Decidable Logics via Automata

Michael Benedikt, revised from joint notes with Michael Vanden Boom

1 Introduction

These notes will deal with reasoning problems on logical formulas. Given a sentence φ in some logical language L, the goal is to determine whether there is some structure in which φ holds (the *satisfiability problem* for L). For the logics we consider, an algorithm for the satisfiability problem can easily be converted into an algorithm to determine whether a sentence φ holds in all structures (the *validity problem* for L).

Results from the first-half of the 20^{th} century show that the satisfiability problem for many well-known logics, such as first-order predicate logic and fixpoint logic, are undecidable. But there are many restrictions of these logics that have a decidable satisfiability problem. These notes will overview one of the main techniques for showing decidability, via the use of automata.

We will apply the approach to several logics. The basic idea is to start with a result saying that one can decide whether certain logical sentences in the vocabulary of labelled graphs hold in a tree. This *tree satisfiability procedure* is due to Rabin.

These notes will overview a well-known method that "bootstraps" Rabin's result to get a procedure that determines whether a formula in a logic L is satisfiable. The technique proceeds by showing:

- L has the *tree-like model property*: every satisfiable sentence has a tree-like model. What this means formally will depend on the logic, but informally it means a model that can be "coded" as a tree.
- For each sentence φ of L, one can find another sentence φ^* such that φ holds on a tree-like structure if and only if φ^* holds on the associated tree-code.

Applying these two steps we have reduced checking satisfiability of φ over any structure to checking satisfiability of φ^* over trees, which we can decide using the tree satisfiability procedure.

After executing this two-step procedure for a logic L, we will then "optimize" the process. Underlying the tree satisfiability procedure is another translation: one that takes a logical sentence φ^* and produces a *tree automaton* A^* that captures the behavior of φ^* over trees. A tree automaton is a device for computing over trees. It is (relatively) easy to tell whether φ^* is satisfiable by looking at A^*. Our optimized procedure will take φ and directly produce a tree automaton

A^*, without going through a logic-to-automaton translation hidden in the tree satisfiability theorem.

Note. These are **draft notes**, and may contain mistakes. Please do report errors to the author.

1.1 Bibliographic remarks and suggestions for further reading

The basic approach outlined in these notes is overviewed in Vardi's article [Vardi, 1998]. This is a good place to start for readers looking for a longer introduction. A later follow-up article is Grädel's [Grädel, 1999c].

A reader looking for more background may also want to do some preliminary reading about bisimulations and unravellings, since these play a large role in these notes. The use of bisimulations in analyzing logics is well-established in the literature. Some previous expositions include Grädel's [Grädel, 2002, 1999a] and Grädel and Otto's [Grädel and Otto, 2014].

2 Logic basics

In these notes we will deal with first-order logic without function symbols. To describe the syntax of this language, we need to specify the "non-logical symbols" that can be used. A *signature for function-free first-order logic* consists of:

- A finite collection of *relation names* (or simply *relations* or *predicates* henceforward), with each relation R associated with an *arity*, denoted $\mathsf{arity}(R)$.
- A finite collection of *constants* ("Smith", 3, ...).

For short, we will refer to this as just a *signature* or *vocabulary*.

Example 1. *Our signature might consist of only a relation symbol G of arity 2, two unary relations U and V of arity 1, and no constants. This is an appropriate vocabulary for talking about certain* labelled graphs *or* labelled directed graphs*: G represents the edges of the graph, while U and V represent node labels.*

Syntax of first-order logic. We now review the syntax of function-free first-order logic. We will be using standard terminology to describe formulas, the notion of free variable, quantifiers, and connectives [Abiteboul et al., 1995, Ebbinghaus and Flum, 1999, Libkin, 2004].

First-order logic is built up from *atomic formulas*, which can be either:

- *relational atoms* of the form $R(\vec{t})$, where R is a relation symbol and each t_i in $\vec{t}$ is either a constant or a variable;
- *equality atoms* of the form $t_i = t_j$, with t_i, t_j either a constant or a variable.

Formulas include atomic formulas and are closed under boolean operations, with formation rules $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, and $\neg\varphi$. They are also closed under existential and universal quantifiers, allowing the inductive formation of formulas $\forall x\ \varphi$ and $\exists x\ \varphi$.

