query equivalent to φ .

We still have to prove that there exists a $\text{GF}(\mathcal{D})$ query for which there is no equivalent formula in μGF . But this is rather clear. There are mainly two constructs in $\text{GF}(\mathcal{D})$ that can not be simulated in μGF . Each of the constructs alone would already imply the above proposition and therefore, we have at least two possibilities to prove it. We present them both in the remainder of this proof. Each of them exploits one of the additional constructs.

First, $\text{GF}(\mathcal{D})$ admits positive and negative updates. This means that it is allowed to use updates of the form $\alpha := \text{false}$ and updates of the form $\alpha := \text{true}$ both in ASMs from $\text{GF}(\mathcal{D})$. Therefore, the evaluation of the dynamic relations is not always inductive. It even does not need to halt. In contrast to this, the evaluation of a fixed-point variable is always inductive in μGF . As a consequence, such evaluations of dynamic relations can not be simulated in μGF .

A concrete example for such an ASM is the one defined by the program

```

do-in-parallel
  forall  $\bar{x} : R\bar{x}$  do
    if  $\neg Q\bar{x}$  then  $Q\bar{x} := \text{true}$  endif
  endforall

```

```

forall  $\bar{x} : R\bar{x}$  do
  if  $Q\bar{x}$  then  $Q\bar{x} := \text{false}$  endif
endforall
enddo

```

Second, the guards in μGF are not allowed to contain fixed-point variables. There is an extension of μGF (we denote it by μGF^e) that admits to use fixed-point variables in the guards. M. Otto has proven that there is a μGF^e -sentence that is not equivalent to a μGF -sentence. But in $\text{GF}(\mathcal{D})$, dynamic relations are allowed to appear in the guards and with the translation of μGF -formula given in this proof, we have also shown that every sentence in μGF^e is equivalent to a $\text{GF}(\mathcal{D})$ -query. This implies immediately the above proposition. $\square$

References

1. H. Andréka, J. van Benthem, and I. Németi. Modal Languages and Bounded Fragments of Predicate Logic. Technical Report ILLC Research Report ML-96-03, ILLC, 1996.
2. A. Blass and Y. Gurevich. Abstract state Machines Capture Parallel Algorithms. *ACM Transactions on Computation Logic*, 4(4):578–651, October 2003.
3. E. Börger, P. Päppinghaus, and J. Schmid. Report on a Practical Application of ASMs in Software Design. In Y. Gurevich, M. Odersky, and L. Thiele, editors, *International Workshop on Abstract State Machines ASM 2000*, volume 1912 of *LNCS*, pages 361–366. Springer-Verlag, 2000.
4. G. Gottlob, E. Grädel, and H. Veith. Datalog LITE: A deductive query language with linear time model checking. *ACM Transactions on Computational Logic*, 3(1):1–35, 2002.
5. E. Grädel. On the restraining power of guards. *Journal of Symbolic Logic*, 64:1719–1742, 1999.
6. E. Grädel. The decidability of guarded fixed point logic. In J. Gerbrandy, M. Marx, M. de Rijke, and Y. Venema, editors, *JFAK. Essays Dedicated to Johan van Benthem on the Occasion of his 50th Birthday*, CD-ROM. Amsterdam University Press, 1999.
7. E. Grädel. Guarded fixed point logic and the monadic theory of trees. *Theoretical Computer Science*, 2001. To appear.
8. E. Grädel and A. Nowack. Quantum Computing and Abstract State Machines. In *Abstract State Machines - Advances in Theory and Applications*, volume 2589 of *LNCS*, pages 309 – 323. Springer-Verlag, 2003.
9. E. Grädel and I. Walukiewicz. Guarded Fixed Point Logic. In *Proceedings of 14th IEEE Symposium on Logic in Computer Science LICS '99, Trento*, pages 45–54, 1999.
10. Y. Gurevich. Evolving Algebras 1993: Lipari Guide. In E. Börger, editor, *Specification and Validation Methods*, pages 9–36. Oxford University Press, 1995.
11. Y. Gurevich. Sequential Abstract State Machines capture sequential algorithms. *ACM Transactions on Computational Logic*, 1:77–111, 2000.
12. A. Nowack. Deciding the Verification Problem for Abstract State Machines. In *Abstract State Machines - Advances in Theory and Applications*, volume 2589 of *LNCS*. Springer-Verlag, 2003.
13. A. Nowack. Slicing abstract state machines. In *Abstract State Machines 2004. Advances in Theory and Practice*, volume 3052 of *LNCS*. Springer-Verlag, 2004.

19 Automata for Guarded Fixed Point Logics

Dietmar Berwanger and Achim Blumensath

Mathematische Grundlagen der Informatik
RWTH Aachen

19.1 Introduction

The guarded fixed point logics μGF and μCGF introduced in the previous chapter extend the guarded fragments of first-order logic GF and CGF on the one hand and the modal μ -calculus on the other hand. Thereby, the expressive power of the underlying formalisms is increased considerably. On transition systems, for instance, μGF already subsumes the μ -calculus with backwards modalities. Hence, the question arises, whether these logics are still manageable algorithmically. In this chapter we will study the complexity of their satisfiability problems.

As a consequence of the general criterion stated in Theorem 18.23, it follows that the satisfiability problems for μGF and μCGF are decidable. Yet, the argument does not allow us to derive precise complexity bounds for the decision problem. A lower bound can be obtained from the respective results for L_μ and GF. For L_μ the satisfiability problem is EXPTIME-complete [2], whereas for GF it is complete for 2EXPTIME [5]. However, if we consider formulae of bounded width, i.e., with a bounded number of variables, it becomes EXPTIME-complete as well.

Following Grädel and Walukiewicz [6,5] we will prove that even for μCGF , the strongest logic considered, the satisfiability problem is still in 2EXPTIME in the general case, and in EXPTIME for formulae of bounded width. In other words, the fixed point extensions of guarded logics are almost for free in terms of complexity of the satisfiability problem.

Given the expressive power of these logics, this result is rather surprising. For instance, in contrast to L_μ , already the weakest guarded fixed point logic μGF lacks the finite model property. An example of a formula with only infinite models was given in the previous chapter:

$$(\exists xy.Exy) \wedge (\forall xy.Exy)(\exists z.Eyz)[\text{LFP}_{Z,z}(\forall y.Eyz)Zy](z).$$

A crucial model theoretic aspect of guarded logics is their (generalised) tree model property stated in Theorem 18.16. Informally, this asserts that models of guarded formulae can be represented as trees. In [8,4] Vardi and Grädel emphasise that the tree model property seems to be the key feature responsible for the good algorithmic behavior of modal logics because it makes them amenable to automata-theoretic techniques for solving satisfiability and model-checking problems. The generalised tree model property allows us to lift these techniques to guarded logics. In order to decide whether a given formula ψ is satisfiable one can construct two automata: the first one, called *model checking automaton*,

takes an appropriate representation of a structure as input and accepts if and only if the structure satisfies ψ ; the other automaton recognises the set of all appropriate representations. Then, the formula is satisfiable iff the product of these automata recognises a non-empty language.

This scheme outlines the plan of the present chapter. First, we introduce appropriate tree representations of structures in Section 19.2 together with a suitable automata model. For a better understanding we will proceed on two tracks. On the one hand, we define the *unravelling tree* of a structure. The nodes of this tree are associated to the guarded sets of the structure, in such a way that every node has all guarded sets represented in its successors. This rich encoding allows checking of the encoded model by rather simple automata. Moreover, the underlying technique was already discussed in the previous chapter, in the proof of Theorem 18.17. On the other hand, we introduce *decomposition trees* which, being more compact representations, require more sophisticated two-way automata for model checking.

Section 19.3 is dedicated to the construction of the model checking automaton. Starting from input structures encoded as unravelling trees, we define a one-way automaton which, when viewed as a two-way automaton, still recognises the same structures, but in a different encoding, as decomposition trees. At that point, the two tracks of our exposition converge.

Finally, Section 19.4 concludes the reduction by presenting an automaton which recognises valid decomposition trees. Putting all pieces together we are able to derive the desired complexity bounds for the satisfiability problem.

19.2 Requisites

19.2.1 Clique guarded fixed point formulae

When speaking of formulae we always mean μ CGF-formulae as introduced in the previous chapter. To simplify our notation we will, however, omit the clique guards, i.e., instead of $(\exists \bar{x}.\text{clique}(\bar{x}))\eta(\bar{x})$ we will write $\exists \bar{x}.\eta(\bar{x})$ and accordingly for universal formulae.

Furthermore, we will assume that all formulae are *well named* and in *negation normal form*, that is, fixed point variables are defined at most once and negation applies to atomic formulae only. Clearly, every μ CGF-formula can be rewritten to meet these requirements.

A crucial parameter of a formula is its **width** which is defined as the greatest number of free variables occurring in a subformula. Equivalently, a formula has width k iff it can be transformed, by renaming of variables, so that it uses only k variables. In the following we will always assume that every formula of width k is written with the variables $\{x_0, x_1, \dots, x_{k-1}\}$.

19.2.2 Tree representations

In order to use tree automata for model checking and satisfiability we encode structures by trees. Recall that every subformula of a formula of width k can

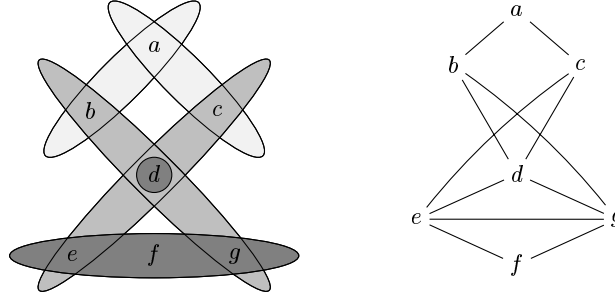


Fig. 19.1. A structure with relations of arity 1, 2, and 3 and its Gaifman graph

refer to at most k structure elements at the same time, which, moreover, have to be guarded. On account of this, we associate to a given structure $\mathfrak{A}$ a tree whose nodes are labelled by the substructures of $\mathfrak{A}$ induced by at most k guarded elements. In addition, the overlap of two adjacent nodes is stored in the label of their common edge.

Let us fix some notation for the remainder of this chapter. The set of guarded subsets of size at most k of a σ -structure $\mathfrak{A}$ is denoted by

$$\Gamma_k(\mathfrak{A}) := \{ K \subseteq A \mid K \text{ is } k\text{-clique-guarded in } \mathfrak{A} \}.$$

The substructures induced by these sets are mapped onto the fixed universe $[k] = \{0, \dots, k-1\}$ and then arranged to form a tree while keeping track of overlaps along the edges. Thus, the nodes of the resulting trees are labelled by the alphabet

$$\Sigma_k := \{ \mathfrak{C} \mid \mathfrak{C} \text{ is a } \sigma\text{-structure over a universe } C \subseteq [k] \}$$

while the edges are labelled by subsets of $[k]$. We call trees labelled by these alphabets shortly **k -type trees**. When we speak about a D -edge, we mean an edge labelled with $D \subseteq [k]$, and a D -neighbour or D -successor of a node is a neighbour respectively a successor along some D -edge.

Definition 19.1. For a given width k , the **k -unravelling tree** of a structure $\mathfrak{A}$ is the k -type tree T over the set of nodes $\Gamma_k(\mathfrak{A})^*$ labelled as follows:

- (i) The root of T is labelled with the empty structure $(\emptyset, \sigma)$ and all outgoing edges are labelled with $\emptyset$.
- (ii) Every node $v \in \Gamma_k(\mathfrak{A})^* K$ is labelled with an isomorphic copy $\mathfrak{C}$ of $\mathfrak{A}|_K$, the restriction of $\mathfrak{A}$ to $K \in \Gamma_k(\mathfrak{A})$.
- (iii) If $\pi : \mathfrak{A}|_K \rightarrow \mathfrak{C}$ and $\pi' : \mathfrak{A}|_{K'} \rightarrow \mathfrak{C}'$ are isomorphisms labelling, respectively, a node $v \in \Gamma_k(\mathfrak{A})^* K$ and its successor $v' = vK'$, then π and π' agree on $K \cap K'$ and the edge (v, v') is labelled with $\pi(K \cap K')$.

Remark 19.2. It is easy to see that for every D -edge (v, v') of an unravelling tree T the following conditions hold:

- (i) **Consistency:** the labels $\mathfrak{C}$ of v and $\mathfrak{C}'$ of v' agree on D , that is, $\mathfrak{C}|_D = \mathfrak{C}'|_D$.
- (ii) **Completeness:** for any $H \subseteq [k]$ the H -successors of v and v' agree on $D \cap H$, i.e., there is a one-to-one map assigning to each H -successor w of v an H -successor w' of v' such that the labels of w and w' agree on $D \cap H$.

Generally, we call a k -type tree **consistent**, if it satisfies the first condition. Let us now look at the relationship between a tree representation and the encoded structure.

Definition 19.3. Given a consistent k -type tree T , consider the disjoint sum of its node labels,

$$\mathfrak{D} := \bigcup \{ (\mathfrak{C}, v) \mid \mathfrak{C} \text{ is the label of } v \in T \}.$$

Let $\sim$ be the least equivalence relation on the universe of $\mathfrak{D}$ with

$$(i, v) \sim (i, v') \quad \text{if } v' \text{ is a successor of } v \text{ and } i \text{ is in the label of } (v, v').$$

Then, by consistency of T , $\sim$ is a congruence relation on $\mathfrak{D}$. We call the quotient $\mathfrak{D}/\sim$ the structure **recovered** from T .

Definition 19.4. The **k -unravelling** $\mathfrak{A}^{(k)}$ of a structure $\mathfrak{A}$ is the structure recovered from the k -unravelling tree of $\mathfrak{A}$.

Since μCGF is invariant under guarded bisimulation (see [5]), it follows that sentences of width up to k cannot distinguish between a structure $\mathfrak{A}$ and its k -unravelling $\mathfrak{A}^{(k)}$.

Proposition 19.5. *Every structure $\mathfrak{A}$ is k -clique bisimilar to its k -unravelling $\mathfrak{A}^{(k)}$. That is, for all μCGF -sentences ψ of width at most k we have*

$$\mathfrak{A} \models \psi \quad \text{iff} \quad \mathfrak{A}^{(k)} \models \psi.$$

If we recall the notion of tree decomposition of a structure introduced in the previous chapter we can easily establish the following connection.

Proposition 19.6. *A k -type tree T is a tree decomposition of some structure $\mathfrak{A}$ iff the structure recovered from T is isomorphic to $\mathfrak{A}$.*

This relationship suggests tree decompositions as candidates for structure representations.

Definition 19.7. For a given width k , a **k -decomposition tree** of a structure $\mathfrak{A}$ is a k -type tree T where

- (i) for every $K \in \Gamma_k(\mathfrak{A})$ there is a node labelled with (an copy of) $\mathfrak{A}|_K$;
- (ii) the labels of any two nodes connected by a D -edge agree on D ;

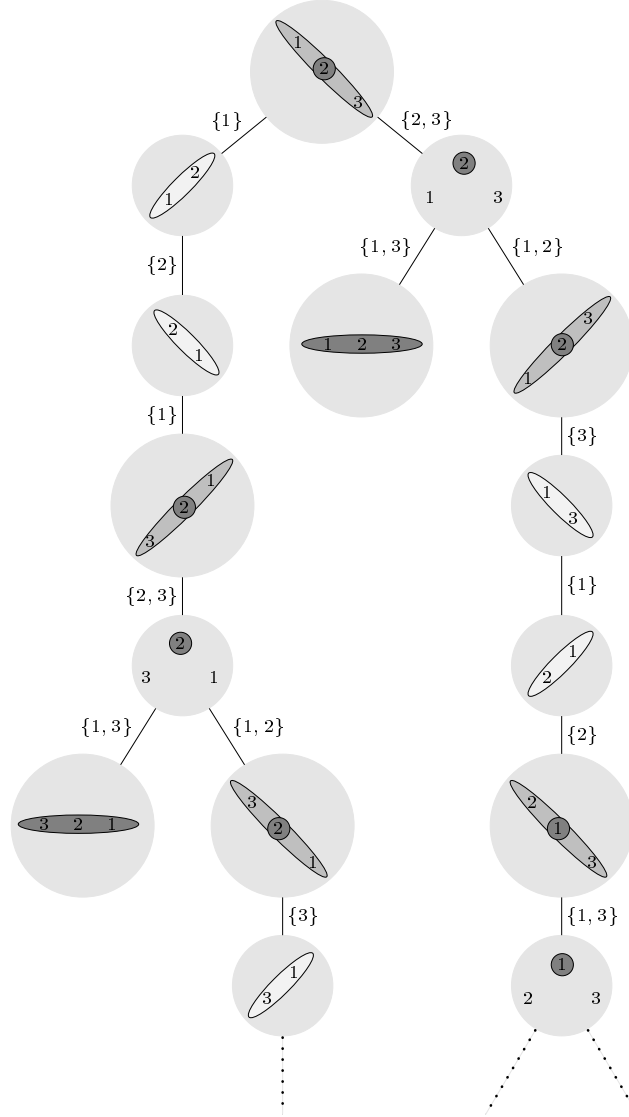


Fig. 19.2. A k -decomposition tree of the structure in Fig. 19.2.2

- (iii) every node v is labelled with $\mathfrak{A}|_K$ for some $K \in \Gamma_k(\mathfrak{A})$ via an isomorphism π . Moreover, for each $K' \in \Gamma_k(\mathfrak{A})$ there is a node v' labelled with $\mathfrak{A}|_{K'}$, such that all edges on the path between v and v' include $\pi(K \cap K')$ in their labels.

Remark 19.8. (i) The k -unravelling tree of a structure is also a k -decomposition tree of that structure.
(ii) Each k -decomposition tree of a structure $\mathfrak{A}$ induces a subtree in the k -unravelling tree of $\mathfrak{A}$.

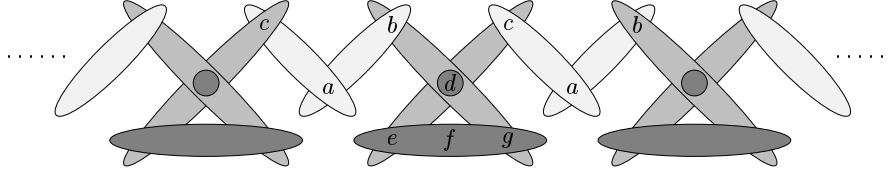


Fig. 19.3. The structure recovered from the decomposition tree in Fig. 19.2.2

It is an easy exercise to show that the process of k -decomposing a structure preserves its properties up to bisimulation, yielding a more compact representation than unravelling does.

Proposition 19.9. *Given a structure $\mathfrak{A}$, let $\mathfrak{A}'$ be the structure recovered from a k -decomposition tree of $\mathfrak{A}$. Then $\mathfrak{A}$ and $\mathfrak{A}'$ are clique- k -bisimilar.*

19.2.3 The automata model

We employ alternating automata that work on trees where nodes and edges are labelled.

Definition 19.10. An **alternating tree automaton** over a node alphabet Σ and an edge alphabet Δ is given by a tuple

$$\mathcal{A} = (Q, \Sigma, \Delta, \delta, q_I, \Omega)$$

where $Q = Q_0 \cup Q_1$ is the set of existential and universal states, q_I designates the initial state, $\Omega : Q \rightarrow \omega$ is a parity condition and

$$\delta : Q \times \Sigma \rightarrow \mathcal{P}(\Delta \times Q)$$

is the transition function. The pairs $(d, q) \in \Delta \times Q$ are called transitions.

We define the behaviour of such automata by way of games.

Definition 19.11. Let $\mathcal{A} = (Q, \Sigma, \Delta, \delta, q_I, \Omega)$ be an automaton and T an appropriately labelled tree. The game $\mathcal{G}(\mathcal{A}, T)$ associated to $\mathcal{A}$ and T is the parity game with positions $Q \times T$ and acceptance condition Ω played as follows.

Every play starts in state q_I at the root of T . Assume that the play reached some position (q, v) where the node v is labelled with c . If q belongs to Q_0 , Player 0 can move to a position (q', v') if

- (i) there is a transition $(d, q') \in \delta(q, c)$ and
- (ii) v' is a d -successor of v .

The moves of Player 1 are defined analogously.

The language $L(\mathcal{A})$ accepted by a tree automaton $\mathcal{A}$ is the set of all trees T , such that Player 0 has a winning strategy in the game $\mathcal{G}(\mathcal{A}, T)$.

Usually, automata are defined as devices scanning their input only in one direction. However, for our purpose it is convenient to allow them to move backwards and remain still as well.

Definition 19.12. An **alternating two-way tree automaton** is given in the same way as a (one-way) alternating automaton,

$$\mathcal{A}^2 = (Q, \Sigma, \Delta, \delta, q_I, \Omega)$$

where acceptance is defined in a different way. The game $\mathcal{G}(\mathcal{A}^2, T)$ associated to a two-way automaton $\mathcal{A}^2$ and a tree T is the parity game obtained as in Definition 19.11, but replacing rule (ii) with

(ii') either $v' = v$ or v' is a d -neighbour of v .

The language $L(\mathcal{A}^2)$ accepted by a two-way tree automaton $\mathcal{A}^2$ is the set of all trees T such that Player 0 has a winning strategy in the game $\mathcal{G}(\mathcal{A}^2, T)$.

19.3 Model Checking

The results presented in Chapter 14 and 10 reveal a close relationship between alternating automata and games on the one side, and logical formalisms on the other side. The automaton constructed in Section 10.3 for L_μ translates first-order operations into state transitions, while fixed point predicates are encoded as priorities.

In a similar way, we will construct automata for μCGF . But unlike L_μ , where a formula is evaluated at a single node of a transition system, a μCGF -formula with several free variables may involve several structure elements. Since these elements have to be clique-guarded, they appear together in the label of some node in the unravelling (or, decomposition) tree. To allow our automaton to access the structure in the node labels of the input tree, its states will contain two components: a subformula, and an assignment of the variables appearing free therein.

The *closure* $\text{cl}(\psi)$ of a formula ψ is the set consisting of all subformulae of ψ together with the formulae true and false.

Definition 19.13. To any formula $\psi \in \mu\text{CGF}$ of width k we associate the automaton $\mathcal{A}_\psi = (Q, \Sigma_k, \mathcal{P}([k]), \delta, q_I, \Omega)$ over k -type trees where the state set

$$Q := \{ (\varphi, \beta) \mid \varphi \in \text{cl}(\psi) \text{ and } \beta : \{x_0 \dots x_{k-1}\} \rightarrow [k] \}$$

is partitioned into existential and universal states by

$$\begin{aligned} Q_0 &:= \{ (\varphi, \beta) \mid \varphi = \text{false}, \text{ or } \varphi = \eta \vee \vartheta, \text{ or } \varphi = \exists \bar{y}. \eta \}, \text{ and} \\ Q_1 &:= Q \setminus Q_0. \end{aligned}$$

The initial state is $q_1 = (\psi, \emptyset)$ where $\emptyset$ stands for the void assignment.

It remains to specify the transition function. To simplify our notation we use expressions $\langle H \rangle S$ with $H \subseteq [k]$ and $S \subseteq Q$ to denote the set of transitions $\{ D \subseteq [k] \mid H \subseteq D \} \times S$. In particular, when we refer to the universe of $\mathfrak{C}$ we write $\langle \cdot \rangle S$ instead of $\langle C \rangle S$. Furthermore, omitting parenthesis, we simply write $\delta(\varphi, \beta, \mathfrak{C})$ instead of $\delta((\varphi, \beta), \mathfrak{C})$.

- (i) If $\varphi = \text{true}$ or $\varphi = \text{false}$ then $\delta(\varphi, \beta, \mathfrak{C}) = \emptyset$.
- (ii) If φ is a σ -atom or a negated σ -atom then

$$\delta(\varphi, \beta, \mathfrak{C}) = \begin{cases} \langle \cdot \rangle \{(\text{true}, \emptyset)\} & \text{if } \mathfrak{C}, \beta \models \varphi, \\ \langle \cdot \rangle \{(\text{false}, \emptyset)\} & \text{if } \mathfrak{C}, \beta \not\models \varphi. \end{cases}$$

- (iii) If $\varphi = \eta \wedge \vartheta$ or $\varphi = \eta \vee \vartheta$ then

$$\delta(\varphi, \beta, \mathfrak{C}) = \langle \cdot \rangle \{(\eta, \beta), (\vartheta, \beta)\}.$$

- (iv) If $\varphi(\bar{x}) = \text{FP}_{T\bar{y}}(\eta)(\bar{x})$ then

$$\delta(\varphi, \beta, \mathfrak{C}) = \langle \cdot \rangle \{(\eta, \beta)\}.$$

- (v) If $\varphi(\bar{x}) = T\bar{x}$ and $\text{FP}_{T\bar{y}}(\eta)(\bar{x})$ is the unique definition of T in ψ then

$$\delta(\varphi, \beta, \mathfrak{C}) = \langle \cdot \rangle \{(\eta, \beta)\}.$$

- (vi) If $\varphi(\bar{x}) = \exists \bar{y}. \eta(\bar{x}, \bar{y})$ or $\varphi(\bar{x}) = \forall \bar{y}. \eta(\bar{x}, \bar{y})$ then

$$\delta(\varphi, \beta, \mathfrak{C}) = \langle \cdot \rangle \{(\eta, \beta') \mid \beta'|_{\bar{x}} = \beta|_{\bar{x}}\} \cup \langle \beta(\bar{x}) \rangle \{(\varphi, \beta)\}.$$

Finally, if the fixed point variables of ψ occur in the order $Z_1, \dots, Z_n$ the parity condition is given by

$$\Omega(\varphi, \beta) := \begin{cases} 2i & \varphi = Z_i \bar{x} \text{ and } Z_i \text{ is a GFP-variable,} \\ 2i + 1 & \varphi = Z_i \bar{x} \text{ and } Z_i \text{ is an LFP-variable,} \\ 2n + 4 & \varphi = \forall \bar{y}. \eta, \\ 2n + 3 & \varphi = \exists \bar{y}. \eta, \\ 2n + 2 & \text{otherwise.} \end{cases}$$

The automaton works in a similar way as the L_μ -automata defined in Section 10.3: disjunctions are decomposed by Player 0, conjunctions by Player 1 and fixed points are regenerated. Atomic statements are verified locally and terminate the run. Acceptance of infinite runs is determined by the priority function which reflects the nesting and type of fixed point definitions. Note that, except when dealing with quantifiers, the automaton changes only the formula component of its states, while the variable assignment remains the same. Moreover, the $\langle \cdot \rangle$ -transitions allow to move only to successors that retain the structure information of the current node.

To understand the handling of quantification, consider, e.g., an existential formula $\varphi(\bar{x}) = \exists \bar{y}.\eta(\bar{x}, \bar{y})$. Player 0 may use a transition from $\langle \beta(\bar{x}) \rangle \{(\varphi, \beta)\}$ to proceed to a successor that retains the structure living on elements currently assigned to the free variables $\bar{x}$. In this way, he can reassign the quantified variables $\bar{y}$ to elements of the chosen successor. After such a move, the formula in the new state is still φ and Player 0 is again in turn to move. But, as existential formulae have odd priority, he can reiterate these moves only finitely many times and must then take a transition of the form $\langle \cdot \rangle \{(\eta, \beta') \mid \beta'|_{\bar{x}} = \beta|_{\bar{x}}\}$.

Given an input tree that k -unravels a structure $\mathfrak{A}$, the structures labelling the nodes are all induced by k -cliques in $\mathfrak{A}$. Moreover, from each node (a copy of) every other k -clique of $\mathfrak{A}$ is accessible within one move.

It remains to prove that our construction is correct, that is, that we can use the automaton defined above to solve the model checking problem for μCGF .

Proposition 19.14. *Given a formula ψ of width k and a structure $\mathfrak{A}$, the automaton $\mathcal{A}_\psi$ accepts the k -unravelling tree of $\mathfrak{A}$ iff $\mathfrak{A} \models \psi$.*

Proof. It is convenient to argue in terms of games. Model checking games for μCGF were introduced in [1] as a generalisation of the model checking games for L_μ . Although defined for finite structures, the extension of these games to the transfinite case is straightforward.

Let T be the k -unravelling tree of the structure $\mathfrak{A}$. We will show that the game which determines acceptance of T by the automaton $\mathcal{A}_\psi$ is essentially the model checking game associated to $\mathfrak{A}$ and ψ .

Let $\mathcal{G}$ be the acceptance game $\mathcal{G}(\mathcal{A}_\psi, T)$. We can simplify this game by collapsing positions which share the same formula and map its free variables to the same part of the structure.

Recall that any node v of T is labelled via some isomorphism π . Furthermore, at every position (φ, β, v) in a play of $\mathcal{G}$, the image of π includes the image of the assignment β . Thus, we can define a mapping from the positions of $\mathcal{G}$ to $\{(\varphi, \chi) \mid \varphi \in \text{cl}(\psi) \text{ and } \chi: \{x_0 \dots x_{k-1}\} \rightarrow A\}$ as follows:

$$\hat{\cdot}: (\varphi, \beta, v) \mapsto (\varphi, \pi^{-1} \circ \beta).$$

By the construction of $\mathcal{G}$, we can easily verify that this mapping induces a congruence relation $\hat{\sim}$ among the positions of $\mathcal{G}$,

$$(\varphi, \beta, v) \hat{\sim} (\varphi, \beta, v') \quad \text{iff} \quad (\widehat{\varphi, \beta, v}) = (\widehat{\varphi, \beta, v'}),$$

which is also a bisimulation on $\mathcal{G}$.

Consider now the (strong homomorphic) image $\hat{\mathcal{G}}$ of $\mathcal{G}$ under $\hat{\cdot}$. On the one hand, $\mathcal{G}$ and $\hat{\mathcal{G}}$ are bisimilar via $\hat{\cdot}$ and, consequently, the same player has a winning strategy in both plays. On the other hand, $\hat{\mathcal{G}}$ is almost the model checking game $\mathcal{G}' = \mathcal{G}(\mathfrak{A}, \psi)$ as defined in [1]. The only difference arises at positions (φ, χ) where φ is an existential or universal formula, say $\varphi = \exists \bar{y}.\eta(\bar{x}, \bar{y})$. Then, the model checking game allows moves to (η, χ') with χ' such that

- (i) χ and χ' agree on the values of $\bar{x}$ and

(ii) $\mathfrak{A}, \chi' \models \text{clique}(\bar{x}, \bar{y})$,

whereas in $\widehat{\mathcal{G}}$ the legal moves go either to (φ, χ') with χ' as above, or to (η, χ) .

Nevertheless, we will show that the same player wins both $\mathcal{G}'$ and $\widehat{\mathcal{G}}$. If Player 0 has a winning strategy in the model checking game $\mathcal{G}'$, he can also play this strategy in $\widehat{\mathcal{G}}$, as long as no existential formula is met. Otherwise, at positions (φ, χ) as above, he can imitate the move to the position (η, χ') he would perform in $\mathcal{G}'$ by taking two steps:

- (i) move to (φ, χ') ; this is possible since, for every χ' agreeing with χ on the free variables of φ , the position (φ, χ') is reachable from (φ, χ) in one step.
- (ii) At (φ, χ') it's still Player 0 turn: move to (η, χ') .

Towards a contradiction, let us assume that Player 1 wins this play. Then, after any universal formula $\varphi = \forall \bar{y} \eta(\bar{x}, \bar{y})$ occurring in the play, there can follow only finitely many positions with φ until Player 1 chooses some position (η, χ') ; otherwise he would lose with the highest even priority. But then, Player 1 also wins by choosing (φ, χ') right from position (φ, χ) and proceeding with (η, χ') . However, these two moves translate into one move in the corresponding play of $\mathcal{G}'$ which leads Player 1 to a win in $\mathcal{G}$ despite Player 0's winning strategy, which is not possible. This concludes our proof that a player has a winning strategy in the model checking game iff he has one in the acceptance game.

The correctness of our construction relies on the fact that the input trees are complete in the sense of Remark 19.2 (ii). That is, if the current node is labelled by a k -clique of the represented structure, then every other k -clique appears in the label of some successor node. Unfortunately, it is very hard to check whether a given tree satisfies this property. By letting $\mathcal{A}_\psi$ run as a two-way automaton $\mathcal{A}_\psi^2$, we can relax this requirement and claim instead that every k -clique shall be reachable via a finite path from the current node.

Proposition 19.15. *Given a formula ψ of width k and a structure $\mathfrak{A}$, let T be a k -decomposition tree of $\mathfrak{A}$. Then the automaton $\mathcal{A}_\psi^2$ accepts T iff $\mathfrak{A} \models \psi$.*

Proof. The idea is to show that $\mathcal{A}_\psi^2$ runs on T in a similar way as its one-way variant does on the k -unravelling tree T' of $\mathfrak{A}$. Towards this we will transform the acceptance game $\mathcal{G}(\mathcal{A}_\psi^2, T)$ by introducing shortcuts into a game which is bisimilar to the acceptance game $\mathcal{G}(\mathcal{A}, T')$ of the one-way automaton.

Let $\mathcal{G}^*$ be the least game extending $\mathcal{G} := \mathcal{G}(\mathcal{A}_\psi^2, T)$ by new transitions in such a way that, whenever there are two transitions

$$(\varphi, \beta, v) \rightarrow (\varphi, \beta, v') \rightarrow (\varphi, \beta, v'')$$

in $\mathcal{G}^*$, the shortcut $(\varphi, \beta, v) \rightarrow (\varphi, \beta, v'')$ is also a transition in $\mathcal{G}^*$.

Observe that the new transitions just shortcut a sequence of steps in the original game, all performed by the same player. To see that this does not change the winning partitions, assume, towards a contradiction, that Player 1 has a winning strategy for $\mathcal{G}^*$ while Player 0 has one for $\mathcal{G}$. All moves in $\mathcal{G}$ are still

available in $\mathcal{G}^*$, so Player 0 can apply his winning strategy for $\mathcal{G}$ in the play π of $\mathcal{G}^*$ against the winning strategy of Player 1. Let us now look at the play in $\mathcal{G}$ in which both players move like in π except at positions (φ, β, v) where Player 1 used a shortcut to, say (φ, β, v') , for φ a universal formula. At that point, Player 1 can move step by step via finitely many positions (φ, β, w) along the path leading to the destination of the shortcut. From there, the play proceeds like in π . Clearly, Player 1 wins this play in $\mathcal{G}$ in contradiction to our assumption on Player 0's winning strategy.

The mapping $\hat{\cdot}$ which was defined in the proof of Proposition 19.14 can be applied to the positions of $\mathcal{G}^*$. It induces a congruence relation on $\mathcal{G}^*$ and, as such, a bisimulation between $\mathcal{G}^*$ and its strong homomorphic image $\hat{\mathcal{G}}^*$. This image is precisely the game $\hat{\mathcal{G}}(\mathfrak{A}_\psi, T')$ which is bisimilar to $\mathcal{G}(\mathfrak{A}_\psi, T')$.

Accordingly, the automaton $\mathcal{A}_\psi^2$ accepts the k -decomposition tree T iff $\mathcal{A}_\psi$ accepts the k -unravelling tree T' .

19.4 Satisfiability

The model checking automata introduced above operate correctly on inputs which represent structures. But in order to solve the satisfiability problem this does not suffice. We need to make sure that all inputs which do not represent structures are rejected.

Checking representation validity. From a given a k -type tree T , we can recover a structure according to Definition 19.3, only if T is consistent, that is, if every node agrees with its D -neighbours on the D -part of its label.

Provided T is consistent, let $\mathfrak{A}$ be the recovered structure. Now T is a k -decomposition tree of $\mathfrak{A}$ iff every node label of T induces a clique in $\mathfrak{A}$. This is crucial, since the model-checking automaton assumes that all elements appearing in the label of its input represent clique-guarded elements of the structure. Another way to formulate this condition is: For every node v and every pair of elements $\{i, j\} \subseteq [k]$ in its label, there is a node v' in which i and j are guarded by an atom and all edges on the path between v and v' include $\{i, j\}$ in their labels.

Now, we build an automaton that checks the above two conditions.

Definition 19.16. For every width k , we construct a two-way automaton $\mathcal{A}_k^2 = (Q, \Sigma_k, \mathcal{P}([k]), \delta, \text{check}, \Omega)$ over k -type trees whose set of states is partitioned into

$$\begin{aligned} Q_0 &= \{\text{false}\} \cup [k]^2 \quad \text{and} \\ Q_1 &= \{\text{true}, \text{check}\} \cup \{R\bar{a} \mid R \in \sigma, \bar{a} \subseteq [k]\}. \end{aligned}$$

In state check the automaton allows Player 1 to move freely on the input tree to reach a node where either the consistency or the guardedness condition may be violated. At that event, state $R\bar{a}$ records the loss of the atom $R\bar{a}$ along

an edge that preserves $\bar{a}$ while the states $(i, j) \in [k]^2$ indicate the search for witnesses to the guardedness of i and j . The transitions are as follows.

$$\begin{aligned} \delta(\text{check}, \mathfrak{C}) = & \langle \emptyset \rangle \{ \text{check} \} \cup \bigcup \{ \langle \bar{a} \rangle \{ R\bar{a} \} \mid \mathfrak{C} \models R\bar{a} \} \\ & \cup \langle \cdot \rangle \{ (i, j) \mid \mathfrak{C} \not\models \text{clique}(i, j) \}. \end{aligned}$$

At a node where the elements i and j are not guarded, Player 1 can challenge his opponent to find a node where $\{i, j\}$ appear guarded, along a path where these elements persist in the edge labels.

$$\delta((i, j), \mathfrak{C}) = \begin{cases} \langle \cdot \rangle \{ \text{true} \} & \mathfrak{C} \models \text{clique}(i, j), \\ \langle \{i, j\} \rangle \{ (i, j) \} & \text{otherwise.} \end{cases}$$

Also, Player 1 may pick a currently valid atomic fact to check whether it is indeed preserved along the edges that contain all involved elements in their label.

$$\delta(R\bar{a}, \mathfrak{C}) = \begin{cases} \langle \cdot \rangle \{ \text{true} \} & \mathfrak{C} \models R\bar{a}, \\ \langle \cdot \rangle \{ \text{false} \} & \text{otherwise.} \end{cases}$$

If the player agree on a local test, the run is finite: $\delta(\text{true}) = \delta(\text{false}) = \emptyset$.

On an infinite run, the automaton assumes forever either the state check or some state (i, j) . Since in the first case Player 0 should win, we set $\Omega(\text{check}) = 0$. In the second case, instead, Player 0 should lose, because he does not provide a witness to the guardedness of i and j after a finite number of steps. To enforce that, we set $\Omega((i, j)) = 1$ for all $(i, j) \in [k]^2$.

It is easy to see that the above checks ensure the consistency and the guardedness of the input tree.