An occurrence of a variable x in a formula φ is said to be *free* if it is not in the scope of some quantifier $\exists x$ or $\forall x$. Otherwise it is said to be *bound*. The free variables of a formula are those with some free occurrence. We write $\varphi(\vec{x})$ to indicate that the free variables are among $\vec{x}$, and write $\mathsf{Free}(\varphi)$ to denote the actual free variables of φ. A formula with no free variables is a *sentence*.

A formula φ whose relations and constants all come from a given signature Sig is said to be a *formula over* Sig. When we want to emphasize the signature, we write $\mathsf{FO}(\mathsf{Sig})$, denoting the set of first-order logic formulas over the signature Sig.

The *equality-free* first-order formulas are built up as above but without equality atoms. Note that the equality atom $x = x$ expresses the formula True that is always true, while the equality formula $\neg x = x$ expresses the formula False that is always false. In equality-free first-order logic, we still want to express these,

so we allow the special atomic formulas False and True to be in equality-free first-order logic as well.

If φ is a formula whose free variables include $\vec{x}$, while $\vec{t}$ is a sequence of constants and variables whose length matches $\vec{x}$, then $\varphi[\vec{x} := \vec{t}]$ denotes the formula obtained by simultaneously substituting each x_i with t_i. When the mapping of the free variables x_i to constants or variables t_i is either clear from context or unimportant, we denote this as $\varphi(\vec{t})$.

Example 2. *Consider the vocabulary with a binary predicate G. Then the following are formulas of first-order logic:*

$$\begin{aligned}\varphi_1 &= G(y, y)\\ \varphi_2 &= \exists y\ G(x, y) \wedge G(y, y)\\ \varphi_3 &= \forall x\ \exists y\ G(x, y) \wedge G(y, y)\\ \varphi_4 &= \exists y_1\ y_2\ y_3\ G(y_1, y_2) \wedge G(y_2, y_3) \wedge G(y_3, y_1)\end{aligned}$$

Informally, the first is a formula specifying elements in a directed graph that have a self-loop. The second is a formula specifying elements that have an edge to a node with a self-loop. The third specifies directed graphs in which each element is adjacent to a self-loop, while the fourth specifies directed graphs which contain a three-clique. The last two are sentences.

Structures. A *structure* for a signature consists of a set called the *domain* of the structure, interpretations for each relation as sets of tuples with values in the domain, and an interpretation for each constant as a single element of the domain. For a structure M, $\mathsf{domain}(M)$ denotes its domain, and for a relation symbol G, $M(G)$ denotes the interpretation of G in M.

Example 3. *We return to the signature of Example 1. One possible structure for the signature is M_1:*

$$\begin{aligned}\mathsf{domain}(M_1) &= \{1, 2, 3\}\\ M_1(G) &= \{(1, 2), (2, 3), (3, 1)\}\\ M_1(U) &= \{1, 3\}\\ M_1(V) &= \emptyset\end{aligned}$$

Another structure M_2 is specified as:

$$\begin{aligned}\mathsf{domain}(M_2) &= \mathbb{N}, \textit{the natural numbers}\\ M_2(G) &= \{(i, j) : i < j\}\\ M_2(U) &= \{1\}\\ M_2(V) &= \{17\}\end{aligned}$$

Semantics. We now explain the meaning of a first-order logic formula. The function Eval takes as argument a formula φ, a structure M, and a *variable*

binding for φ in M: a mapping σ taking each free variable of φ to an element of the domain of M. The function is defined by induction on the structure of the formula φ.

- $\mathsf{Eval}(x = y, M, \sigma)$ is true iff $\sigma(x) = \sigma(y)$
- $\mathsf{Eval}(R(x_1 \dots x_n), M, \sigma)$ is true iff the tuple $\langle \sigma(x_1) \dots \sigma(x_n) \rangle$ is in $M(R)$, where $M(R)$ denotes the interpretation of R in structure M
- $\mathsf{Eval}(\varphi_1 \wedge \varphi_2, M, \sigma)$ is true iff both $\mathsf{Eval}(\varphi_1, M, \sigma)$ is true and $\mathsf{Eval}(\varphi_2, M, \sigma)$ is true
- $\mathsf{Eval}(\neg\varphi, M, \sigma)$ is true iff $\mathsf{Eval}(\varphi, M, \sigma)$ is false
- $\mathsf{Eval}(\exists x\ \varphi, M, \sigma)$ is true iff there is some value c in $\mathsf{domain}(M)$ such that: letting $\sigma' = \sigma + x \mapsto c$, we have $\mathsf{Eval}(\varphi, M, \sigma')$ is true (if x is already bound in σ, then σ' overwrites this binding with c).