Lemma 19.17. *The automaton $\mathcal{A}_k^2$ recognises the set of k -decomposition trees of all σ -structures.*

Reduction to the emptiness problem. After having constructed an automaton $\mathcal{A}_\psi^2$ for the model checking of a tree representation and an automaton $\mathcal{A}_k^2$ to check the validity of the input tree, we can build the product automaton $\mathcal{B}_\psi^2 := \mathcal{A}_\psi^2 \times \mathcal{A}_k^2$ which recognises precisely the set of k -decomposition trees of all models of ψ . In this way, the satisfiability problem for ψ is reduced to the emptiness problem for $\mathcal{B}_\psi^2$.

Proposition 19.18. *A μ CGF formula ψ is satisfiable iff $L(\mathcal{B}_\psi^2) \neq \emptyset$.*

Emptiness of two-way automata. In order to establish the complexity of the emptiness problem for our automata model we will reduce it to the two-way automata introduced by Vardi [9] defined for input trees of bounded branching degree.

Lemma 19.19. *Any two-way automaton recognising a non-empty language, accepts some tree whose degree is bounded by the size of its state set.*

Proof. Let $\mathcal{A}^2$ be a two-way automaton accepting some input tree T . Hence, Player 0 has a winning strategy in the parity game $\mathcal{G}(\mathcal{A}^2, T)$ and, by [3,7], even a memoryless one: $f : Q_0 \times T \rightarrow Q \times T$. For any node $v \in T$, let $S(v)$ be the set of nodes targeted by f at some position with v :

$$S(v) := \{v' \in T \mid f(q, v) \in Q \times \{v'\} \text{ for some } q \in Q_0\}.$$

Consider now the tree T' obtained from T by discarding at every node v those successors which are not in $S(v)$. Since $|S(v)| \leq |Q_0|$, this yields a tree of branching degree bounded by $|Q_0| \leq |Q|$. Moreover, f is still a winning strategy in $\mathcal{G}(\mathcal{A}^2, T')$. In other words, the automaton $\mathcal{A}^2$ also accepts T' .

19.5 Complexity

Since Vardi's automata work on trees with unlabelled edges, we have to remove the edge labels and place them into their target node. Then, our automaton has to verify the validity of taken transitions, thus, requiring a blow-up of its state set by the size of the edge alphabet. Taking into account this modification, we can transfer the complexity of the emptiness test of Vardi's automata to our model.

Theorem 19.20. *The emptiness of a two-way alternating automaton with s states and t edge symbols can be decided in time $2^{O((st)^2)}$.*

For the computations in the remainder of this chapter, let us fix a formula ψ of size n and width k . Note that, the number of states of the automata $\mathcal{A}_\psi^2$ and $\mathcal{A}_k^2$ is bounded by $O(n \cdot k^k)$. Accordingly, their product $\mathcal{B}_\psi^2$ has at most $O(n^2 k^{2k})$ states. From Lemma 19.19 we can now infer a stronger variant of the tree model property for μCGF .

Proposition 19.21. *Any satisfiable μCGF -formula of width k has a model with a tree decomposition of width at most $k - 1$ and branching degree bounded by $O(n^2 k^{2k})$.*

By Theorem 19.20, the reduction of the satisfiability problem for ψ to the emptiness of $\mathcal{B}_\psi^2$ yields the following complexity bounds:

$$2^{O((n \cdot k^k)^4)} = 2^{O(n^4 2^{4k \log k})} = 2^{2^{O(n)}}.$$

When k is bounded by a constant, the above expression boils down to $2^{O(n^4)}$.

Since the complexity results on CGF quoted in the introduction of this chapter imply hardness of this bounds, we can state:

Theorem 19.22. *The satisfiability problem for μCGF is 2EXPTIME-complete in the general case. For clique guarded fixed point sentences of bounded width it is EXPTIME-complete.*

Literature

1. D. BERWANGER AND E. GRÄDEL, *Games and model checking for guarded logics*, in Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR 2001, vol. 2250 of Lecture Notes in Artificial Intelligence, Springer-Verlag, 2001, pp. 70–84.
2. E. A. EMERSON AND C. S. JUTLA, *The complexity of tree automata and logics of programs (extended abstract)*, in Proceedings of the 29th Annual Symposium on Foundations of Computer Science, FoCS '88, IEEE Computer Society Press, 1988, pp. 328–337.
3. ———, *Tree automata, mu-calculus and determinacy (extended abstract)*, in Proceedings of the 32nd Annual Symposium on Foundations of Computer Science, FoCS '91, IEEE Computer Society Press, 1991, pp. 368–377.
4. E. GRÄDEL, *On the restraining power of guards*, Journal of Symbolic Logic, 64 (1999), pp. 1719–1742.
5. ———, *Guarded fixed point logics and the monadic theory of countable trees*, Theoretical Computer Science, 288 (2002), pp. 129 – 152.
6. E. GRÄDEL AND I. WALUKIEWICZ, *Guarded fixed point logic*, in Proceedings of the 4th Annual IEEE Symposium on Logic in Computer Science, LICS '99, IEEE Computer Society Press, 1999, pp. 45–54.
7. A. W. MOSTOWSKI, *Games with forbidden positions*, Tech. Rep. 78, Instytut Matematyki, Uniwersytet Gdański, Poland, 1991.
8. M. Y. VARDI, *Why is modal logic so robustly decidable?*, in Descriptive Complexity and Finite Models: Proceedings of a DIMACS Workshop, vol. 31, American Mathematical Society, 1996, pp. 149–184.
9. ———, *Reasoning about the past with two-way automata.*, in Proceedings of the 25th International Colloquium on Automata, Languages and Programming, ICALP '98, vol. 1443 of Lecture Notes in Computer Science, Springer-Verlag, 1998, pp. 628–641.

Counting Quantifiers and the Stellar Fragment

Ian Pratt-Hartmann
Department of Computer Science,
University of Manchester *

Abstract

This paper defines a class of clauses which we call the *stellar fragment*, and establishes that both the satisfiability and finite satisfiability problems for this fragment are complete for nondeterministic exponential time. It is shown that the stellar fragment is closely related to the familiar two-variable fragment of first-order logic with counting quantifiers, C^2 . Our results thus demonstrate that finite satisfiability of C^2 -formulas is decidable in nondeterministic exponential time. Our method also yields a much easier proof of the previously known result that the satisfiability of C^2 -formulas is decidable in nondeterministic exponential time.

1 Background

The two-variable fragment with counting quantifiers, C^2 , is the set of function-free first-order formulas containing at most two variables, with the numerical quantifiers $\exists_{<n}x$ permitted. Although C^2 lacks the finite model property, satisfiability of C^2 -formulas is decidable. This was first shown by Grädel, Otto and Rosen [2], and by Pacholski, Szwast and Tendera [4] (with a preliminary version appearing in [3]), where a NEXPTIME complexity bound is obtained. Grädel, *et al.* also report the decidability of the finite satisfiability problem for C^2 , but give no upper complexity bounds.

This paper defines a class of clauses which we call the *stellar fragment*, and establishes, via a reduction to integer programming, that the satisfiability and finite satisfiability problems for this fragment are both NEXPTIME-complete. It is also shown, using techniques from resolution theorem-proving, that the stellar fragment has essentially the same expressive power as C^2 . Our results demonstrate that the finite satisfiability problem for C^2 -formulas is in NEXPTIME,

*This is a simplified version of a paper originally posted on the Mathematics Preprint Server (<http://www.mathpreprints.com/math/Preprint/show/index.htm>). It was written in part during a sabbatical visit to the Division of Informatics, University of Edinburgh. The author acknowledges the support of the EPSRC (grant number GR/S22509). The author would further like to thank Prof. Andrei Voronkov for helpful conversations and Dr. Ulrike Sattler for her comments on the manuscript.

and furthermore yield a new and more perspicuous proof that the satisfiability problem for C^2 -formulas is in NEXPTIME.

The plan of the paper is as follows. The next section introduces the stellar fragment, and shows that both satisfiability and finite satisfiability for this fragment are NEXPTIME-hard. Section 3 shows that the finite satisfiability problem for the stellar fragment is in NEXPTIME. Section 4 shows that the satisfiability problem for the stellar fragment is in NEXPTIME. Section 5 explores the connection between the stellar fragment and C^2 .

In the sequel, we take $\mathbf{C}$ to be a set of constants, $\mathbf{F}$ a set of unary function-symbols, $\mathbf{P}$ a set of unary predicates and $\mathbf{Q}$ a set of binary predicates. We assume the signature $(\mathbf{C}, \mathbf{F}, \mathbf{P}, \mathbf{Q})$ as a default unless otherwise stated. A *literal* is an atom(ic formula) or the negation of an atom. A *clause* is a (finite) disjunction of literals. We allow the falsum $\perp$ as a clause (the ‘empty’ clause), and we occasionally write clauses as implications where this makes for clarity. Semantically, we treat clauses as shorthand for their universal closures: in particular, we say that a structure $\mathfrak{A}$ *satisfies* a clause γ , and write $\mathfrak{A} \models \gamma$, if the universal closure of γ is true in $\mathfrak{A}$. If Γ is a set of clauses, denote by $\|\Gamma\|$ the total number of symbols in Γ . If α is an atom, we write $\pm\alpha$ to stand ambiguously for α or $\neg\alpha$. (Disambiguation of $\pm$ is assumed to be uniform within the obvious contexts.) Throughout this section and the next, we shall assume *nontrivial domains*. That is, we take a set of clauses to be *satisfiable* if it is satisfied in a structure $\mathfrak{A}$ with $|A| > 1$. It is easy to see that this assumption does not affect the complexity results reported here.

2 The stellar fragment

We now turn to the clausal fragment which forms the main topic of this paper.

Definition 1. An atom is *stellar* if it is of one of the eleven forms

$$\begin{array}{cccccc} p(x) & p(fx) & q(x, x) & q(fx, fx) & q(x, fx) & q(fx, x) \\ x \approx c & fx \approx c & fx \approx x & fx \approx gx & fgx \approx x, & \end{array}$$

where x is a variable, p a unary predicate, q a binary predicate, c an individual constant and f, g function-symbols. A literal is *stellar* if the atom it features is stellar; a clause is *stellar* if all its literals are stellar and all have the same variable. The set of all stellar clauses will be called the *stellar fragment*.

The term “stellar” alludes to the system of sidereal metaphors adopted by Pacholski *et al* in their work on the fragment C^2 . Its appositeness should manifest itself in the course of the paper.

Example 1. The clause $p(x) \vee fgx \approx x$ is stellar. However, $p(x) \vee q(fgx) \vee fx \approx gx$ and $p(x) \vee fgx \approx c$ are not stellar, because the atoms $q(fgx)$ and $fgx \approx c$ are not among the eleven forms above; likewise, $p(x) \vee q(y)$ is not stellar, because it has two variables.

Although stellar clauses by definition feature only non-ground atoms, certain ground literals are logically equivalent to stellar clauses, thus:

$$\begin{aligned} \pm p(c) &\equiv (x \approx c \rightarrow \pm p(x)) & \pm p(fc) &\equiv (x \approx c \rightarrow \pm p(fx)) \\ \pm c \approx d &\equiv (x \approx c \rightarrow \pm x \approx d) & \pm fc \approx d &\equiv (x \approx c \rightarrow \pm fx \approx d). \end{aligned}$$

Accordingly, we shall occasionally treat such literals as stellar clauses by courtesy, understanding that they could be equivalently expressed as such.

The restrictiveness of Definition 1 is deceptive, for the stellar fragment is in fact highly expressive. In particular, it is easily seen that the stellar fragment lacks the finite model property. Thus, for example, the clauses

$$gfx \approx x \quad fx \not\approx c$$

state that f is injective but not surjective.

Indeed, rather more can be said on the subject of the sizes of satisfying structures. Let $X_1, \dots, X_n, X_1^*, \dots, X_{n+1}^*$ be unary predicates and f a unary function-symbol. Consider the clauses

$$\begin{aligned} X_1(z) \wedge \dots \wedge X_{i-1}(z) \wedge \neg X_i(z) &\rightarrow X_i^*(z) & (1 \leq i \leq n) \\ X_1(z) \wedge \dots \wedge X_n(z) &\rightarrow X_{n+1}^*(z) \\ X_i^*(z) &\rightarrow \neg X_j(fz) & (1 \leq j < i \leq n+1) \\ X_i^*(z) &\rightarrow X_i(fz) & (1 \leq i \leq n) \\ \pm X_j(z) \wedge X_i^*(z) &\rightarrow \pm X_j(fz) & (1 \leq i < j \leq n). \end{aligned}$$

This clause set, which we shall denote by $\Gamma_{n,f,\overline{X},\overline{X}^*}$, works as a kind of counting machine. If $\mathfrak{A} \models \Gamma_{n,f,\overline{X},\overline{X}^*}$ and $a \in A$, say that the *index* of a is the integer in the range $[0, 2^n - 1]$ whose n -digit binary representation $d_n \dots d_1$ is given by

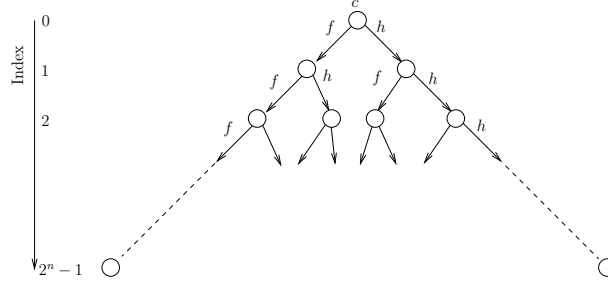
$$d_i = \begin{cases} 1 & \text{if } \mathfrak{A} \models X_i[a] \\ 0 & \text{otherwise.} \end{cases}$$

A little checking shows that, if $f^{\mathfrak{A}}(a) = a'$, the index of a' is one greater than the index of $a \pmod{2^n}$.

Using this technique, we can write clause sets which are finitely satisfiable, but only in very large structures. Let $\Gamma_{n,f,\overline{X},\overline{X}^*}$ be as above, and $\Gamma_{n,h,\overline{X},\overline{X}^*}$ the corresponding clauses with f replaced by another function-symbol h . Now let Δ be all these clauses together with

$$\begin{aligned} \neg X_{n+1}^*(z) &\rightarrow gfx \approx z & fz \not\approx hz \\ \neg X_{n+1}^*(z) &\rightarrow ghz \approx z & \neg X_i(c) \quad (1 \leq i \leq n) \end{aligned}$$

The clause set Δ is certainly finitely satisfiable, as depicted schematically in Fig. 1. (Unimportant functional links have been suppressed for clarity.) It is easy to see that any model of Δ incorporates a binary tree of depth 2^n , with f and g mapping each element with index less than $2^n - 1$ to an element

Figure 1: Schematic diagram of a model of Δ .

with index one greater; and this tree has $2^{2^n} - 1$ nodes. Thus, the size of the smallest satisfying model for clause sets in the stellar fragment may grow doubly exponentially as the size of the clause set itself. (It follows from Theorem 2 below that things never get essentially worse than that.)

The very same technique can be used to prove a lower complexity bound for the stellar fragment. Denote by $\mathbb{N}_m$ the natural numbers less than m . Recall that an *exponential tiling problem* is a triple (T, H, V) , where T is a finite set and H and V are binary relations over T . We call the elements of T *tiles*, and we call H and V the *horizontal constraints* and the *vertical constraints*, respectively. An *instance* of (T, H, V) is a list $t_0, \dots, t_{n-1}$ of elements of T (repetitions allowed). Such an instance is *positive* if there exists a function $\tau : \mathbb{N}_{2^n} \times \mathbb{N}_{2^n} \rightarrow T$ such that: (i) $\tau(i, 0) = t_i$ for all i ($0 \leq i \leq n-1$); (ii) $\langle \tau(i, j), \tau(i+1, j) \rangle \in H$ for all i, j ($0 \leq i < 2^n - 1, 0 \leq j \leq 2^n - 1$); and (iii) $\langle \tau(i, j), \tau(i, j+1) \rangle \in V$ for all i, j ($0 \leq i \leq 2^n - 1, 0 \leq j < 2^n - 1$). We refer to τ as a *tiling*. Intuitively, the elements of T represent colours of unit square tiles which must be arranged so as to fill a grid of $2^n \times 2^n$ squares. The constraints H (respectively, V) list which colours are allowed to go to the right of (respectively, above) which others. To show that a problem is NEXPTIME-hard, it suffices to encode it, in polynomial time, as an exponential tiling problem (T, H, V) .

Theorem 1. *The satisfiability and finite satisfiability problems for the stellar fragment are both NEXPTIME-hard.*

Proof. For $n > 0$, let $X_1, \dots, X_n, X_1^*, \dots, X_{n+1}^*, Y_1, \dots, Y_n, Y_1^*, \dots, Y_{n+1}^*$ be unary predicates, and let h, v be unary function-symbols. Now form the clause sets $\Gamma_{n,h,\bar{X},\bar{X}^*}, \Gamma_{n,v,\bar{Y},\bar{Y}^*}$ as explained above, and let Γ_{XY} be all these clauses together with

$$\begin{aligned} \pm X_i(z) &\rightarrow \pm X_i(vz) & (1 \leq i \leq n) \\ \pm Y_i(z) &\rightarrow \pm Y_i(hz) & (1 \leq i \leq n). \end{aligned}$$

If $\mathfrak{A} \models \Gamma_{XY}$ and $a \in A$, we can think of the satisfaction of the X_i by a as defining the x -coordinate of a in the range $[0, 2^n - 1]$, and the satisfaction of the Y_i by a as defining its y -coordinate in the same range. These coordinates assign

every element a location on an exponentially large grid. In terms of this grid, the function $h^{\mathfrak{A}}$ moves horizontally one step to the right, while the function $v^{\mathfrak{A}}$ moves vertically one step up (with wrap-around in both cases).

Let $c_0, \dots, c_{n-1}, c^*$ be individual constants and let Γ_0 be the collection of clauses

$$\begin{aligned} \neg X_i(c_0) & & (0 \leq i \leq n-1) \\ \neg Y_i(c_0) & & (0 \leq i \leq n-1) \\ \bigvee_{0 \leq i \leq n-1} \neg X_i(z) \vee \bigvee_{0 \leq i \leq n-1} \neg Y_i(z) \vee z \approx c^* \end{aligned}$$

stating that c_0 has grid-coordinates $(0, 0)$ and that c^* is the only element with grid-coordinates $(2^n - 1, 2^n - 1)$. Finally, let Γ^* be the pair of clauses

$$h^* h z \approx z \qquad v^* v z \approx z,$$

stating that the functions h and v have respective inverses h^* and v^* . Let $\Gamma = \Gamma_{XY} \cup \Gamma_0 \cup \Gamma^*$.

We adopt the following notation. For $l, m > 0$, let $\mathcal{T}[l, m]$ denote a term formed by applying l copies of the function-symbol h and m copies of the function-symbol v to the constant c_0 in some order or other.

Claim 1. *For any l, m ($0 \leq l, m \leq 2^n - 1$), and any pair of terms $\mathcal{T}_1[l, m]$, $\mathcal{T}_2[l, m]$, the clauses Γ entail the literal $\mathcal{T}_1[l, m] \approx \mathcal{T}_2[l, m]$.*

Write $2^n - 1 = N$. Evidently, $\Gamma \models X_i(\mathcal{T}[N, N])$ and $\Gamma \models Y_i(\mathcal{T}[N, N])$ for all i ($0 \leq i \leq n-1$), whence $\Gamma \models \mathcal{T}_1[N, N] \approx c^*$. Likewise, $\Gamma \models \mathcal{T}_2[N, N] \approx c^*$. This establishes the claim for $l = m = N$. Now consider any pair of terms $\mathcal{T}_j[N-1, N]$ ($j = 1, 2$). Since $h(\mathcal{T}_j[N-1, N])$ is of the form $\mathcal{T}'_j[N, N]$ for some $\mathcal{T}'_j$, we have already proved $\Gamma \models h(\mathcal{T}_1[N-1, N]) \approx h(\mathcal{T}_2[N-1, N])$. Using Γ^* , we have $\Gamma \models \mathcal{T}_1[N-1, N] \approx \mathcal{T}_2[N-1, N]$. Symmetrically, we have $\Gamma \models \mathcal{T}_1[N, N-1] \approx \mathcal{T}_2[N, N-1]$. By repeating this argument a sufficient number of times, the claim is proved.

Thus, any model of Γ must contain a collection of $(2^n)^2$ elements organized by the function-symbols h and v into a grid. Treating each $t \in T$ as a unary predicate, let Γ_I be the collection of unit clauses

$$\begin{aligned} h(c_i) \approx c_{i+1} & & (0 \leq i < n-1) \\ t_i(c_i) & & (0 \leq i \leq n-1) \end{aligned}$$

stating that the squares $(0, 0), \dots, (n-1, 0)$ are to be tiled with colours $t_0 \dots t_{n-1}$, and let Γ_T be the set of clauses

$$\begin{aligned} \neg t(z) \vee \neg t'(z) & \quad (\text{for } t, t' \in T \text{ with } t \neq t') \\ \bigvee_{t \in T} t(x), \end{aligned}$$

stating that each square on the grid must be covered by exactly one colour of tile. The horizontal constraints can then be encoded by the clauses Γ_H

$$X_n^*(z) \vee \neg t(z) \vee \neg t'(hz) \quad (t, t') \notin H$$

and the vertical constraints by a set of clauses Γ_V similarly. Consider the claims

1. $\Gamma \cup \Gamma_I \cup \Gamma_T \cup \Gamma_H \cup \Gamma_V$ has a model;
2. $t_0, \dots, t_{n-1}$ is a positive instance of the exponential tiling problem (T, H, V) ;
3. $\Gamma \cup \Gamma_I \cup \Gamma_T \cup \Gamma_H \cup \Gamma_V$ has a finite model.

The above construction shows that (1) implies (2) and that (2) implies (3). It is trivial that (3) implies (1). $\square$

3 Upper complexity bound: finite satisfiability

In spite of the large-model properties established in the previous section, both the satisfiability and the finite satisfiability problems for the stellar fragment are in NEXPTIME. The following terminology and notation will prove useful.

Definition 2. Let the structure $\mathfrak{A}$ be given. For $a' \neq a$, we write $a \xrightarrow{f} a'$ if $f^{\mathfrak{A}}(a) = a'$, and $a \rightarrow a'$ if $a \xrightarrow{f} a'$ for some function-symbol f . Mildly abusing notation, we speak of the *ray* $a \xrightarrow{f} a'$ or $a \rightarrow a'$ to denote a pair of (distinct) elements so related. If $a \rightarrow a'$ and $a' \rightarrow a$ we say that the rays in question are *invertible*, and write $a \leftrightarrow a'$.

Note that $a \rightarrow a$ is false by definition: functional links always feature distinct elements.

The following definitions recapitulate and extend some very familiar concepts from model theory. (Assume throughout the default signature $(\mathbf{C}, \mathbf{F}, \mathbf{P}, \mathbf{Q})$).

Definition 3. A *1-type* is a triple $\langle C, P, Q \rangle$ where $C \subseteq \mathbf{C}$ $P \subseteq \mathbf{P}$ $Q \subseteq \mathbf{Q}$.

Let $\mathfrak{A}$ be a structure, with $a \in A$. Then the triple

$$\pi = \langle \{c \in \mathbf{C} \mid a = c^{\mathfrak{A}}\}, \{p \in \mathbf{P} \mid a \in p^{\mathfrak{A}}\}, \{q \in \mathbf{Q} \mid \langle a, a \rangle \in q^{\mathfrak{A}}\} \rangle$$

is a 1-type, called the 1-type of a in $\mathfrak{A}$, and denoted $\text{tp}^{\mathfrak{A}}[a]$, or simply $\text{tp}[a]$ if $\mathfrak{A}$ is clear from context. It is obvious that $\pi = \text{tp}^{\mathfrak{A}}[a]$ provides a complete *individual* description of a in $\mathfrak{A}$ in the following sense. Suppose γ is a function-free stellar clause and $\mathfrak{A} \models \gamma[a]$; then for any structure $\mathfrak{B}$ with $b \in B$ such that $\text{tp}^{\mathfrak{B}}[b] = \pi$, we have $\mathfrak{B} \models \gamma[b]$. In that case, we may write $\pi \models \gamma$, since the $\mathfrak{A}$ and a are immaterial. Furthermore, given π and γ encoded as strings of symbols in the usual way, a trivial (polynomial time) syntactic check determines whether $\pi \models \gamma$.

Definition 4. A *ray-type* is a sextuple $\rho = \langle \pi^*, F, G, R, S, \pi \rangle$, where π^*, π are 1-types, $\emptyset \neq F \subseteq \mathbf{F}$, $G \subseteq \mathbf{F}$, $R \subseteq \mathbf{Q}$ and $S \subseteq \mathbf{Q}$. We denote the respective components of ρ by $\pi_\rho^*, F_\rho, G_\rho, R_\rho, S_\rho$ and π_ρ . If ρ is a ray-type with $G_\rho \neq \emptyset$, then ρ is said to be *invertible*, with *inverse* $\rho^{-1} = \langle \pi_\rho, G_\rho, F_\rho, S_\rho, R_\rho, \pi_\rho^* \rangle$.

The intuition behind ray-types is as follows (see Fig. 2a). Let $\mathfrak{A}$ be a structure, and a, a' distinct elements of A with $a \rightarrow a'$. Setting

$$\begin{aligned} \pi^* &= \text{tp}^{\mathfrak{A}}[a] & \pi &= \text{tp}^{\mathfrak{A}}[a'] \\ F &= \{f \in \mathbf{F} \mid f^{\mathfrak{A}}(a) = a'\} & G &= \{g \in \mathbf{F} \mid g^{\mathfrak{A}}(a') = a\} \\ R &= \{q \in \mathbf{Q} \mid \mathfrak{A} \models q[a, a']\} & S &= \{q \in \mathbf{Q} \mid \mathfrak{A} \models q[a', a]\}, \end{aligned}$$

the ray-type $\rho = \langle \pi^*, F, G, R, S, \pi \rangle$ describes the relationship between a and a' in $\mathfrak{A}$. When $\mathfrak{A}$ is clear from context, we write $a \xrightarrow{\rho} a'$ in this case, and say that a' *absorbs* a ray of type ρ emitted by a .

Definition 5. A *star* σ is a set of ray-types such that:

1. the (nonempty) sets F_ρ ($\rho \in \sigma$) are pairwise disjoint; and
2. the 1-types π_ρ^* ($\rho \in \sigma$) are all identical.

The common 1-type π_ρ^* for $\rho \in \sigma$ is denoted by $\text{tp}(\sigma)$; in addition, we write:

$$H_\sigma = \mathbf{F} \setminus \bigcup_{\rho \in \sigma} F_\rho.$$

Note that, if $H_\sigma = \emptyset$, then $\{F_\rho \mid \rho \in \sigma\}$ is a partition of $\mathbf{F}$; otherwise, $\{H\} \cup \{F_\rho \mid \rho \in \sigma\}$ is a partition of $\mathbf{F}$.

The intuition behind stars is as follows (see Fig. 2b). Let $\mathfrak{A}$ be a structure, and $a \in A$. Set

$$\sigma = \{\rho \mid \text{for some } a' \in A, a \xrightarrow{\rho} a'\}.$$

Then σ is a star, called the *star of a* , and denoted $\text{st}^{\mathfrak{A}}[a]$, or simply $\text{st}[a]$ if $\mathfrak{A}$ is clear from context. Thus, for each $a \in A$, $\text{tp}(\text{st}[a]) = \text{tp}[a]$, and for each $\rho \in \text{st}[a]$ and each $f \in F_\rho$, $\pi_\rho = \text{tp}[f^{\mathfrak{A}}(a)]$. Finally, we note that, if σ is $\text{st}^{\mathfrak{A}}[a]$, then H_σ is the set of function-symbols h such that $h^{\mathfrak{A}}(a) = a$. It is obvious that $\sigma = \text{st}^{\mathfrak{A}}[a]$ provides a complete *local* description of a in $\mathfrak{A}$ in the following sense. Suppose γ is a stellar clause, and $\mathfrak{A} \models \gamma[a]$. Then for any structure $\mathfrak{B}$ with $b \in B$ such that $\text{st}^{\mathfrak{B}}[b] = \sigma$, we have $\mathfrak{B} \models \gamma[b]$. In that case, we may write $\sigma \models \gamma$, since the $\mathfrak{A}$ and a are immaterial. Again, given σ and γ encoded as strings of symbols in the usual way, a simple (polynomial time) syntactic check determines whether $\sigma \models \gamma$.

The following general observations on structures simplify the ensuing analysis.

Definition 6. Let $\mathfrak{A}$ be a structure. We say that $\mathfrak{A}$ is *chromatic* if

1. For all (distinct) $a, a' \in A$, if $a \leftrightarrow a'$, then $\text{tp}[a] \neq \text{tp}[a']$.

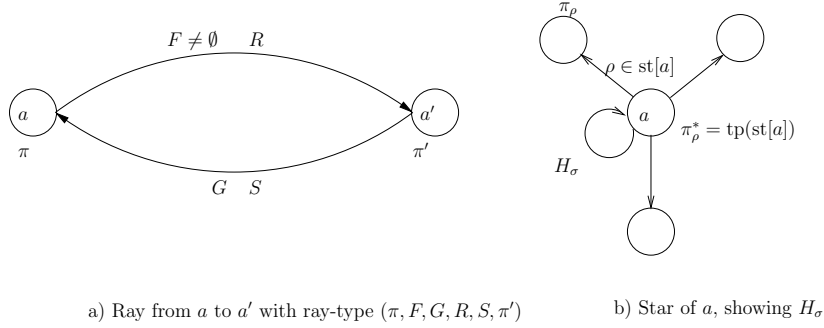


Figure 2: Schematic diagram of a ray and a star.

2. For all pairwise distinct $a, a', a'' \in A$, if $a \leftrightarrow a'$ and $a' \leftrightarrow a''$ then $\text{tp}[a] \neq \text{tp}[a'']$.

In addition, let m be an integer. We say that $\mathfrak{A}$ is *m-differentiated* if, for every 1-type π , the number u of elements in A realizing π satisfies either $u \leq 1$ or $u \geq m$.

Lemma 1. *Let $m > 1$ and $\mathfrak{A}$ be a structure interpreting at most n function-symbols. Then $\mathfrak{A}$ can be expanded to a chromatic, m -differentiated structure by interpreting at most $\log(n^2 + n + 1) + \log(m - 1) + 1$ (rounded up) new unary predicates.*

Proof. Consider the (undirected) graph G on A whose edges E are the pairs of distinct elements connected by a chain of 1 or 2 invertible rays, that is:

$$E = \{(a, a') \mid a \leftrightarrow a'\} \cup \{(a, a'') \mid a \leftrightarrow a' \text{ and } a' \leftrightarrow a'' \text{ for some } a'\}.$$

The degree of this graph is bounded by $n^2 + n$. Now use a greedy algorithm to colour G with $n^2 + n + 1$ colours. By interpreting $\log(n^2 + n + 1)$ (rounded up) new unary predicates to encode these colours, we thus ensure that the structure is chromatic. Finally, for each 1-type π (in the expanded signature) realized more than once but fewer than m times, colour the elements realizing π using $m - 1$ different colours. By interpreting $\log(m - 1)$ (rounded up) new unary predicates to encode these colours, we ensure that the structure is m -differentiated. $\square$

We are now in a position to establish our first main result on the stellar fragment.

Theorem 2. *Let Γ be a set of simple stellar clauses over the signature $(\mathbf{C}, \mathbf{F}, \mathbf{P}, \mathbf{Q})$, and let n be such that $|\mathbf{F}| \leq n$, $|\mathbf{P}| \leq n$, $|\mathbf{Q}| \leq n$ and $|\mathbf{C}| \leq 2^n$. If Γ is finitely satisfiable, then Γ is satisfied in a structure whose size is bounded by a doubly exponential function of n .*

Proof. Let Δ be some maximal consistent set of clauses of the forms $\pm c \approx c'$, where $c, c' \in \mathbf{C}$. (Remember: we write these clauses, in stellar form, as $x \approx c \rightarrow \pm x \approx c'$.) Such a clause set partitions the set $\mathbf{C}$ into at most $|\mathbf{C}| \leq 2^n$ cells in the obvious way. We address the problem of determining whether $\Gamma_\Delta = \Gamma \cup \Delta$ is satisfiable.

Let $n' = \log(n^2 + n + 1) + \log(3n - 1) + 1$ (rounded up), and add n' new unary predicates to $\mathbf{P}$ to obtain the set $\mathbf{P}'$. If Γ_Δ is finitely satisfiable, by Lemma 1, it has a finite, chromatic, $3n$ -differentiated model over the signature $(\mathbf{C}, \mathbf{F}, \mathbf{P}', \mathbf{Q})$; and if σ is the star of any element in such a model, then σ obeys the following conditions. (The final two both follow from the fact that the model is chromatic.)

(A1) $\sigma \models \Gamma$.

(A2) $\sigma \models \Delta$.

(A3) σ contains no invertible ray-type ρ such that $\pi_\rho = \text{tp}(\sigma)$.

(A4) For each 1-type π , σ contains at most one invertible ray-type ρ such that $\pi_\rho = \pi$.

Enumerate the set of all stars satisfying (A1)–(A4) as $\sigma_1 \dots \sigma_N$. Enumerate the set of ray-types $\bigcup \{\sigma_k \mid 1 \leq k \leq N\}$ as $\rho_1 \dots \rho_M$. Enumerate the set of 1-types $\{\text{tp}(\sigma) \mid 1 \leq k \leq N\}$ as $\pi_1 \dots \pi_L$. For the purpose of determining the (finite) satisfiability of Γ_Δ , these are the only stars, ray-types and 1-types which interest us. For any σ satisfying (A2), the set of $c \in \mathbf{C}$ such that $\sigma \models x \approx c$ is either empty or a cell of the partition of $\mathbf{C}$ induced by Δ . Since there are at most 2^n such cells and at most $n + n'$ unary predicates, we have $L \leq (2^n + 1) \cdot 2^{n+n'}$. Moreover, $M \leq L^2 \cdot 2^{4n}$, and $N \leq M^n$. Thus, all these numbers are singly exponential in n .

Now we define arrays H , I , $\mathcal{R}$ and $\mathcal{S}$ as follows. Let H be the $N \times L$ integer-valued array and I the $N \times M$ integer-valued array, with entries given by

$$H_{ki} = \begin{cases} 1 & \text{if } \text{tp}(\sigma_k) = \pi_i \\ 0 & \text{otherwise,} \end{cases} \quad I_{kj} = \begin{cases} 1 & \text{if } \rho_j \in \sigma_k \\ 0 & \text{otherwise.} \end{cases}$$

These arrays are self-explanatory: $H_{ki} = 1$ indicates that any element with star σ_k has 1-type π_i ; $I_{kj} = 1$ indicates that any element with star σ_k emits a ray of type ρ_j .

Now let $\mathcal{R}$ be the $L \times N$ set-valued array with entries given by

$$\mathcal{R}_{ik} = \{j \mid 1 \leq j \leq M, \rho_j \text{ is non-invertible, } \rho_j \in \sigma_k \text{ and } \pi_{\rho_j} = \pi_i\},$$

and let $\mathcal{S}$ be the $L \times L$ set-valued array with entries given by

$$\mathcal{S}_{i'i} = \{k \mid 1 \leq k \leq N, \text{tp}(\sigma_k) = \pi_{i'} \text{ and for all } \rho \in \sigma_k, \pi_\rho \neq \pi_i\}.$$

These two arrays are admittedly somewhat perplexing. The entry $\mathcal{R}_{ik}$ is the set of (indices of) *non-invertible* ray-types in σ_k which require absorption by an element whose 1-type is π_i . And the entry $\mathcal{S}_{i'i}$ is the set of (indices of) stars whose 1-type is $\pi_{i'}$ and which contain no ray-type that requires to be absorbed by an element whose 1-type is π_i . Finally, if $c \in \mathbf{C}$, let

$$\mathcal{W}_c = \{k \mid 1 \leq k \leq N \text{ and } \sigma_k \models x \approx c\}.$$

Using this machinery, we construct a quantifier-free formula of Presburger arithmetic ϕ_{Γ_Δ} such that ϕ_{Γ_Δ} is satisfiable in Presburger arithmetic if and only if Γ_Δ is finitely satisfiable. To motivate some of the technical details, we first *assume* that Γ_Δ is finitely satisfiable, and then show, as we construct ϕ_{Γ_Δ} , that it is satisfiable in Presburger arithmetic. We then establish the converse result: that if ϕ_{Γ_Δ} is satisfiable in Presburger arithmetic, then Γ_Δ is finitely satisfiable.

Suppose then that Γ_Δ is finitely satisfiable. Choose a finite, chromatic, $3n$ -differentiated model $\mathfrak{A} \models \Gamma_\Delta$ over the signature $(\mathbf{C}, \mathbf{F}, \mathbf{P}', \mathbf{Q})$. For all i, i' ($1 \leq i, i' \leq L$), all j ($1 \leq j \leq M$) and all k ($1 \leq k \leq N$), define:

$$\begin{aligned} u_i &= |\{a \in A : \text{tp}[a] = \pi_i\}| \\ v_j &= |\{(a, a') \in A^2 : a \xrightarrow{\rho_j} a'\}| \\ w_k &= |\{a \in A : \text{st}[a] = \sigma_k\}| \\ t_{i'i} &= |\{a \in A : \text{tp}[a] = \pi_{i'} \text{ and for all } a' \in A, \text{ if } a \rightarrow a', \text{ then } \text{tp}[a'] \neq \pi_i\}|. \end{aligned}$$

Thus, for i, i', j, k in the indicated ranges, u_i is the number of elements having 1-type π_i ; v_j is the number of rays of type ρ_j ; and w_k is the number of elements having star σ_k . The quantities $t_{i'i}$ are admittedly somewhat perplexing. For all i, i' ($1 \leq i, i' \leq L$), $t_{i'i}$ is the number of elements whose 1-type is $\pi_{i'}$ but which do not emit any ray that is absorbed by an element whose 1-type is π_i . We write $\vec{u}$ for the vector $(u_1, \dots, u_N)$, and similarly with $\vec{v}$, $\vec{w}$. We write $\vec{t}$ to denote the $t_{i'i}$ in some arbitrary order.

It is immediate from the foregoing definitions that the following constraints are satisfied:

- (B1) $\vec{u} = \vec{w}H$
- (B2) $\vec{v} = \vec{w}I$
- (B3) $t_{i'i} = \sum_{k \in \mathcal{S}_{i'i}} w_k$ for all i, i' ($1 \leq i, i' \leq L$).
- (B4) $\sum_{k \in \mathcal{W}_c} w_k = 1$ for all $c \in \mathbf{C}$.