Example 4. *Consider the sentence φ_3 from Example 2:*

$$\forall x\ \exists y\ G(x, y) \wedge G(y, y)$$

Consider the structure M_1 from Example 3. Intuitively, the sentence is false in M_1, since there are no self-loops. We can make this precise using the semantics.

There is clearly no binding σ such that $M_1, \sigma \models G(y, y)$, since there is no pair of the form $(y_0, y_0) \in M_1(G)$. Using the inductive definitions for $\wedge$ and $\exists$, we see that there is no binding σ such that $M_1, \sigma \models G(x, y) \wedge G(y, y)$, and thus no binding such that $M_1, \sigma \models \exists y\ G(x, y) \wedge G(y, y)$. But the inductive semantics of $\forall x$ requires that for every $x_0 \in \mathsf{domain}(M_1)$, the formula $\varphi_2 = \exists y\ G(x, y) \wedge G(y, y)$ holds in M_1 with the binding mapping x to x_0. In particular, this must hold for the binding σ_1 mapping x to 1, which is a contradiction.

We will often use "turnstile notation" for the semantics. That is, instead of "$\mathsf{Eval}(\varphi, M, \sigma)$ is true", we write $M, \sigma \models \varphi$. Again, when the ordering of variables in a valuation σ obvious, we often write the valuation as just a list of values. In particular, if the free variables are $x_1 \dots x_n$ in a formula $\varphi(x_1 \dots x_n)$, and $c_1 \dots c_n$ are values, then we will use "turnstile notation" combined with this ordering convention: we write $M, c_1 \dots c_n \models \varphi$, or $M, c_1 \dots c_n$ satisfies φ, instead of writing that $\mathsf{Eval}(\varphi, M, x_1 \mapsto c_1 \dots x_n \mapsto c_n) = \mathsf{True}$. We will also sometimes write "$\varphi(c_1, \dots c_n)$ holds in M" (again using the fact that the mapping of variables to constants is obvious).

A *fact* consists of a relation R and a tuple $\vec{t}$ whose arity is the arity of R. We write such a fact as $R(\vec{t})$. A fact is said to be true in a structure if $\vec{t}$ is in the interpretation of R in the structure.

The semantics of a first-order logic formulas depends in general on both the interpretations of the symbols and the domain. However, some formulas are *domain-independent*: their truth or falsity does not depend on the domain, but only on the interpretations.

Example 5. *The sentence $\exists x\ A(x)$ is domain-independent: its truth depends only on the interpretation of the relation A.*

The sentence $\forall x\ A(x)$ is not domain-independent: knowing the interpretation of A alone will not allow us to know if the sentence is true.

Special kinds of FO formulas. Formulas that are built up as above but disallowing $\neg$ or $\forall$ will be called *positive existential formulas*, or $\exists^+$ *formulas* for short.

A subset of the $\exists^+$ formulas are the *conjunctive queries* (CQs), which are of the form

$$\varphi(\vec{x}) = \exists \vec{y}\ (A_1 \wedge \cdots \wedge A_n)$$

where A_i is a relational atom with arguments that are either variables from $\vec{x}$ and $\vec{y}$ or constants. A normalization argument shows that any logical formula built up using $\wedge$ and $\exists$ can be expressed as a CQ. For a conjunctive query Q, a variable binding that witnesses that Q holds in an instance I will also be called a *homomorphism of Q in I*.

A *unions of conjunctive queries* (UCQ), is a disjunctions of CQs in which every CQ has the same free variables. Another simple normalization argument shows that any $\exists^+$ sentence can be converted to a UCQ.

Example 6. *Consider a schema that includes the relation* UEmployee *containing ids of each university employee, along with relations* Researcher *containing the same information but only about researchers, and a relation* Lecturer *containing the same information about lecturers. The sentence stating that there is either a researcher or a lecturer can be written as a UCQ:* $(\exists x\ \mathsf{Researcher}(x)) \vee (\exists x\ \mathsf{Lecturer}(x))$.