Finally, the following constraints are also satisfied for all i, i', j, j', k (in the appropriate ranges) subject to the indicated conditions:

- (C1) If ρ_j is invertible and $\rho_j^{-1} = \rho_{j'}$, then $v_j = v_{j'}$.
- (C2) If σ_k contains some ρ such that $\pi_\rho = \pi_i$, then $u_i > 0$ or $w_k = 0$.

- (C3) If σ_k contains distinct ray-types ρ, ρ' such that $\pi_\rho = \pi_{\rho'} = \pi_i$, then $u_i > 1$ or $w_k = 0$.
- (C4) Either $u_i \leq 1$ or $u_i \geq 3n$
- (C5) If $\text{tp}(\sigma_k) = \pi_i$, then either $u_i > 1$ or $w_k = 0$ or $t_{i'i} \geq |\mathcal{R}_{i'k}|$.

Constraint (C1) follows from the fact that, if the ray-type $\rho_{j'}$ is the inverse of the ray-type ρ_j , then every invertible ray contributing to v_j will have a distinct inverse ray contributing to $v_{j'}$. To see that (C2) is true, suppose σ_k contains a ray-type ρ with $\pi_\rho = \pi_i$. If $w_k > 0$, then at least one ray of type ρ is emitted, and this ray must be absorbed by an element a' of 1-type π_i , whence $u_i > 0$. Constraint (C3) follows by a similar argument. Constraint (C4) follows from the fact that $\mathfrak{A}$ is $3n$ -differentiated. To see that (C5) is true, suppose $\text{tp}(\sigma_k) = \pi_i$, with $u_i \leq 1$ and $w_k > 0$. Then $\mathfrak{A}$ contains exactly one element a whose 1-type in $\mathfrak{A}$ is π_i ; moreover, $\text{st}[a] = \sigma_k$. It follows that $t_{i'i} = |T|$, where $T = \{a' \in C : \text{tp}[a'] = \pi_{i'} \text{ and } a' \not\approx a\}$; moreover, $|\mathcal{R}_{i'k}|$ is the number of *non-invertible* rays emitted by a and absorbed by elements with 1-type $\pi_{i'}$. Each of these rays must be absorbed by some element of T (otherwise, they would not be non-invertible); and distinct rays emitted by a are by definition absorbed by distinct elements. It follows that $t_{i'i} \geq |\mathcal{R}_{i'k}|$.

Regarding the u_i, v_j, w_k and $t_{i'i}$ as variables, the constraints (B1)–(B4) form a set of equations involving these variables. In fact, each of these equations is a quantifier-free formula of Presburger arithmetic. Moreover, each of the conditional constraints (C1)–(C5) forms a Boolean combination of equations and inequations, obtained by substituting the various values of i, i', j, j', k which make the relevant conditions true. Again, each of these Boolean combinations is a quantifier-free formula of Presburger arithmetic. Let $\phi_{\Gamma\Delta}$ be the conjunction of all these Presburger formulas taken together. Since $\mathfrak{A}$ satisfies (B1)–(B4) and (C1)–(C5), it follows that $\phi_{\Gamma\Delta}$ is satisfiable in Presburger arithmetic. Of course, the construction of $\phi_{\Gamma\Delta}$ made reference only to Γ and Δ , not to $\mathfrak{A}$!

We now argue the converse: that if $\phi_{\Gamma\Delta}$ is satisfiable in Presburger arithmetic, then Γ_Δ is finitely satisfiable. To reduce notational clutter, take the free variables in $\phi_{\Gamma\Delta}$ to stand for the integer values which form a satisfying assignment; we proceed to construct a model $\mathfrak{A} \models \Gamma_\Delta$ based on these values.

The first step is to define the domain. For every k ($1 \leq k \leq N$), let A_k be a set of cardinality w_k , and let A be the disjoint union of the A_k . We now convert A into a structure $\mathfrak{A}$.

Interpreting the constants and functional loops of order 1: Given $a \in A$, let k be such that $a \in A_k$. If c is an individual constant, set $c^{\mathfrak{A}} = a$ if and only if $\sigma_k \models x \approx c$. By (B4), there is exactly one value of k' ($1 \leq k' \leq N$) for which $w_{k'} > 0$ and $\sigma_{k'} \models x \approx c$; moreover, in that case, $w_{k'} = 1$. Hence, these assignments interpret every individual constant uniquely. In addition, we decree that, for every function-symbol $h \in H_{\sigma_k}$, $h^{\mathfrak{A}}(a) = a$.

Interpreting the unary predicates and diagonals of binary predicates: If p is a unary predicate, set $\mathfrak{A} \models p[a]$ if and only if $\sigma_k \models p(x)$. Finally, if q is a binary predicate, set $\mathfrak{A} \models q[a, a]$ if and only if $\sigma_k \models q(x, x)$.

Interpreting the function-symbols and off-diagonals of binary predicates: We deal with the invertible rays first. For each j ($1 \leq j \leq M$), set D_j to be the union of those A_k such that $\rho_j \in \sigma_k$. It follows from **(B2)** that $|D_j| = v_j$. For any j ($1 \leq j \leq M$) such that ρ_j is invertible, let j' be such that $\rho_{j'} = \rho_j^{-1}$. By **(C1)**, D_j and $D_{j'}$ are equinumerous. (By **(A3)**, they are also disjoint.) Pick any 1–1 correspondence between them, and for $a \in D_j$, let $a' \in D_{j'}$ be the corresponding element. Now set

$$\begin{aligned} f^{\mathfrak{A}}(a) &= a' \text{ for all } f \in F_\rho & \mathfrak{A} \models q[a, a'] &\text{ iff } q \in R_\rho \\ g^{\mathfrak{A}}(a') &= a \text{ for all } g \in G_\rho & \mathfrak{A} \models q[a', a] &\text{ iff } q \in S_\rho. \end{aligned} \quad (1)$$

In this way, all invertible rays will be properly absorbed. We need to check that the assignments in (1) can never conflict. But **(A3)** ensures that if $\rho \in \sigma_k$ is invertible, then $\pi_\rho \neq \pi_{\sigma_k}$; and **(A4)** ensures that, if we have distinct invertible ray-types $\rho', \rho'' \in \sigma_k$, then $\pi_{\rho'} \neq \pi_{\rho''}$, so that, in particular, ρ'^{-1}, ρ''^{-1} cannot belong to a common star $\sigma_{k'}$. It follows that no loops of order 1 or ‘parallel edges’ can arise in the above assignments.

Having dealt with the invertible rays, we move on to the non-invertible ones. First, some preliminary definitions. For all i ($1 \leq i \leq L$), let B_i be the union of those A_k such that $\text{tp}(\sigma_k) = \pi_i$. It follows from **(B1)** that $|B_i| = u_i$, and thence from **(C4)** that if $|B_i| > 1$, then $|B_i| \geq 3n$: if so, let B_i be decomposed into disjoint sets B_{i0} , B_{i1} and B_{i2} , each containing at least n elements. Finally, for all i', i ($1 \leq i', i \leq L$), let $C_{i'i} = \bigcup_{k \in \mathcal{S}_{i'i}} A_k$. (Think of $C_{i'i}$ as the set of elements which have been assigned 1-type $\pi_{i'}$ and do not ‘want’ to emit any ray ρ with $\pi_\rho = \pi_i$.) Note that $C_{i'i} \subseteq B_{i'}$. It follows from **(B3)** that $|C_{i'i}| = t_{i'i}$. Now we are ready to deal with the non-invertible rays. We proceed according to the 1-types they feature. Fix i and i' ($1 \leq i, i' \leq L$), and consider any non-invertible ray-type ρ such that $\pi_\rho^* = \pi_i$ and $\pi_\rho = \pi_{i'}$. We have three cases to consider. (See Fig. 3.)

Case 1: $|B_i| \geq 3n$ and $|B_{i'}| \geq 3n$. Suppose $a \in B_i$. Then for some h ($0 \leq h < 3$), $a \in B_{ih}$. Let k be such that $a \in A_k$. If σ_k contains the ray-type ρ , let $h' = h + 1 \pmod{3}$, choose $a' \in B_{i'h'}$, and set

$$\begin{aligned} f^{\mathfrak{A}}(a) &= a' \text{ for all } f \in F_\rho & \mathfrak{A} \models q[a, a'] &\text{ iff } q \in R_\rho \\ & & \mathfrak{A} \models q[a', a] &\text{ iff } q \in S_\rho. \end{aligned} \quad (2)$$

We must check that the assignments (2) do not lead to conflicts created by parallel edges or loops of order 1 or 2. Since $|B_{i'h'}| \geq n$, there are always enough choices for a' avoiding parallel edges; and the fact that $h' = h + 1 \pmod{3}$ guarantees that no loops of order 1 or 2 arise (even if $i = i'$).

Case 2: $|B_i| \leq 1$. Suppose $a \in B_i$. Let k be such that $a \in A_k$, so that $w_k > 0$, whence, by **(C5)**, $C_{i'i}$ is at least as numerous as $\mathcal{R}_{i'k}$. If, then, σ_k

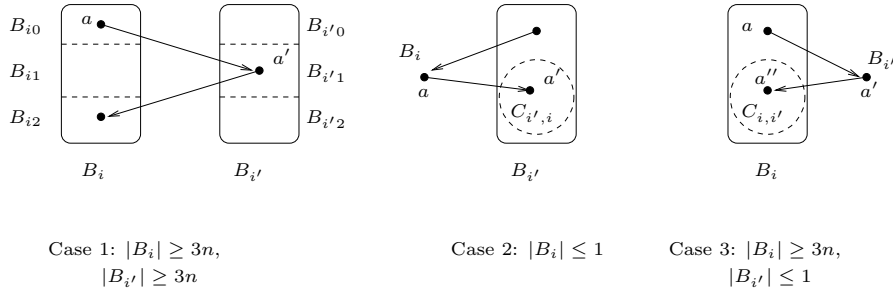


Figure 3: Dealing with non-invertible rays.

contains the ray-type ρ , select $a' \in C_{i'i}$, and carry out the assignments (2). Again, we must check that these assignments do not lead to conflicts. But the fact that $|C_{i'i}| \geq |\mathcal{R}_{i'i}|$ guarantees that no parallel edges need be created; and the definition of $C_{i',i}$ guarantees that no loops of order 1 or 2 can be created. Notice that all the absorption sites chosen under this case are in $C_{i',i}$.

Case 3: $|B_i| \geq 3n$ and $|B_{i'}| \leq 1$. Suppose $a \in B_i$. Let k be such that $a \in A_k$, so that $w_k > 0$. If $\rho \in \sigma_k$, then, by **(C2)**, $|B_{i'}| = 1$, and by **(C3)**, ρ is the only ray-type $\rho' \in \sigma_k$ such that $\pi_{\rho'} = \pi_{i'}$. Let a' be the unique element of $B_{i'}$ and carry out the assignments (2). Again, we must check that these assignments do not lead to conflicts. The fact that ρ is the only ray-type ρ' in σ_k with $\pi_{\rho'} = \pi_{i'}$ guarantees that no parallel edges can be created. To see that no loops of order 1 or 2 can be created, let k' be such that $a' \in A_{k'}$. The only danger is from a non-invertible ray-type $\rho' \in \sigma_{k'}$ with $\pi_{\rho'} = \pi_i$. Let a'' be an element absorbing any ray of this type emitted by a' . Clearly, a'' is assigned under Case 2, guaranteeing that $a'' \in C_{i,i'}$. But the definition of $C_{i,i'}$ ensures that $a \notin C_{i,i'}$, so $a'' \neq a$ as required.

Finally, complete the extensions of the binary predicates in some arbitrary way.

From the above construction, it is evident that the following holds for all $a \in A$. Let k be such that $a \in A_k$; then $\text{st}^{\mathfrak{A}}[a] = \sigma_k$. It is then immediate from **(A1)** and **(A2)** that $\mathfrak{A} \models \Gamma_{\Delta}$. We have thus shown that Γ_{Δ} has a finite model if and only if $\phi_{\Gamma_{\Delta}}$ is satisfiable in Presburger arithmetic. Given the singly exponential bounds on L , M and N , the size of $\phi_{\Gamma_{\Delta}}$ is singly exponential in n .

To complete the proof, suppose Γ is finitely satisfiable. Let $\mathfrak{A}'$ be a finite model of Γ , and let Δ consist of the clauses $\pm c \approx c'$ true in $\mathfrak{A}'$. Construct $\phi_{\Gamma_{\Delta}}$. Since $\mathfrak{A}' \models \Gamma_{\Delta}$, $\phi_{\Gamma_{\Delta}}$ is satisfiable in Presburger arithmetic. But it is well-known that any integer programming problem which has a solution at all has a solution which is bounded by a singly exponential function of the size of the problem (see, e.g. Papadimitriou [5]), a result which obviously carries over to quantifier-free Presburger formulas. So choose a solution $\vec{u}, \vec{v}, \vec{w}, \vec{t}$ of $\phi_{\Gamma_{\Delta}}$ which is so bounded, and then build the model $\mathfrak{A}$ of Γ_{Δ} as outlined above. Since $|A| = w_1 + \dots + w_N$,

$\mathfrak{A}$ satisfies the size bound of the theorem, and we are done. $\square$

The full strength of the following (slightly fussy) complexity result is needed for Section 5.

Theorem 3. *There exists a non-deterministic algorithm $\mathcal{A}$ and a singly exponential function $b : \mathbb{N} \rightarrow \mathbb{N}$ with the following property. Let Γ be a set of simple stellar clauses over the signature $(\mathbf{C}, \mathbf{F}, \mathbf{P}, \mathbf{Q})$, where $|\mathbf{F}| \leq n$, $|\mathbf{P}| \leq n$, $|\mathbf{Q}| \leq n$, $|\mathbf{C}| \leq 2^n$ and $\|\Gamma\| \leq 2^n$. Then $\mathcal{A}(\Gamma)$ has a successfully terminating run in time bounded by $b(n)$ if and only if Γ is finitely satisfiable.*

Proof. Guess the set of equalities and inequalities Δ . Given the size bounds of the theorem, any solution of $\phi_{\Gamma\Delta}$ of size $O(2^{2^n})$ can be checked in time $O(2^n)$. $\square$

That is to say: finite satisfiability for sets of stellar clauses is in NEXPTIME.

4 Upper complexity bound: satisfiability

Having established the complexity of determining finite satisfiability in the stellar fragment, we now turn to the corresponding (general) satisfiability problem. In fact, there is almost no further work to do.

Let $\mathbb{N}^+$ be $\mathbb{N} \cup \{\aleph_0\}$. We extend the usual ordering $>$ on $\mathbb{N}$ to $\mathbb{N}^+$ by setting

$$\aleph_0 > n \quad \text{for all } n \in \mathbb{N}.$$

(Of course $\aleph_0 \not> \aleph_0$.) We extend the operations of addition and multiplication on $\mathbb{N}$ to $\mathbb{N}^+$ by setting

$$\begin{aligned} n + \aleph_0 &= \aleph_0 + n = \aleph_0 \\ 0 \cdot \aleph_0 &= \aleph_0 \cdot 0 = 0 \\ n \cdot \aleph_0 &= \aleph_0 \cdot n = \aleph_0 \quad \text{if } n > 0. \end{aligned}$$

A system of linear equalities and inequalities defining an integer programming problem can of course be re-interpreted so that solutions are sought not over $\mathbb{N}$ but over $\mathbb{N}^+$. (We always assume that the coefficients occurring in such problems are in $\mathbb{N}$.) As an example, the single inequality $x_1 \geq x_1 + 1$ has no solutions over $\mathbb{N}$, but it does have a solution over $\mathbb{N}^+$, namely, $x_1 = \aleph_0$.

We have the following result.

Lemma 2. *Let Φ be a finite set of linear equalities and inequalities*

$$a_0 + a_1x_1 + \cdots + a_nx_n \leq b_0 + b_1x_1 + \cdots + b_nx_n$$

in variables $x_1, \dots, x_n$. Here, all coefficients are assumed to be in $\mathbb{N}$. We take the size of Φ , denoted $\|\Phi\|$, to be measured in the usual way (with binary encoding if you want). If Φ has a solution over $\mathbb{N}^+$, then Φ has a solution over $\mathbb{N}^+$ such that all finite values are bounded by some (fixed) singly exponential function of $\|\Phi\|$.

Proof. Re-order the variables if necessary so that $\bar{a}\bar{\aleph}_0$ is a solution, with $\bar{a} = a_1, \dots, a_k \in \mathbb{N}^k$ for some k ($0 \leq k \leq n$) and $\bar{\aleph}_0$ a $(n - k)$ -tuple of $\aleph_0$ s. Say that an element of Φ *does not involve* the variable x_i if the corresponding coefficients a_i and b_i are zero. Let Ψ be the set of elements of Φ which do not involve any of the $x_{k+1}, \dots, x_n$. Thus, Ψ has a solution over $\mathbb{N}$, whence it has a solution $\bar{a}'$ bounded by some singly exponential function of $\|\Psi\|$ (and hence of $\|\Phi\|$). But then it is easy to see that $\bar{a}'\bar{\aleph}_0$ is a solution of Φ . $\square$

Theorem 4. *Let Γ be a set of simple stellar clauses over the signature $(\mathbf{C}, \mathbf{F}, \mathbf{P}, \mathbf{Q})$, and let n be such that $|\mathbf{F}| \leq n$, $|\mathbf{P}| \leq n$, $|\mathbf{Q}| \leq n$ and $|\mathbf{C}| \leq 2^n$. If Γ is satisfiable, then Γ is satisfied in a structure such that, for every star σ , the number of elements whose star is σ is either $\aleph_0$ or else is bounded by a some (fixed) doubly exponential function of n .*

Proof. Given Lemma 2, the argument of Theorem 2 applies unchanged if we allow the variables $\vec{u}, \vec{v}, \vec{w}, \vec{t}$ to take the value $\aleph_0$. $\square$

The proof that satisfiability for stellar clauses can be decided in nondeterministic exponential time then follows as for finite satisfiability. Again, for the purposes of the next section, we state this fact in a rather elaborate way.

Theorem 5. *There exists a non-deterministic algorithm $\mathcal{A}$ and a singly exponential function $b : \mathbb{N} \rightarrow \mathbb{N}$ with the following property. Let Γ be a set of stellar clauses over the signature $(\mathbf{C}, \mathbf{F}, \mathbf{P}, \mathbf{Q})$, where $|\mathbf{F}| \leq n$, $|\mathbf{P}| \leq n$, $|\mathbf{Q}| \leq n$, $|\mathbf{C}| \leq 2^n$ and $\|\Gamma\| \leq 2^n$. Then $\mathcal{A}(\Gamma)$ has a successfully terminating run in time bounded by $b(n)$ if and only if Γ is satisfiable.*

5 Counting quantifiers

Recall that C^2 denotes the two-variable fragment of first-order logic with counting quantifiers. In this section, we reduce the problem of determining the finite satisfiability (satisfiability) of a C^2 -formula to that of determining the finite satisfiability (respectively, satisfiability) of sets of clauses in the stellar fragment. Theorems 3 and 5 then establish that the finite satisfiability and satisfiability problems for C^2 are in NEXPTIME (assuming that the counting quantifiers are encoded in unary). The second of these results was previously proved, using a difficult argument, by Pacholski, Szwoast and Tendera [4]. As before, we restrict attention to domains containing more than one element, since satisfiability of first-order formulas over one-element domains reduces to propositional logic.