A formula identifying the researchers that are also lecturers can be expressed as a CQ:

$$\mathsf{Researcher}(x) \wedge \mathsf{Lecturer}(x)$$

$\exists^+$ formulas are closely connected to the notion of a *homomorphism*. A homomorphism between structures I and I' is a function h from the domain of I to the domain of I' such that for any relation R of arity n and any tuple $c_1 \ldots c_n$:

$I \models R(c_1 \ldots c_n)$ implies that $I' \models R(h(c_1) \ldots h(c_n))$

Informally a homomorphism is a function that preserves facts. An easy observation is that homomorphisms preserve $\exists^+$ formulas. That is, if h is a homomorphism from I to I' and φ is $\exists^+$ formula, then $I \models \varphi(c_1 \ldots c_n)$ implies that $I' \models \varphi(h(c_1) \ldots h(c_n))$.

We can extend the notion of homomorphism to take as input a CQ. If φ is a CQ with atoms $A_1 \ldots A_k$ and I is a structure, a *homomorphism of φ in I* is a function mapping variables of φ into I, such that $A_1(x_1 \ldots x_k) \in \varphi$ implies that $I \models A(h(x_1) \ldots h(x_k))$. One can rephrase satisfaction of a CQ in terms of homomorphisms: a conjunctive query $\varphi(\vec{x})$ holds of $\vec{c}$ in I exactly when there is a homomorphism of φ to I mapping each x_i to c_i.

Second-Order Logic. A much more expressive language for describing structures is *Second-Order Logic* (SO) over a vocabulary. We now have relations and constants in the signature, and an infinite collection of first-order variables $(x_1 \ldots)$ and for every number k an infinite collection of second-order variables $(X_1 \ldots)$ of arity k. Atomic formulas are the same as for first-order logic except we also have an atomic formula $S(t_1 \ldots t_k)$ where t is a constant or first-order variable, and S is a second-order variable of arity k.

Formulas are built up inductively from atomic formulas via $\wedge, \vee, \neg$, first-order quantifiers $\exists x$ and $\forall x$ as before, and the quantifications $\exists S\ \varphi$, $\forall S\ \varphi$, where S is a second-order variable.

The semantics extend that of first-order logic. The semantic function now takes as argument a formula φ, model M, and variable binding σ, but σ also maps each second-order variable X of φ of arity k to a subset of $\mathsf{domain}(M)^k$. The induction cases for the new quantifiers are:

- $\mathsf{Eval}(\exists S\ \varphi, M, \sigma)$ for S of arity k is true iff there is some subset S of $\mathsf{domain}(M)^k$ such that letting $\sigma' = \sigma + X \mapsto S$, we have $\mathsf{Eval}(\varphi, M, \sigma')$ is true.
- $\mathsf{Eval}(\forall S\ \varphi, M, \sigma)$ for S of arity k is true iff for each subset S of $\mathsf{domain}(M)^k$, letting $\sigma' = \sigma + X \mapsto S$, we have $\mathsf{Eval}(\varphi, M, \sigma')$.

Example 7. *Consider the vocabulary from Example 1 again, and the following second-order sentence, where S is a second-order variable of arity 1:*

$$\begin{aligned}
&\exists x_0\ x_1 U(x_0) \wedge V(x_1) \wedge \exists S \\
&\qquad S(x_0) \wedge S(y_1) \wedge \\
&\qquad [\forall x\ y(\ (S(x) \wedge G(x,y)) \rightarrow S(y))] \wedge \\
&\neg \exists S_2\ (S_2(x_0) \wedge \exists z\ S(z) \wedge\ negS_2(z)) \wedge (\forall x\ y(\ (S(x) \wedge G(x,y)) \rightarrow S(y)))
\end{aligned}$$

That is, G contains a set with a U element x_0, a V element x_1 in it, and there is a certain subset S containing x_0 and x_1. S is closed under G-successors, and any proper subset of S containing x_0 can not be closed under G-successors. Thus S can only be the closure of x_0 under G-successor. Thus this sentence holds exactly when there is a path from a node labelled U to a node labelled V.

In the structure M_2 of Example 3, the sentence is true: a witness for y is 1, for z is 17, and a witness for S consists of all numbers between 1 and 17.

Consider the following sentence in which R, B and W are second-order variables of arity 1:

$$\begin{aligned}
&\exists R\ B\ W \\
&\forall x\ y\ [G(x,y) \rightarrow (R(x) \vee B(x) \vee W(x))] \wedge \\
&\forall x\ y\ [G(x,y) \rightarrow G(y,x)] \wedge \\
&\forall x\ \neg[(R(x) \wedge B(x)) \vee (W(x) \wedge B(x)) \vee (W(x) \wedge R(x))] \wedge \\
&\forall x\ y\ [R(x) \wedge G(x,y) \rightarrow \neg R(y)] \wedge \\
&\forall x\ y\ [B(x) \wedge G(x,y) \rightarrow \neg B(y)] \wedge \\
&\forall x\ y\ [W(x) \wedge G(x,y) \rightarrow \neg W(y)]
\end{aligned}$$

This holds exactly if graph G has a 3-coloring.

Note that the above example gives one common use of second-order logic: to express properties involving recursive definitions. Recursive definition can actually be captured in a subset of second-order logic, Least fixpoint logic (LFP).

Least Fixpoint Logic. LFP extends first-order logic with second-order variables. But instead of allowing second-order quantification to bind second-order variables, LFP adds a *fixpoint operator*:

Suppose $\varphi \in \mathsf{LFP}$ and φ contains second-order variables $\vec{Y}$ and also a distinguished second-order variable X of arity k along with k first-order variables $\vec{x}$. Suppose further that X only occurs positively in φ: that is, it appears in the scope of an even number of negations. Then:

$$[\mu_{X,\vec{x}}.\varphi(\vec{x}, X, , \vec{Y})](\vec{t})$$

is a formula of LFP. Note that the formula φ may have first-order variables $\vec{v}$ in addition to the variables $\vec{x}$ that are in the scope of the fixpoint.

We define the semantics of this fixpoint operator now. The fact that X occurs positively implies that $\varphi(\vec{x}, X, , \vec{Y})$ is monotone in X. That is, φ induces a monotone operator $U \mapsto \mathcal{O}_{\varphi}^{M,\vec{v},\vec{V}}(U) := \left\{\vec{u} : M, \vec{u}, U, \vec{V} \models \varphi(\vec{x}, X, \vec{Y}\right\}$ on every structure M with valuations $\vec{v}$ and $\vec{V}$ for the remaining second-order variables $\vec{Y}$ of φ. If the domain of the structure is finite, it is clear that if we iterate this operator the resulting sets must reach a fixpoint, since they can not keep increasing. The same holds for infinite structures, by a basic result concerning monotone set functions, the Knaster-Tarski fixpoint theorem. Given some ordinal β, the *fixpoint approximant* $\varphi^{\beta}(M, \vec{V})$ of φ on $M, \vec{V}$ is defined such that

$$\begin{aligned}
\varphi^{0}(M, \vec{V}) &:= \emptyset \\
\varphi^{\beta+1}(M, \vec{V}) &:= \mathcal{O}_{\varphi}^{M,\vec{V}}(\varphi^{\beta}(M, \vec{V})) \\
\varphi^{\beta}(M, \vec{V}) &:= \bigcup_{\beta' < \beta} \varphi^{\beta'}(M, \vec{V}) \quad \text{where } \beta \text{ is a limit ordinal.}
\end{aligned}$$

We let $\varphi^{\infty}(M, \vec{V}) := \bigcup_{\beta} \varphi^{\beta}(M, \vec{V})$ denote the least fixpoint based on this operation, and the least ordinal β such that $\varphi^{\beta}(M, \vec{V}) = \varphi^{\beta+1}(M, \vec{V}) = \varphi^{\infty}(M, \vec{V})$ is called the *closure ordinal*. Thus, $[\mu_{X,\vec{x}}.\varphi(\vec{x}, X, \vec{Y})]$ defines a new predicate named X of arity k for every valuation for $\vec{Y}$, and $M, \vec{u}, , \vec{V} \models [\mu_{X,\vec{x}}.\varphi(\vec{x}, X, \vec{Y})](\vec{z})$ iff $\vec{u} \in \varphi^{\infty}(M, \vec{V})$. If $\vec{V}$ is empty or understood in context, we just write $\varphi^{\infty}(M)$.