Let ϕ be a formula in C^2 . Without loss of generality, we may assume that ϕ contains no constants, since these may be eliminated in the standard way. Thus, we take the relevant signature to be $(\mathbf{P}, \mathbf{Q})$, where $\mathbf{P}$ is a set of unary predicates and $\mathbf{Q}$ a set of binary predicates. As a preliminary, we normalize ϕ so that resolution can be applied. It is routine to show that ϕ can be transformed into an equisatisfiable formula of the form

$$\phi_0 = \forall x \forall y \alpha'(x, y) \wedge \bigwedge_{1 \leq k \leq n} \forall x \exists! y \beta'_k(x, y),$$

where α' and the β'_k are quantifier- and function-free formulas in conjunctive normal form. This transformation is, in essence, the one exploited by Scott [6] for the purpose of reducing the two-variable fragment (without equality) to the Gödel fragment. In fact, under the restriction that domains contain at least two elements, ϕ_0 is easily shown to be logically equivalent to a formula of the form

$$\phi_1 = \forall x \forall y (\alpha(x, y) \vee x \approx y) \wedge \bigwedge_{1 \leq k \leq n} \forall x \exists! y (\beta_k(x, y) \wedge x \not\approx y),$$

where the $\alpha(x, y)$ and $\beta_k(x, y)$ are quantifier, equality- and function-free formulas in conjunctive normal form. (For details, see, e.g. Börger, Grädel and Gurevich [1], pp. 377–378.) The transformation runs in polynomial time if unary encoding is used to represent the numbers in the quantifiers $\exists_{\geq n}$, and exponential time (with exponentially large result) if binary encoding is used.

Our strategy will be to transform ϕ_1 into a collection $\mathcal{G}_\phi$ of sets of stellar clauses such that, ϕ is satisfiable over some domain A if and only some $\Gamma \in \mathcal{G}_\phi$ is satisfiable over A . The most obvious problem is that ϕ_1 contains two variables. We proceed therefore to re-write this formula so as to eliminate all occurrences of the variable y . We proceed in three steps.

Step1: (Skolemization). For each k , ($1 \leq k \leq n$), let f_k be a new function-symbol. Defining the formula $\pi(x, y)$ by

$$\pi(x, y) = (\alpha(x, y) \vee x \approx y) \wedge \bigwedge_{1 \leq k' \leq n} (\neg \beta_{k'}(x, y) \vee x \approx y \vee f_{k'} x \approx y), \quad (3)$$

ϕ_1 is evidently equisatisfiable with

$$\phi_2 = \forall x \forall y \pi(x, y) \wedge \bigwedge_{1 \leq k \leq n} \forall x (\beta_k(x, f_k x) \wedge x \not\approx f_k x), \quad (4)$$

Thus we have eliminated all occurrences of y bound by $\exists!$ -quantifiers.

Step2: (Two-variable resolution).

Definition 7. A *proper two-variable literal* is a literal whose predicate is not $\approx$, and which contains both variables x and y .

In this step, we eliminate proper two-variable literals. Let the formula $\pi(x, y)$ be written as the conjunction of a set of clauses

$$Z(x, y) = \{\zeta_1(x, y), \dots, \zeta_M(x, y)\},$$

and decompose each $\zeta_i(x, y)$ ($1 \leq i \leq M$) as:

$$\zeta_i(x, y) = \zeta_i^+(x, y) \vee \zeta_i^-(x, y), \quad (5)$$

where all of the literals in $\zeta_i^+(x, y)$ are proper two-variable literals, and none of the literals in $\zeta_i^-(x, y)$ is. (Note that the ζ_i^+ may be empty, but the ζ_i^- all contain at least the literal $x \approx y$.)

Let Z^∞ denote the result of saturating Z under resolution, *but only allowing literals in the $\zeta_i^+(x, y)$ to be resolved upon*. Since the clauses $\zeta_i^+(x, y)$ are all non-functional, Z^∞ is a finite set of clauses. Let Z° be the subset of Z^∞ consisting of those clauses none of whose literals is a proper two-variable literal. Finally, construct the formula

$$\phi_3 = \forall x \forall y \bigwedge Z^\circ(x, y) \wedge \bigwedge_{1 \leq k \leq n} \forall x (\beta_k(x, f_k x) \wedge x \not\approx f_k x \wedge \pi(x, f_k x) \wedge \pi(f_k x, x)). \quad (6)$$

Lemma 3. *The formulas ϕ_2 and ϕ_3 constructed above are equisatisfiable.*

Proof. Visibly, $\phi_2 \models \phi_3$. It remains to show that, given a model $\mathfrak{A}$ of ϕ_3 , we can manufacture a model $\mathfrak{A}'$ of ϕ_2 , or, what amounts to the same thing, of

$$\phi'_2 = \forall x \forall y \bigwedge_{1 \leq i \leq M} \zeta_i(x, y) \wedge \bigwedge_{1 \leq k \leq n} \forall x (\beta_k(x, f_k x) \wedge x \not\approx f_k x).$$

Let $\mathfrak{A} \models \phi_3$. Recalling the decomposition (5), if a, b are distinct elements of the domain A , define

$$I_{ab} = \{i \mid 1 \leq i \leq M \text{ and } \mathfrak{A} \not\models \zeta_i^-[a, b]\}.$$

Now define

$$\epsilon_{\{a, b\}}(x, y) = \bigwedge_{i \in I_{ab}} \zeta_i^+(x, y) \wedge \bigwedge_{i \in I_{ba}} \zeta_i^+(y, x).$$

Since every ζ_i^+ is non-functional, so is $\epsilon_{\{a, b\}}$. Moreover, we claim that $\epsilon_{\{a, b\}}$ is consistent. For otherwise, by the completeness of resolution, Z^∞ and hence Z° must contain some clause subsuming

$$\bigvee_{i \in I_{ab}} \zeta_i^-(x, y) \vee \bigvee_{i \in I_{ba}} \zeta_i^-(y, x).$$

On the other hand, by the definition of I_{ab} and I_{ba} , $\mathfrak{A} \not\models \zeta_i^-[a, b]$ for $i \in I_{ab}$ and $\mathfrak{A} \not\models \zeta_i^-[b, a]$ for $i \in I_{ba}$, thus contradicting the assumption that $\mathfrak{A} \models \phi_3$. Hence $\epsilon_{\{a, b\}}$ is consistent as claimed.

We may now modify the structure $\mathfrak{A}$ as follows. Impose some arbitrary total order $<$ on A . For each ordered pair $a, b \in A$ with $a < b$, if

$$\mathfrak{A} \not\models \pi[a, b] \text{ or } \mathfrak{A} \not\models \pi[b, a] \quad (7)$$

change the set of binary predicates satisfied by the pairs a, b and b, a so that a, b satisfies $\epsilon_{\{a, b\}}(x, y)$. This is obviously possible, since $\epsilon_{\{a, b\}}(x, y)$ is consistent and non-functional. Leave every other aspect of $\mathfrak{A}$ unaltered. Let the resulting

structure be $\mathfrak{A}'$. Since these modifications affect only proper two-variable literals, it follows that $\mathfrak{A}' \models \zeta_i^-[a, b]$ if and only if $\mathfrak{A} \models \zeta_i^-[a, b]$ for all $a, b \in A$ and all i ($1 \leq i \leq M$).

We claim that $\mathfrak{A}' \models \phi_2'$. First, we show that, given $a, b \in A$, we have for all i ($1 \leq i \leq M$), $\mathfrak{A} \models \zeta_i[a, b]$. Since every clause in Z contains the literal $x \approx y$, we may assume $a \neq b$. If $i \in I_{ab}$ then, however a and b are ordered, the modifications to $\mathfrak{A}'$ will ensure that $\mathfrak{A}' \models \zeta_i^+[a, b]$. If, on the other hand, $i \notin I_{ab}$ then, by definition, $\mathfrak{A} \models \zeta_i^-[a, b]$, whence, as we have already argued, $\mathfrak{A}' \models \zeta_i^-[a, b]$. Next, we show that, for all k ($1 \leq k \leq n$) and all $a \in A$, $\mathfrak{A}' \models \beta_k[a, f_k^{\mathfrak{A}'}(a)]$. Since $\mathfrak{A} \models \phi_3$, we have $\mathfrak{A} \models \pi[a, f_k^{\mathfrak{A}}(a)]$ and $\mathfrak{A} \models \pi[f_k^{\mathfrak{A}}(a), a]$: in other words, condition (7) fails if $b = f_k^{\mathfrak{A}}(a)$. But in that case, the pair $\langle a, f_k^{\mathfrak{A}}(a) \rangle$ satisfy exactly the same binary and unary predicates in the two structures $\mathfrak{A}$ and $\mathfrak{A}'$. Since $\mathfrak{A} \models \beta_k[a, f_k^{\mathfrak{A}}(a)]$, and since the interpretations of function symbols are the same in $\mathfrak{A}$ and $\mathfrak{A}'$, the result follows. $\square$

This completes second step: in the formula ϕ_3 , there are no proper two-variable literals.

Step 3: (Splitting). In this step, we remove all remaining occurrences of y . Consider again the formula ϕ_3 as given in (6). Since Z° is obtained from Z^∞ by removing any clause containing a proper two-variable literal, we may write each clause in Z° in the form:

$$\zeta(x, y) = \gamma(x) \vee \delta(y) \vee x \approx y \vee \bigvee_{1 \leq i \leq l} y \approx g_i x \vee \bigvee_{1 \leq j \leq m} x \approx h_j y \quad (8)$$

where $\gamma(x)$ is a clause not involving the variable y , $\delta(y)$ is a clause not involving the variable x , l and m are non-negative integers, and the function-symbols g_i ($1 \leq i \leq l$) and h_j ($1 \leq j \leq m$) are chosen from $f_1, \dots, f_n$. (The occurrence of literals $x \approx h_j y$ arises from the fact that variables can be switched as a result of resolution steps used to obtain Z^∞ .) The key idea is to split any formula $\forall x \forall y \zeta(x, y)$ into a disjunction of formulas involving one variable.

Lemma 4. *Let ζ be as given in (8), and suppose $\mathfrak{A} \models \forall x \forall y \zeta$. Then the following two cases are exhaustive:*

1. *There are at most $m + 1$ elements $a \in A$ such that $\mathfrak{A} \models \neg \gamma[a]$.*
2. *There are at most $l \cdot (m + 2)$ elements $b \in A$ such that $\mathfrak{A} \models \neg \delta[b]$.*

Proof. If Case 1 does not obtain, let $a_1, \dots, a_{m+2}$ be distinct elements of A such that $\mathfrak{A} \models \neg \gamma[a_k]$ for all k ($1 \leq k \leq m + 2$). Suppose now $\mathfrak{A} \models \neg \delta[b]$. Since at most $m + 1$ of the a_k can be either equal to b or of the form $h_j^{\mathfrak{A}}(b)$ ($1 \leq j \leq m$), we must have some k and some i ($1 \leq k \leq m + 2$, $1 \leq i \leq l$) such that $b = g_i^{\mathfrak{A}}(a_k)$. Thus, there can be at most $l \cdot (m + 2)$ such b . $\square$

This gives us all we need to perform the splitting.

Lemma 5. Let $\zeta(x, y)$ be as given in (8) and let $c_1, \dots, c_{m+1}$ and $d_1, \dots, d_{l(m+2)}$ be new individual constants. Let $\psi_\zeta^1, \psi_\zeta^2$ be the formulas $\forall x \gamma(x)$ and $\forall x \delta(x)$, respectively, and let ψ_ζ^3 and ψ_ζ^4 be the formulas:

$$\begin{aligned} \psi_\zeta^3 &= \forall x \left(\gamma(x) \vee \bigvee_{1 \leq k \leq m+1} x \approx c_k \right) \\ &\quad \wedge \bigwedge_{1 \leq k \leq m+1} \forall x \left(\delta(x) \vee x \approx c_k \vee \bigvee_{1 \leq i \leq l} x \approx g_i c_k \vee \bigvee_{1 \leq j \leq m} c_k \approx h_j x \right) \\ \psi_\zeta^4 &= \forall x \left(\delta(x) \vee \bigvee_{1 \leq k \leq l \cdot (m+2)} x \approx d_k \right) \\ &\quad \wedge \bigwedge_{1 \leq k \leq l \cdot (m+2)} \forall x \left(\gamma(x) \vee x \approx d_k \vee \bigvee_{1 \leq i \leq l} d_k \approx g_i x \vee \bigvee_{1 \leq j \leq m} x \approx h_j d_k \right). \end{aligned}$$

Then, for all i ($1 \leq i \leq 4$), $\psi_\zeta^i \models \forall x \forall y \zeta(x, y)$; moreover, if $\mathfrak{A} \models \forall x \forall y \zeta(x, y)$, then $\mathfrak{A}$ has an expansion $\mathfrak{A}'$ such that $\mathfrak{A}' \models \psi_\zeta^i$ for some i ($1 \leq i \leq 4$).

Proof. It is completely routine to show that $\psi_\zeta^i \models \forall x \forall y \zeta(x, y)$ for all i ($1 \leq i \leq 4$). For the second part of the Lemma, suppose $\mathfrak{A} \models \forall x \forall y \zeta(x, y)$. If $\mathfrak{A} \models \psi_\zeta^1$ or $\mathfrak{A} \models \psi_\zeta^2$, then then we may expand $\mathfrak{A}$ to $\mathfrak{A}'$ interpreting the constants c_k and d_k in any way whatsoever and we are done. Hence, we may assume that there is at least one a such that $\mathfrak{A} \models \neg \gamma[a]$, and at least one a such that $\mathfrak{A} \models \neg \delta[a]$. Suppose now that Case 1 of Lemma 4 holds. Thus, we are guaranteed at least 1 and at most $m+1$ elements $a \in A$ such that $\mathfrak{A} \models \neg \gamma[a]$. Let $a_1, \dots, a_{m+1}$ be those elements (not necessarily distinct), and let $\mathfrak{A}'$ be the expansion of $\mathfrak{A}$ obtained by interpreting c_k as a_k ($1 \leq k \leq m+1$) and by interpreting the d_k in any way whatsoever. Then it is easy to check that $\mathfrak{A}' \models \psi_\zeta^3$. A similar argument shows that, if Case 2 of Lemma 4 holds, then we can obtain an expansion $\mathfrak{A}'$ of $\mathfrak{A}$ such that $\mathfrak{A}' \models \psi_\zeta^4$. $\square$

Applying Lemma 5 to each $\zeta \in Z^\circ$, we may transform ϕ_3 into the equisatisfiable formula

$$\bigwedge_{\zeta \in Z^\circ} \bigvee_{1 \leq i \leq 4} \psi_\zeta^i \wedge \bigwedge_{1 \leq k \leq n} \forall x (\beta_k(x, f_k x) \wedge x \not\approx f_k x \wedge \pi(x, f_k x) \wedge \pi(f_k x, x)), \quad (9)$$

where the sets of new constants in the formulas ψ_ζ^i are disjoint for distinct clauses ζ . Writing out the definition of $\pi(x, y)$ given in (3) and simplifying, we

obtain

$$\begin{aligned} \phi_4 = & \left(\bigwedge_{\zeta \in Z^\circ} \bigvee_{1 \leq i \leq 4} \psi_\zeta^i \right) \wedge \\ & \bigwedge_{1 \leq k \leq n} \forall x \left(\beta_k(x, f_k x) \wedge x \not\approx f_k x \wedge \right. \\ & \quad \alpha(x, f_k x) \wedge \bigwedge_{1 \leq k' \leq n} (\neg \beta_{k'}(x, f_k x) \vee f_{k'} x \approx f_k x) \wedge \\ & \quad \left. \alpha(f_k x, x) \wedge \bigwedge_{1 \leq k' \leq n} (\neg \beta_{k'}(f_k x, x) \vee f_{k'} f_k x \approx x) \right). \quad (10) \end{aligned}$$

This completes the third and final step in the process of eliminating the variable y : ϕ_4 is a Boolean combination of universally quantified formulas involving only the variable x .

Multiplying out all the four-way disjunctions $\bigvee_{1 \leq i \leq 4} \psi_\zeta^i$ in (10), and performing some routine re-arrangement, we obtain an equisatisfiable formula

$$\phi_5 = \bigvee_{1 \leq i \leq 4^{|Z^\circ|}} \forall x \bigwedge \Gamma_i(x), \quad (11)$$

where each $\Gamma_i(x)$ is a set of stellar clauses. We at last have the desired collection of clause sets

$$\mathcal{G}_\phi = \{\Gamma_i \mid 1 \leq i \leq 4^{|Z^\circ|}\}$$

such that ϕ_1 is satisfiable if and only if some clause set in $\mathcal{G}_\phi$ is satisfiable.

The only remaining matter of concern is the size of the signature—say, $(\mathbf{C}, \mathbf{F}, \mathbf{P}', \mathbf{Q}')$ —of the clause-sets $\mathcal{G}_\phi$. If the original formula ϕ has size n , it is evident that $|\mathbf{P}'|$, $|\mathbf{Q}'|$ and $|\mathbf{F}|$ are bounded by a polynomial function of n , and that $|\mathbf{C}|$ and the $|\Gamma_i|$ are bounded by a (singly) exponential function of n . This gives us:

Corollary 1. *Finite satisfiability for C^2 is decidable in nondeterministic exponential time.*

Proof. Theorem 3 and the above translation from C^2 to the stellar fragment. $\square$

Corollary 2 (Pacholski *et al.*). *Satisfiability for C^2 is decidable in nondeterministic exponential time.*

Proof. Theorem 5 and the above translation from C^2 to the stellar fragment. $\square$

References

- [1] Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem*. Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1997.
- [2] Erich Grädel, Martin Otto, and Eric Rosen. Two-variable logic with counting is decidable. In *Proceedings of the 12th IEEE Symposium on Logic in Computer Science*, pages 306–317. IEEE Online Publications, 1997.
- [3] Leszek Pacholski, Wiesław Szwał, and Lidia Tendera. Complexity of two-variable logic with counting. In *Proceedings of the 12th IEEE Symposium on Logic in Computer Science*, pages 318–327. IEEE Online Publications, 1997.
- [4] Leszek Pacholski, Wiesław Szwał, and Lidia Tendera. Complexity results for first-order two-variable logic with counting. *SIAM Journal on Computing*, 29(4):1083–1117, 1999.
- [5] Christos H. Papadimitriou. On the complexity of integer programming. *Journal of the Association for Computing Machinery*, 28(4):765–768, 1981.
- [6] Dana Scott. A decision method for validity of sentences in two variables. *Journal of Symbolic Logic*, 27:477, 1962.

Monodic packed fragment with equality is decidable

Ian Hodkinson*

Department of Computing, Imperial College, 180 Queen's Gate, London SW7 2BZ, UK
imh@doc.ic.ac.uk <http://www.doc.ic.ac.uk/~imh>

Abstract

We prove decidability of satisfiability of sentences of the monodic packed fragment of first-order temporal logic with equality and connectives Until and Since, in models with various flows of time and domains of arbitrary cardinality. We also prove decidability over models with finite domains, over flows of time including the real order.

1 Introduction

In [6], a new ‘monodic’ fragment of predicate temporal logic was introduced. It consists of those formulas whose subformulas beginning with a temporal connective have at most one free variable. It was shown that if the first-order part of the language was also restricted to (roughly speaking) a decidable fragment of first-order logic, then satisfiability of monodic sentences was decidable over a wide range of (strict linear) flows of time.