Example 8. *Consider the first property from Example 7. We claim that this can be expressed as an* LFP *formula:*

$$\exists z \Big(V(z) \wedge [\mu_{S,y}.U(y) \vee \exists x(G(x, y) \wedge S(x))](z) \Big).$$

To see this, consider the subformula with the fixpoint in it, $\varphi(z) = [\mu_{S,y}.U(y) \vee \exists x(G(x, y) \wedge S(x))](z)$. *We will explain why this formula holds of elements* z

that are G-reachable from a node labelled U. The initial approximant of φ, φ^0, is formed by setting S to be empty. This gets us just the elements that are in U. The next approximant φ^1 is formed by setting S to be φ^0: this gives us all the elements that are in U or reach an element of U in 1 step. Reasoning in inductively, we see that φ^i is the set of elements that reach U in i steps. φ^ω is the union of these sets, thus the set of elements that are reachable from U. We can check that φ^ω is a fixpoint, and thus the least fixpoint is either φ^ω or some φ^n. In either case, we can conclude that the least fixpoint is the set of elements reachable from U.

Satisfiability and validity problems. The main computational problem we consider in these notes is the *satisfiability problem*. A formula $\varphi(\vec{x})$ is *satisfiable* if there is a structure M and binding σ for the variables $\vec{x}$ such that $M, \sigma \models \varphi$.

The satisfiability problem for a logic L takes as input an L-formula and outputs yes if the formula is satisfiable.

The satisfiability problem is related to the *validity problem*: a formula is *valid* if for every structure M and binding σ for the variables $\vec{x}$, $M, \sigma \models \varphi$. The validity problem for a logic is the task of determining whether a formula in the logic is valid. For logics that are closed under negation, one can reduce validity to satisfiability, since φ is valid if and only if $\neg\varphi$ is not satisfiable.

Example 9. *Consider the formulas:*

$$\begin{aligned}
\varphi_1 &= \forall x\ U(x) \vee \neg U(x) \\
\varphi_2 &= \forall x\ W(x) \to \exists y\ (R(x,y) \wedge \neg R(x,y)) \\
\varphi_3 &= (\forall x\ y\ U(x,y) \to \neg U(y,x)) \wedge (\exists z\ U(z,z))
\end{aligned}$$

φ_3 is not satisfiable, since the second conjunct implies there is z_0 is such that $U(z_0, z_0)$, and letting x and y bind to z_0 in the first conjunct we get a contradiction.

φ_1 is valid (and thus satisfiable).

φ_2 is satisfiable: we can take a structure where W is always false. It is not valid since we can take a structure where W holds of some x_0, but R holds of no pairs of elements.

In the satisfiability problem, we looked at whether there is any model M where the formula holds, without restricting M to be finite. If we restrict to witness structures M that are finite we have the *finite satisfiability problem*, and there is a corresponding *finite validity problem*. We will not discuss finite satisfiability or validity in detail in these notes, but for many of the logics discussed here they will agree with the general satisfiability and validity problems.

A fundamental result is that these problems are undecidable for first-order logic:

Theorem 1. *The satisfiability and validity problems for first-order logic are undecidable. The same is true for the finite variants. That is, there cannot be a decision procedure for these problems.*

Thus in looking for decision procedures, we will need to look for restricted subsets of the logics.

Quantifier rank and games. The *quantifier-rank* of a first-order formula φ, written $\mathsf{QR}(\varphi)$ is the number of nested quantifications. That is, a formula with no quantifiers has quantifier-rank 0, while the inductive definition is:

$$\begin{aligned} \mathsf{QR}(\neg\varphi) &= \mathsf{QR}(\varphi) \\ \mathsf{QR}(\varphi_1 \wedge \varphi_2) = \mathsf{QR}(\varphi_1 \vee \varphi_2) &= \max(\mathsf{QR}(\varphi_1), \mathsf{QR}(\varphi_2)) \\ \mathsf{QR}(\exists x\ \varphi) = \mathsf{QR}(\forall x\ \varphi) &= \mathsf{QR}(\varphi) + 1 \end{aligned}$$

So a formula with quantifier-rank 0 has no quantifiers at all. If a formula has quantifier rank 1, in any quantified subformula $\exists x\varphi$ of the formula, φ must have no quantifiers at all.

Example 10. *Returning to Example 2 the formula $\varphi_1 = G(y, y)$ had no quantifiers, so its quantifier rank is 0. The formula $\varphi_2 = \exists y\ G(x, y) \wedge G(y, y)$ had a single quantifier, and thus had quantifier rank 1, while $\varphi_3 = \forall x\ \exists y\ G(x, y) \wedge G(y, y)$ has quantifier rank 2. If we consider*

$$\varphi_5 = \forall x\ (\exists y\ G(x, y) \wedge \exists z\ H(x, z))$$

we see it has 3 quantified variables, but has quantifier-rank 2, since there is only one level of nesting.