The proof in [6] of decidability of a monodic sentence ϕ relied on representing a temporal model of ϕ by a ‘quasimodel’. In a quasimodel, each element of the domain at each time is represented only by its *type* — the set of subformulas $\psi(x)$ of ϕ that are true at the element at that time. The quasimodel records which types are realized at each time. (The integrity of domain elements over time is represented in the quasimodel by functions called runs that select a type at each time; monodicity ensures that this is sufficient.) A quasimodel is simpler than a model because there are only finitely many types; assuming decidability of the underlying first-order logic, it followed that the existence of a quasimodel was decidable over a wide range of flows of time.

The crucial lemma in the proof was of course that ϕ has a temporal model iff it has a quasimodel, and the crucial part of this was to construct a model of ϕ from a quasimodel. To show that the runs in a quasimodel could be turned into the points of the domain of a temporal model involved replacing a first-order structure M by a larger one, $M^{(\kappa)}$, for some infinite cardinal κ . $M^{(\kappa)}$ should realize the same types as M , but each of them should be realized by κ elements of $M^{(\kappa)}$. This was done by excluding equality from the logic. Then, every element of M could simply be ‘duplicated’ κ times to form $M^{(\kappa)}$. Lacking equality, the logic could not see what had happened.

In the presence of equality, this trick no longer works. As a trivial example, $\forall y(y = x)$ is not preserved by duplicating elements. Indeed, it was shown in [9] that with equality, the monodic validities over natural numbers time are not even recursively enumerable, and undecidability results for the monadic monodic fragment with only two variables but with equality are proved in [2].

In this paper, we show that the decidability argument can be resurrected if we limit the first-order part of monodic formulas to come from the *packed fragment* of first-order logic, defined by

*I thank Michael Zakharyashev for valuable conversations and encouragement.

Marx in [7]. The packed fragment is more expressive than the guarded fragment and (for non-binary signatures) the loosely guarded fragment; the clique-guarded fragment of [3] is a syntactic variant of it. In spite of this rather good expressive power, it is still amenable to duplication of elements. For example, in a relational signature, the *disjoint union* of κ copies of a structure M realizes the same types as M , if these types consist of packed formulas. But simply duplicating (some) elements of M as before (see proposition 7 below) works in the presence of constants, and also serves to prove decidability over temporal models with finite domains.

In section 3.3 we will prove:

Theorem 1 (decidability) *Let C be any of $\{(\mathbb{N}, <)\}$, $\{(\mathbb{Z}, <)\}$, $\{(\mathbb{Q}, <)\}$, the class of all finite strict (i.e., irreflexive) linear orders, or any first-order definable class of strict linear orders. It is decidable whether a sentence of the monodic packed fragment with Until, Since, and equality in a signature without function symbols has a model with flow of time in C .*

Theorem 2 (finite domain decidability) *Let C be any of the above, or $\{(\mathbb{R}, <)\}$. It is decidable whether a sentence of the monodic packed fragment with Until, Since, and equality in a signature without function symbols has a model with finite domain and flow of time in C .*

Section 3 contains the proofs, and section 2 the preliminary definitions.

Notation 3 We will usually identify (notationally) a structure M with its domain. We will write c^M for the interpretation in M of a constant c of the signature of M .

2 Monodic packed fragment with equality

The following combines the definition of monodic formulas from [6] with a minor modification of Marx's definition of the packed fragment.

Definition 4 (monodic packed fragment) *Let L be a signature with at most constants and relation symbols (no function symbols). An L -formula γ is said to be a packing guard if γ is a conjunction of atomic and existentially-quantified atomic formulas (possibly equalities) such that any two distinct free variables of γ co-occur free in some conjunct of γ . The monodic packed fragment of predicate temporal logic in signature L consists of the following formulas:*

- Any atomic L -formula (which can be an equality, $\top$, or $\perp$) is monodic packed.
- Boolean combinations of monodic packed formulas are monodic packed.
- If γ is a packing guard, ϕ is a monodic packed formula, every free variable of ϕ is free in γ , and $\bar{y}$ is a tuple of variables, then $\exists \bar{y}(\gamma \wedge \phi)$ is a monodic packed formula.
- If x is a variable and ϕ, ψ are monodic packed formulas with free variable at most x , then $\phi U \psi$ and $\phi S \psi$ are monodic packed formulas. We read them as ‘ ϕ until ψ ’, ‘ ϕ since ψ ’, respectively.

The packed fragment $\text{PF}(L)$ of the first-order language over L consists of those monodic packed L -formulas that do not involve U or S .

Note that $\text{PF}(L)$ is closed under subformulas, including subformulas of packing guards.

Definition 5 (temporal structures and semantics) Let L be as above and let T be a strict linear order. A temporal L -structure with flow of time T is a pair $\mathfrak{M} = (D, (M_t)_{t \in T})$, where D is a non-empty set (the domain of $\mathfrak{M}$), each M_t is an L -structure with domain D , and $c^{M_t} = c^{M_u}$ for all constants $c \in L$ and all $t, u \in T$.

Given $\mathfrak{M}$, $t \in T$, and an assignment h of variables into D , we define the semantics of monodic packed formulas written with variables from the domain of h as follows:

- For atomic α , we let $\mathfrak{M}, t, h \models \alpha$ iff $M_t, h \models \alpha$.
- $\mathfrak{M}, t, h \models \neg \phi$ iff $\mathfrak{M}, t, h \not\models \phi$, and $\mathfrak{M}, t, h \models \phi \wedge \psi$ iff $\mathfrak{M}, t, h \models \phi$ and $\mathfrak{M}, t, h \models \psi$.
- $\mathfrak{M}, t, h \models \exists x \phi$ iff $\mathfrak{M}, t, g \models \phi$ for some assignment g that agrees with h except perhaps on x .
- $\mathfrak{M}, t, h \models \phi U \psi$ iff there is $u \in T$ with $u > t$, $\mathfrak{M}, u, h \models \psi$, and $\mathfrak{M}, v, h \models \phi$ for all $v \in T$ with $t < v < u$.
- $\mathfrak{M}, t, h \models \phi S \psi$ iff there is $u \in T$ with $u < t$, $\mathfrak{M}, u, h \models \psi$, and $\mathfrak{M}, v, h \models \phi$ for all $v \in T$ with $u < v < t$.

Note that our temporal structures have constant domains and the interpretations of constants and assignments to variables are rigid. Also, we use the ‘strict’ semantics of Until and Since ($u > t$ rather than $u \geq t$, etc). They can express connectives such as the non-strict and weak Until and Since, Tomorrow, etc. Indeed, [6] proves expressive completeness results for monodic formulas written with them, over strict Dedekind-complete linear orders.

Notation 6 The notation $\psi(\bar{x})$ will indicate that all free variables of formula ψ are in the tuple $\bar{x}$; not all variables in $\bar{x}$ need actually occur free in ψ .

Let h assign a tuple $\bar{a}$ of elements of D (we write ‘ $\bar{a} \in D$ ’ to indicate this) to the variables $\bar{x}$. For a formula $\psi(\bar{x})$, we may write $\mathfrak{M}, t \models \psi(\bar{a})$ rather than $\mathfrak{M}, t, h \models \psi$. If ψ is a sentence, we just write $\mathfrak{M}, t \models \psi$. Similar notation is used for first-order structures too.

3 Decidability of monodic packed fragment with equality

We now give a decision procedure for the monodic packed fragment.

3.1 Types in the packed fragment

Fix a sentence ϕ of the monodic packed fragment written with at most constants and relation symbols (no function symbols). Let L be the finite signature consisting of all the constants and relation symbols that occur in ϕ . For each subformula $\alpha U \beta$ of ϕ , which by monodicity of ϕ has either one or no free variables, introduce a new unary (respectively, nullary) relation symbol $R_{\alpha U \beta}$. Do the same for subformulas $\alpha S \beta$ of ϕ . Let $\bar{L}$ denote the augmentation of L by these new relation symbols.

For each subformula ψ of ϕ , let $\bar{\psi}$ be the $\bar{L}$ -formula obtained from ψ by replacing all maximal subformulas $\alpha U \beta(y)$ of ψ with one free variable, say y , by $R_{\alpha U \beta}(y)$, replacing all maximal sub-sentences $\alpha U \beta$ of ψ by $R_{\alpha U \beta}$, and similarly for S . Note that $\bar{\psi}$ is in $\text{PF}(\bar{L})$ and has the same free variables as ψ .

Fix a variable x . For a packed formula $\psi(y)$, choose a formula $\psi(x/y)$ obtained by renaming the variables in ψ in a one-one way so that at most x is free. $\psi(x/y)$ is packed.

Let $C_x \phi$ denote the set of all formulas $\bar{\psi}(x/y)$ and their negations, where $\psi(y)$ is a subformula of ϕ , together with all $\bar{L}$ -formulas $x = c$ and $x \neq c$ for constants $c \in L$. Note that $C_x \phi \subseteq \text{PF}(\bar{L})$.

A *type* is a maximal boolean-consistent subset of $C_x\phi$. There are finitely many types. A type p is said to be a *constant type* if $x = c \in p$ for some constant $c \in L$. For an $\bar{L}$ -structure M and $a \in M$, we define $tp_M(a) = \{\psi(x) \in C_x\phi : M \models \psi(a)\}$. An element $a \in M$ *realizes* a type p if $p = tp_M(a)$.

Proposition 7 *Let N be any $\bar{L}$ -structure. Let ξ be a map assigning a cardinal $\xi(p)$ to each type p , such that:*

- $\xi(p) = 0$ if p is not realized in N ,
- $\xi(p) = 1$ if p is a constant type that is realized in N ,
- $\xi(p) \geq |N|$ otherwise.

Then there is an $\bar{L}$ -structure N^ξ such that each type p is realized in N^ξ by exactly $\xi(p)$ elements.

Proof The idea is to make copies of the elements of the domain of N that are not named by constants. For each $a \in N$ let κ_a be a non-zero cardinal such that $\kappa_c = 1$ for all constants $c \in L$. (We will impose further conditions on the κ_a later.) We use i, j to denote ordinals. The domain of N^ξ will be $\{(a, i) : a \in N, i < \kappa_a\}$. A subset S of this domain is said to be *thin* if whenever $(a, i), (a, j) \in S$ then $i = j$. The $\bar{L}$ -symbols are interpreted in N^ξ as follows:

$$\begin{aligned} c^{N^\xi} &= (c^N, 0) \\ N^\xi \models R((a_1, i_1), \dots, (a_n, i_n)) &\iff \{(a_1, i_1), \dots, (a_n, i_n)\} \text{ is thin} \\ &\quad \text{and } N \models R(a_1, \dots, a_n), \end{aligned}$$

for all constants $c \in \bar{L}$, n -ary relation symbols $R \in \bar{L}$, and elements $(a_1, i_1), \dots, (a_n, i_n)$ of the domain. We let $\pi : N^\xi \rightarrow N$ be the homomorphism taking each $(a, i) \in N^\xi$ to $a \in N$. Fix a set V of variables. If $\eta : V \rightarrow N^\xi$ is an assignment, then $\eta' \stackrel{\text{def}}{=} \pi \circ \eta : V \rightarrow N$ is an assignment of the same variables into N .

Lemma 8 *Let $\eta : V \rightarrow N^\xi$ be an assignment, and assume that the range $\eta(V)$ of η is thin. Then $N^\xi, \eta \models \alpha$ iff $N, \eta' \models \alpha$ for any atomic $\bar{L}$ -formula α written with variables from V .*

Proof Because $\kappa_c = 1$ for all constants c , the set $\eta(V) \cup \{c^{N^\xi} : c \in L \text{ a constant}\}$ is thin. The lemma now follows because the restriction of π to any thin set is a partial isomorphism from N^ξ to N . $\square$

Let $\eta : V \rightarrow N^\xi$ be an assignment with thin range. Let $\bar{y}$ be a tuple of variables in V , and suppose that $h : V \rightarrow N$ agrees with η' except perhaps on $\bar{y}$. Lift h to an assignment $\eta_h : V \rightarrow N^\xi$ by:

$$\eta_h(v) = \begin{cases} \eta(w) & \text{if } h(v) = \eta'(w) \text{ for some } w \in V; \\ (h(v), 0) & \text{otherwise.} \end{cases}$$

Then it is easily checked that η_h is well-defined, $\eta_h(V)$ is thin, $(\eta_h)' = h$, and η_h agrees with η except perhaps on $\bar{y}$.

Lemma 9 *For any packed $\bar{L}$ -formula ψ written with variables from V , and any assignment $\eta : V \rightarrow N^\xi$ such that $\eta(V)$ is thin, we have $N^\xi, \eta \models \psi$ iff $N, \eta' \models \psi$. Hence, for all $(a, i) \in N^\xi$ we have $tp_{N^\xi}((a, i)) = tp_N(a)$.*

Proof We give a syntactic proof, by induction on ψ ; the lemma can also be proved by bisimulations, using [7, fact 4.4]. The result for atomic ψ follows from lemma 8. The boolean cases are easy and we omit them. So consider a packed formula θ of the form $\exists \bar{y}(\gamma \wedge \psi)$, where γ is a packing guard and ψ is a packed formula for which the claim is assumed inductively.

Assume that $N, \eta' \models \theta$. So there is an assignment $h : V \rightarrow N$ agreeing with η' except perhaps on $\bar{y}$, and with $N, h \models \gamma \wedge \psi$. By the inductive hypothesis, $N^\xi, \eta_h \models \psi$. To see that $N^\xi, \eta_h \models \gamma$ as well, take any conjunct $\exists \bar{z}\alpha$ of γ ($\bar{z}$ could be empty). Since $N, h \models \gamma$, there is $g : V \rightarrow N$ agreeing with h except perhaps on $\bar{z}$ and with $N, g \models \alpha$. By lemma 8, $N^\xi, (\eta_h)_g \models \alpha$ also, and so $N^\xi, \eta_h \models \exists \bar{z}\alpha$. Hence, $N^\xi, \eta_h \models \gamma \wedge \psi$, whence certainly $N^\xi, \eta_h \models \theta$. Since η and η_h agree on the free variables of θ , we obtain $N^\xi, \eta \models \theta$.

For the converse, assume that $N^\xi, \eta \models \theta$. So there is some $\zeta : V \rightarrow N^\xi$ agreeing with η on the free variables of θ and with $N^\xi, \zeta \models \gamma \wedge \psi$. We may assume that $\bar{y}$ is a non-empty tuple of free variables of γ , and that for some y in $\bar{y}$ we have $\zeta(v) = \zeta(y)$ for every $v \in V$ that does not occur free in γ .

It now follows that $\zeta(V)$ is thin. For, let $v, w \in V$ be distinct variables such that $\zeta(v) = (a, i)$ and $\zeta(w) = (a, j)$. We require $i = j$. By the assumption just made, we can suppose that v, w occur free in γ . So there is a conjunct $\exists \bar{z}\alpha$ of γ in which v, w both occur free. If α is an equality, then since v, w are free in $\exists \bar{z}\alpha$, we have $\zeta(v) = \zeta(w)$, so $i = j$. So suppose that α is of the form $R(x_1, \dots, x_n)$, where $R \in \bar{L}$. We have $N^\xi, \zeta \models \exists \bar{z}R(x_1, \dots, x_n)$, so choose an assignment $\varepsilon : V \rightarrow N^\xi$ agreeing with ζ except perhaps on $\bar{z}$ and with $N^\xi, \varepsilon \models R(x_1, \dots, x_n)$. By definition of N^ξ , $\{\varepsilon(x_1), \dots, \varepsilon(x_n)\}$ must be thin. Since this set contains $(a, i), (a, j)$, it follows again that $i = j$. We are done.

So by the inductive hypothesis, $N, \zeta' \models \psi$. Since π is a homomorphism, $N, \zeta' \models \gamma$. So certainly $N, \zeta' \models \theta$. Since ζ' and η' agree on the free variables of θ , we have $N, \eta' \models \theta$ as required.

That $tp_{N^\xi}((a, i)) = tp_N(a)$ for all $(a, i) \in N^\xi$ is now immediate, since types consist of packed $\bar{L}$ -formulas with at most one free variable, and $\{(a, i)\}$ is thin. $\square$

For each type p realized in N , let R_p be the set of elements of N that realize p . Choose an arbitrary element $a_p \in R_p$, let $\kappa_{a_p} > 0$ be such that $|R_p \setminus \{a_p\}| + \kappa_{a_p} = \xi(p)$, and for all other elements $b \in N$ realizing p let $\kappa_b = 1$. Note that $\kappa_{c_N} = 1$ for each constant $c \in L$. This specifies N^ξ completely.

Let p be a type. If p is not realized in N then $\xi(p) = 0$; and p is not realized in N^ξ , since by lemma 9, $tp_{N^\xi}((a, i)) = tp_N(a)$ for all $a \in N$ and $i < \kappa_a$. If p is realized in N , then by lemma 9, the elements realizing p in N^ξ are precisely $\{(a_p, i) : i < \kappa_{a_p}\} \cup \{(b, 0) : b \in R_p \setminus \{a_p\}\}$. This set has cardinality $\xi(p)$, as required. $\square$

3.2 Quasimodels

The proof of decidability from now on is exceedingly similar to that of [6], using proposition 7 to replace lemma 9 of [6]. We include the proof to make the presentation self-contained.

Definition 10 (model) A temporal L -structure $\mathfrak{M}$ with flow of time T (say) is said to be a model of φ if $\mathfrak{M}, t \models \varphi$ for some $t \in T$.

A *state candidate* (for φ) is a set of types. A *realizable state candidate* is a state candidate of the form $\Sigma = \{tp_M(a) : a \in M\}$, for some $\bar{L}$ -structure M ; M is said to *realize* Σ . A *finitely realizable state candidate* is a state candidate that is realized by a finite $\bar{L}$ -structure.

Now fix a strict (irreflexive) linear order T . Given realizable state candidates Σ_t ($t \in T$), a *run* in $(\Sigma_t)_{t \in T}$ is a choice function ρ on $(\Sigma_t)_{t \in T}$ (so $\rho(t) \in \Sigma_t$ for all $t \in T$), satisfying:

1. for all constants $c \in L$ and all $t, u \in T$, we have $x = c \in \rho(t)$ iff $x = c \in \rho(u)$,

2. for all $t \in T$ and all subformulas $\alpha U \beta(y)$ of ϕ , $\overline{\alpha U \beta}(x/y) \in \rho(t)$ iff there is $u > t$ with $\overline{\beta}(x/y) \in \rho(u)$ and with $\overline{\alpha}(x/y) \in \rho(v)$ for all $v \in T$ with $t < v < u$,
3. a similar clause for S .

Definition 11 (quasimodel) A quasimodel for ϕ (over T) is a pair $((\Sigma_t)_{t \in T}, \mathcal{R})$ where $(\Sigma_t)_{t \in T}$ is a sequence of realizable state candidates and $\mathcal{R}$ is a set of runs in $(\Sigma_t)_{t \in T}$ such that $\Sigma_t = \{\rho(t) : \rho \in \mathcal{R}\}$ for all $t \in T$, and $\overline{\phi} \in \bigcup \Sigma_t$ for some $t \in T$. The quasimodel is finitary if each Σ_t is a finitely realizable state candidate and $\mathcal{R}$ is finite.

Theorem 12 ϕ has a model with flow of time T (and finite domain) iff there is a (respectively, finitary) quasimodel for ϕ over T .