In various points in the text we will be interested in showing that two structures M and M' agree on all formulas in a logic. Of course, if we have two specific structures, and the logic had only finitely many formulas in it, we could just check that this "language equivalence" holds whenever M' is generated from M by brute-force, checking equivalence for every formula. But suppose we have to check that M and M' agree on an infinite collection of formulas; for example all formulas in some sublogic of first-order logic. To do this we will use a *game technique*. We define a two-player game that involves annotating elements in M and M'. The game will have two players, called Spoiler and Duplicator. Spoiler makes moves in one of the two structures, by annotating an element e with an identifier p. Normally p is referred to as a "pebble" and this move is called "placing pebble p on element e ". Duplicator needs to somehow "mimic" the moves of Spoiler in the other structure, by annotating an element in the other structure, say e', with the same identifier p. Duplicator wants to ensure that the elements that are pebbled in M look "similar to" the elements on the corresponding pebbles in M'. The details of the game will depend on the logic. For some logics Spoiler can make his move in whichever structure he likes, while in others he must always play in the same structure. How similar e must be to e' can also vary depending on the logic. The rules of the game are arranged so that whenever Duplicator has a way to win, the two structures agree in the logic.

We give a simple example of how one can capture equivalence of a logic with a game, by giving a game that works for first-order logic formulas with a fixed quantifier rank j. That is, suppose we are interested in showing that two structures M and M' agree on all first-order formulas of quantifier-rank j. This can be demonstrated using the *j-round pebble game on M, M'*. A position in this game is given by a sequence $\vec{p}$ of elements from M, and a sequence $\vec{p}'$ of the same length as $\vec{p}$ from M'. We can think of the i^{th} element p_i in the sequence being pebbled by the identifier i. There are two players, Spoiler and Duplicator, and a round of the game at position $(\vec{p}, \vec{p}')$ proceeds by Spoiler choosing one of the structures (e.g. M) and appending an element from that structure to the corresponding sequence (e.g. appending an element from the domain of M to $\vec{p}$). Duplicator must respond by appending an element from the other structure to other sequence (e.g. appending an element from the domain of M' to $\vec{p}'$). A j-round play of the game is a sequence of j moves as above. Duplicator wins the game if the sequences represent a partial isomorphism: $p_i = p_j$ if and only if $p'_i = p'_j$ and for any relation R, $R(p_{m_1} \dots p_{m_j}) \in I$ if and only if $R(p'_{m_1} \dots p'_{m_j}) \in I'$. A strategy for Duplicator is a response to each move of Spoiler. Such a strategy is *winning* from a given position $\vec{p}, \vec{p}'$ if every j-round play emerging from following the strategy, starting at these positions, is not winning for Spoiler. The following result states that the game "captures" agreement of the structures on formulas of quantifier-rank j.

Proposition 1. *If there a winning strategy for Duplicator in the j-round pebble game on M, M' starting at $\vec{p}, \vec{p}'$, then for every formula φ of quantifier-rank at most j satisfied by $\vec{p}$ in M, φ is also satisfied by $\vec{p}'$ in M'.*

Proof. The proof is a simple induction on j. In the induction step we assume Duplicator wins the $j+1$ round pebble game, and we look at formulas of quantifier-rank $j+1$. For such formulas we proceed by structural induction. The interesting cases are the induction step for quantifiers. For example, if we consider a formula $\varphi = \exists x\, \varphi_0$ of quantifier rank $j+1$, then φ_0 must have quantifier rank j. We assume M with valuation $\vec{p}$ for its variables satisfies φ, and we have to show that M', $\vec{p}'$ satisfy φ. Because $M, \vec{p} \models \varphi$ there is $p_{j+1} \in M$ such that $M, \vec{p}, p_{j+1} \models \varphi_0$. Since Duplicator has a winning strategy on $\vec{p}, \vec{p}'$ in the $j+1$ game, there is p'_{j+1} that Duplicator can play in M' such that $\vec{p}, p_{j+1}$ and $\vec{p}' p'_{j+1}$ is a winning position in the j-round game. Now $M, \vec{p}', p'_{j+1} \models \varphi_0$ by induction, and therefore $M, \vec{p}' \models \varphi$ as required. The case of universal quantifiers is similar. □