Proof Let $\mathfrak{M} = (D, (M_t)_{t \in T})$ be a model of ϕ with flow of time T . Expand each M_t to an $\overline{L}$ -structure $\overline{M}_t$ by letting $\overline{M}_t \models R_\psi(d)$ iff $\mathfrak{M}, t \models \psi(d)$, for each unary $R_\psi \in \overline{L} \setminus L$ and each $d \in D$, and similarly letting $\overline{M}_t \models R_\psi$ iff $\mathfrak{M}, t \models \psi$, for each nullary $R_\psi \in \overline{L} \setminus L$. Let $\Sigma_t = \{tp_{\overline{M}_t}(d) : d \in D\}$. For each $d \in D$ define $\rho_d : T \rightarrow \bigcup_{t \in T} \Sigma_t$ by $\rho_d(t) = tp_{\overline{M}_t}(d)$. Then $((\Sigma_t)_{t \in T}, \{\rho_d : d \in D\})$ can be checked to be a quasimodel for ϕ over T . If D is finite then clearly the $\overline{M}_t$ and $\{\rho_d : d \in D\}$ are finite, so the quasimodel is finitary.

For the converse, assume that $((\Sigma_t)_{t \in T}, \mathcal{R})$ is a quasimodel for ϕ over T . For each $t \in T$, choose an $\overline{L}$ -structure N_t , finite if possible, that realizes Σ_t . Since there are only finitely many realizable state candidates, we may assume that $\{N_t : t \in T\}$ is finite. Let $\kappa = \max\{|N_t| : t \in T\}$. Define a multiset $\mathcal{R}^{(\kappa)}$ as follows. For each $\rho \in \mathcal{R}$, if some $\rho(t)$ ($t \in T$) is a constant type then put one copy of ρ into $\mathcal{R}^{(\kappa)}$; otherwise, put κ copies of ρ into $\mathcal{R}^{(\kappa)}$. For each type p and each $t \in T$, let

$$\xi_t(p) = |\{\rho \in \mathcal{R}^{(\kappa)} : \rho(t) = p\}|$$

(the right-hand side denotes multiset cardinality, counting multiplicities).

Lemma 13 For each type p and each $t \in T$,

- $\xi_t(p) = 0$ if p is not realized in N_t ,
- $\xi_t(p) = 1$ if p is a constant type that is realized in N_t ,
- $\xi_t(p) \geq |N_t|$ otherwise.

Proof If p is not realized in N_t then $p \notin \Sigma_t$ so there is no $\rho \in \mathcal{R}$ with $\rho(t) = p$. Hence $\xi_t(p) = 0$.

If p is a constant type realized in N_t , then certainly $p \in \Sigma_t$, so there is at least one $\rho \in \mathcal{R}$ with $\rho(t) = p$. Suppose that $\rho, \rho' \in \mathcal{R}$ satisfy $\rho(t) = \rho'(t) = p$. We claim that $\rho = \rho'$. Let $u \in T$; we show that $\rho(u) = \rho'(u)$. Choose a constant $c \in L$ with $x = c \in p$. So $x = c \in \rho(t) \cap \rho'(t)$. As ρ, ρ' are runs, $x = c \in \rho(u) \cap \rho'(u)$. Pick $a, a' \in N_u$ with $tp_{N_u}(a) = \rho(u)$ and $tp_{N_u}(a') = \rho'(u)$. Then $N_u \models a = c \wedge a' = c$, so $a = a'$ and $\rho(u) = \rho'(u)$. As u was arbitrary, $\rho = \rho'$ as claimed.

Hence, there is exactly one $\rho \in \mathcal{R}$ with $\rho(t) = p$. Since p is a constant type, by construction of $\mathcal{R}^{(\kappa)}$ there is exactly one $\rho \in \mathcal{R}^{(\kappa)}$ with $\rho(t) = p$. So $\xi_t(p) = 1$.

Suppose now that p is realized in N_t and is not a constant type. As before, $p \in \Sigma_t$. Choose $\rho \in \mathcal{R}$ with $\rho(t) = p$. Because ρ is a run, no $\rho(u)$ ($u \in T$) is a constant type. So by construction of $\mathcal{R}^{(\kappa)}$, there are κ copies of ρ in $\mathcal{R}^{(\kappa)}$. So $\xi_t(p) = |\{\rho \in \mathcal{R}^{(\kappa)} : \rho(t) = p\}| \geq \kappa \geq |N_t|$. $\square$

By the lemma, proposition 7 applies and we can form the $\bar{L}$ -structure $N_t^{\xi_t}$ for each $t \in T$. For each t , the number of realizations of any type p in $N_t^{\xi_t}$ is the same as the number of $\rho \in \mathcal{R}^{(\kappa)}$ such that $\rho(t) = p$. So we may choose a bijection $\iota_t : \mathcal{R}^{(\kappa)} \rightarrow N_t^{\xi_t}$ such that $tp_{N_t^{\xi_t}}(\iota_t(\rho)) = \rho(t)$ for all $\rho \in \mathcal{R}^{(\kappa)}$. Let $\bar{M}_t$ be the unique $\bar{L}$ -structure with domain $\mathcal{R}^{(\kappa)}$ such that $\iota_t : \bar{M}_t \rightarrow N_t^{\xi_t}$ is an isomorphism. We get $\bar{M}_t$ by pulling back the $\bar{L}$ -structure on $N_t^{\xi_t}$ via ι_t^{-1} . Then $tp_{\bar{M}_t}(\rho) = \rho(t)$ for all $\rho \in \mathcal{R}^{(\kappa)}$ and $t \in T$. So

$$\bar{M}_t \models \psi(\rho) \iff \psi(x) \in \rho(t), \text{ for all } \psi(x) \in C_x\phi, \rho \in \mathcal{R}^{(\kappa)}, t \in T. \quad (1)$$

Let M_t denote the L -reduct of $\bar{M}_t$. Note that for each constant $c \in L$, $t, u \in T$, and $\rho \in \mathcal{R}^{(\kappa)}$, we have $c^{M_t} = \rho$ iff $\bar{M}_t \models \rho = c$, iff $x = c \in \rho(t)$ (by (1)), iff $x = c \in \rho(u)$ (as ρ is a run), iff $\bar{M}_u \models \rho = c$ (by (1) again), iff $c^{M_u} = \rho$. Hence, $c^{M_t} = c^{M_u}$. So we can form the temporal L -structure $\mathfrak{M} = (\mathcal{R}^{(\kappa)}, (M_t)_{t \in T})$, since the interpretation of constants is rigid.

Lemma 14 *For each $t \in T$, each subformula $\psi(\bar{x})$ of ϕ , and each tuple $\bar{\rho} \in \mathcal{R}^{(\kappa)}$, we have*

$$\mathfrak{M}, t \models \psi(\bar{\rho}) \iff \bar{M}_t \models \bar{\psi}(\bar{\rho}).$$

Proof By induction on ψ . For atomic ψ the result is simple, because $\mathfrak{M}, t \models \psi(\bar{\rho})$ iff $M_t \models \psi(\bar{\rho})$; as $\bar{\psi} = \psi$ and $\bar{M}_t$ expands M_t , this is iff $\bar{M}_t \models \bar{\psi}(\bar{\rho})$. The boolean cases are straightforward, noting that $\bar{\psi} \wedge \bar{\chi} = \bar{\psi} \wedge \bar{\chi}$ and $\neg \bar{\psi} = \bar{\neg \psi}$. We consider the case of quantifiers. Assuming the result for $\psi(\bar{x}, y)$, we have $\mathfrak{M}, t \models \exists y \psi(\bar{\rho}, y)$ iff $\mathfrak{M}, t \models \psi(\bar{\rho}, \sigma)$ for some $\sigma \in \mathcal{R}^{(\kappa)}$. Inductively, this is iff $\bar{M}_t \models \bar{\psi}(\bar{\rho}, \sigma)$ for some $\sigma \in \mathcal{R}^{(\kappa)}$. This is iff $\bar{M}_t \models \exists y \bar{\psi}(\bar{\rho}, y)$, iff $\bar{M}_t \models \bar{\exists y \psi}(\bar{\rho}, y)$.

The interesting case is of course the temporal operators. We only do U ; S is similar. Consider a subformula $\alpha U \beta(y)$ of ϕ . Let $\rho \in \mathcal{R}^{(\kappa)}$. We have $\mathfrak{M}, t \models \alpha U \beta(\rho)$ iff there is $u > t$ in T with $\mathfrak{M}, u \models \beta(\rho)$ and $\mathfrak{M}, v \models \alpha(\rho)$ for all $v \in T$ with $t < v < u$. Inductively, this is iff there is $u > t$ in T with $\bar{M}_u \models \bar{\beta}(\rho)$ and $\bar{M}_v \models \bar{\alpha}(\rho)$ for all $v \in T$ with $t < v < u$. It follows from (1) that this holds iff there is $u > t$ in T with $\bar{\beta}(x/y) \in \rho(u)$ and $\bar{\alpha}(x/y) \in \rho(v)$ for all $v \in T$ with $t < v < u$. As ρ is a run, this is iff $\bar{\alpha U \beta}(x/y) \in \rho(t)$. By (1) again, this is iff $\bar{M}_t \models \bar{\alpha U \beta}(\rho)$, as required. $\square$

Since $((\Sigma_t)_{t \in T}, \mathcal{R})$ is a quasimodel for ϕ , there is $t \in T$ and $p \in \Sigma_t$ with $\bar{\phi} \in p$. Choose $\rho \in \mathcal{R}$ with $\rho(t) = p$. There is at least one copy of ρ in $\mathcal{R}^{(\kappa)}$; choose one, and denote it by ρ too. Then $\bar{\phi} \in \rho(t)$, so by (1), $\bar{M}_t \models \bar{\phi}$. By lemma 14, we obtain $\mathfrak{M}, t \models \phi$. Thus, ϕ has a model. We note that if the given quasimodel is finitary then $\mathcal{R}$ is finite, and we chose the N_t to be finite. Hence, κ is finite and the domain $\mathcal{R}^{(\kappa)}$ of the resulting model $\mathfrak{M}$ is also finite. This completes the proof of theorem 12. $\square$

3.3 Decidability

Lemma 15 *It is decidable whether a state candidate Σ for a sentence ϕ of the monodic packed fragment is realizable.*

Proof Each type $p \in \Sigma$ satisfies $p \subseteq C_x\phi \subseteq \text{PF}(\bar{L})$. Therefore, $\bigwedge p(x) \in \text{PF}(\bar{L})$. Hence, up to logical equivalence, the $\bar{L}$ -sentence

$$\alpha_\Sigma = \left(\bigwedge_{p \in \Sigma} \exists x \bigwedge p(x) \right) \wedge \left(\forall x \bigvee_{p \in \Sigma} \bigwedge p(x) \right)$$

is in $\text{PF}(\bar{L})$ too. (The guards for $\exists x, \forall x$ here can be taken to be $x = x$.) Evidently, Σ is a realizable state candidate iff α_Σ has a model, and this question is decidable [7, 3]. $\square$

We can now prove theorem 1.

Proof [of theorem 1] Let ϕ be a sentence of the monodic packed fragment. We will express the existence of a quasimodel for ϕ by a sentence of monadic second-order logic that can be constructed effectively and uniformly from ϕ .

Form L , $C_x\phi$, etc., as before; this can be done effectively. Introduce a unary predicate variable Q_ψ for each $\psi \in C_x\phi$. If p is a type, let

$$\chi_p(t) = \bigwedge_{\psi \in p} Q_\psi(t) \quad \wedge \quad \bigwedge_{\psi \in (C_x\phi) \setminus p} \neg Q_\psi(t),$$

saying that the $Q_\psi(t)$ define the type p at t . Let ρ denote the conjunction of the following three formulas, saying that the Q_ψ define a run. For brevity, we write $\overline{\alpha U \beta}$ instead of $\overline{\alpha U \beta}(x/y)$, and similarly for $\alpha S \beta, \alpha, \beta$.

$$\begin{aligned} & \forall t u \bigwedge_{x=c \in C_x\phi} (Q_{x=c}(t) \leftrightarrow Q_{x=c}(u)), \\ & \forall t \bigwedge_{\overline{\alpha U \beta} \in C_x\phi} \left(Q_{\overline{\alpha U \beta}}(t) \leftrightarrow \exists u (t < u \wedge Q_{\overline{\beta}}(u) \wedge \forall v (t < v < u \rightarrow Q_{\overline{\alpha}}(v))) \right), \\ & \forall t \bigwedge_{\alpha S \beta \in C_x\phi} \left(Q_{\alpha S \beta}(t) \leftrightarrow \exists u (u < t \wedge Q_{\overline{\beta}}(u) \wedge \forall v (u < v < t \rightarrow Q_{\overline{\alpha}}(v))) \right). \end{aligned}$$

By lemma 15, we may effectively construct from ϕ the finite set $\mathfrak{S}$ of all realizable state candidates for ϕ . For each $\Sigma \in \mathfrak{S}$ let P_Σ be a unary predicate variable. We now define the monadic second-order sentence σ_ϕ as follows:

$$\begin{aligned} & \exists_{\Sigma \in \mathfrak{S}} P_\Sigma \left(\forall t \bigvee_{\Sigma \in \mathfrak{S}} \left[P_\Sigma(t) \wedge \bigwedge_{\Sigma' \neq \Sigma \in \mathfrak{S}} \neg P_{\Sigma'}(t) \right] \quad \wedge \quad \bigvee \{ \exists t P_\Sigma(t) : \Sigma \in \mathfrak{S}, \overline{\phi} \in \bigcup \Sigma \} \right. \\ & \quad \left. \wedge \forall t \bigwedge_{p \in \Sigma \in \mathfrak{S}} \left[P_\Sigma(t) \rightarrow \exists_{\psi \in C_x\phi} Q_\psi \left(p \wedge \chi_p(t) \wedge \forall u \bigwedge_{\Sigma' \in \mathfrak{S}} \left[P_{\Sigma'}(u) \rightarrow \bigvee_{q \in \Sigma'} \chi_q(u) \right] \right) \right] \right). \end{aligned}$$

Evaluated in a flow of time T , σ_ϕ says that the P_Σ ($\Sigma \in \mathfrak{S}$) partition T . Thus, a unique realizable state candidate Σ_t is allotted to every $t \in T$. This, σ_ϕ says, is done so that some type in some Σ_t contains $\overline{\phi}$, and for every $t \in T$ and every type $p \in \Sigma_t$, there exists a run (described by the Q_ψ) in $(\Sigma_t)_{t \in T}$ that hits p at t . Thus, $T \models \sigma_\phi$ iff there exists a quasimodel for ϕ over T — for ‘ $\Rightarrow$ ’, let the quasimodel be $((\Sigma_t)_{t \in T}, \mathcal{R})$, where $\mathcal{R}$ is the set of all runs in $(\Sigma_t)_{t \in T}$. By theorem 12, this is iff ϕ has a model with flow of time T .

Clearly, σ_ϕ is effectively constructible from ϕ . So to decide whether ϕ has a model with flow of time T , we only need decide whether $T \models \sigma_\phi$, and this can be done by applying standard results on decidability of monadic second-order logic.

We use this idea to establish theorem 1. We will be brief; see [6] for more details. The statements ‘ $(\mathbb{N}, <) \models \sigma_\phi$ ’ and ‘ $(\mathbb{Z}, <) \models \sigma_\phi$ ’ are decidable, by a result of Büchi [1]. It also follows from Büchi’s theorem that it is decidable whether σ_ϕ is true in some finite linear order. The decidability of ‘ $(\mathbb{Q}, <) \models \sigma_\phi$ ’ follows from Rabin’s theorem on the decidability of S2S [8].

Finally, let $\mathcal{F}$ be a class of strict linear orders definable by a first-order sentence χ of linear order. By considering the ‘standard translation’ of ϕ into two-sorted first-order logic and applying the downward Löwenheim–Skolem–Tarski theorem, it can be seen that ϕ has a model with flow of time in $\mathcal{F}$ iff it has a model with *countable* flow of time in $\mathcal{F}$. This is iff σ_ϕ is true in some countable order

in $\mathcal{F}$. By relativising σ_ϕ to a unary predicate variable which is stated to satisfy χ , we can obtain a new monadic second-order sentence $\sigma_\phi^{(\chi)}$, effectively constructible from ϕ , and such that σ_ϕ is true in some countable order in $\mathcal{F}$ iff $(\mathbb{Q}, <) \models \sigma_\phi^{(\chi)}$. By Rabin's theorem, this last statement is decidable. $\square$

[6] gives a decision procedure over $(\mathbb{N}, <)$ of elementary complexity.

Proof [of theorem 2] Using theorem 12 and the arguments of [6, theorem 36, corollary 37]¹, it suffices to show that it is decidable whether a state candidate Σ is finitely realizable. But Σ is finitely realizable iff α_Σ (as in lemma 15) has a finite model. The packed fragment has the finite model property [4, 5], so this is iff α_Σ has a model. The result now follows from the decidability of the packed fragment. $\square$

Remark 16 Since the packed fragment has the finite model property, any realizable state candidate is finitely realizable. However, not every satisfiable monodic packed sentence is true in some model with finite domain. It is readily checked that the monodic packed sentence

$$\phi = \Box \exists x (P(x) \wedge \neg(\top S P(x)))$$

is satisfied in the model $\mathfrak{M} = (\mathbb{N}, (M_n)_{n \in \mathbb{N}})$ with flow of time $(\mathbb{N}, <)$, domain $\mathbb{N}$, and with $P^{M_n} = \{n\}$, but is false in all models over $(\mathbb{N}, <)$ with finite domains. Here, $\Box\psi$ abbreviates $\psi \wedge \neg(\top U \neg\psi \vee \top S \neg\psi)$, stating that ψ is always true. Cf. [6, theorem 25]. The set of runs in any quasimodel for ϕ over an infinite flow of time must be infinite.

References

- [1] J.R. Büchi, *On a decision method in restricted second order arithmetic*, Logic, Methodology, and Philosophy of Science: Proc. 1960 Internat. Congress, Stanford University Press, 1962, pp. 1–11.
- [2] A Degtyarev, M Fisher, and A Lisitsa, *Equality and monodic first-order temporal logic*, *Studia Logica*, to appear.
- [3] E Grädel, *Decision procedures for guarded logics*, Automated Deduction – CADE16, LNCS, vol. 1632, Springer-Verlag, 1999, Proc. 16th Internat. Conf. on Automated Deduction, Trento, 1999, pp. 31–51.
- [4] I Hodkinson, *Loosely guarded fragment of first-order logic has the finite model property*, *Studia Logica* (2001), to appear. See <http://www.doc.ic.ac.uk/~imh/papers/lgf.ps.gz>
- [5] I Hodkinson and M Otto, *Finite conformal hypergraph covers, with two applications*, preprint (2001). See <http://www-compsci.swan.ac.uk/~csmartin/papers/H0.ps>
- [6] I Hodkinson, F Wolter, and M Zakharyashev, *Decidable fragments of first-order temporal logics*, *Ann. Pure Appl. Logic* **106** (2000), 85–134.
- [7] M. Marx, *Tolerance logic*, *J. Logic, Language and Computation* **10** (2001), 353–373.
- [8] M O Rabin, *Decidability of second order theories and automata on infinite trees*, *Trans. Amer. Math. Soc.* **141** (1969), 1–35.
- [9] F Wolter and M Zakharyashev, *Axiomatizing the monodic fragment of first-order temporal logic*, *Ann. Pure Appl. Logic*, to appear. See <http://www.dcs.kcl.ac.uk/staff/mz/apal.ps>

¹Minor alterations are needed because the definitions have changed to accommodate equality; this has allowed a simpler definition of ‘state candidate’ than in [6].