Example 11. *Consider a vocabulary consisting of a binary relation $G(x, y)$ and two structures M_{16} and M_{17} for it. M_{16} is a chain with 16 elements while M_{17} is a chain with 17 elements. Clearly M_{16} and M_{17} can be distinguished in first-order logic by a long sentence: the sentence stating that there is a chain of length 17 holds in M_{17} but not in M_{16}. Unsurprisingly, these structures agree on sentences that have quantifier-rank at most 3. We can show this by giving a strategy for the Duplicator in the 3-round game.*

Consider the first round of the game, when Spoiler picks one of the two structures M_i and placing a "pebble" on one of the elements e_1 within it. Duplicator measures the distance of this element from both the beginning and the end of the chain, and tries to approximately mimic it in the other structure. If e_1 is "near the beginning" of M_i — within $j < 8$ of the initial element of the structure, then Duplicator responds with an element f_1 that is exactly j edges from the initial element in the other structure. If e_1 is "near the end" of M_i – within $j < 8$ of the final element – then Duplicator plays in the other structure to match the distance to final element. Otherwise — e_1 is "far from both ends" — Duplicator responds with an element f_1 that is far from both ends in the other structure. In the second round, Spoiler picks a structure and marks another element in it, say f_2. Duplicator now measures the distance of f_2 new element from both the ends of the chain and from f_1, and chooses an element e_2 in the other structure to mimic this distance. Since the number of rounds has gone down, Duplicator now just mimics one of these distances exactly if it is $j < 4$; e.g. if e_2 is 3 ahead of e_1, Duplicator choose f_2 that is 3 ahead of f_1. If all the distances are 4 or greater, then Duplicator chooses any element where the distances are 4 or greater. Round 3 is done the same way, except Duplicator defines "far" to be above 2 rather than above 4. Since the two structures are both large enough, one can see that Duplicator can always carry out this strategy.

2.1 Bibliographic remarks and suggestions for further reading

Our treatment of first-order logic can be found in any undergraduate text. Fixpoint logic is described in textbooks on finite model theory [Libkin, 2004, Ebbinghaus and Flum, 1999] and descriptive complexity theory [Immerman, 1999]. These notes will not present the proofs of any *undecidability* results for the satisfiability problem. The reader interested in seeing techniques for proving undecidability for fragments of first-order logic may want to look at [Börger et al., 1997]. The games for first-order logic presented here are discussed at length in [Libkin, 2004, Ebbinghaus and Flum, 1999].

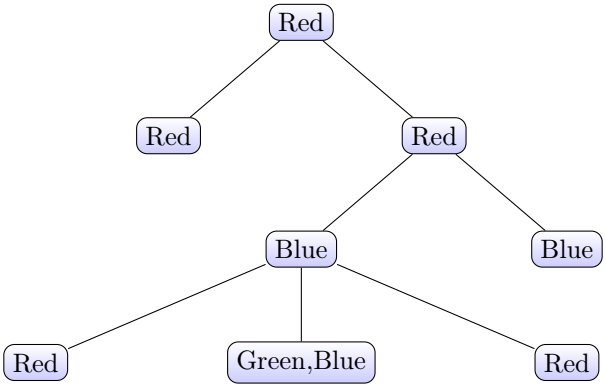

Figure 1: A simple tree

3 Tree and automata preliminaries

Fix a set Σ of unary relations $U_1 \dots U_n$ and a binary relation E. A structure $(V_0, E_0, \dots)$ for such a vocabulary is a *labelled tree* over Σ if (V_0, E_0) forms a tree in the usual sense with labels from $\mathcal{P}(\Sigma)$. Figure 1 shows a finite labeled tree over the labels Red, Blue, Green.

The *outdegree* (also known as the *rank*) of a node in a tree is the number of children. A node with outdegree 0 is a *leaf*. A tree in which every node with non-zero outdegree has the same outdegree is said to be *complete*. A tree in which every node has outdegree 0 or 2 is a *complete binary tree*.

Trees as defined above can have any cardinality. In these notes it will suffice to look only at trees where nodes have at most countably many children. A structure is a *labelled ω-tree over Σ* if it is a labelled tree and each node has only countably children and only finitely many ancestors. We sometimes refer to labelled ω-trees over Σ as Σ-trees for brevity, to emphasize that the labels are subsets of Σ.

A common logic for describing properties of such trees is *Monadic Second-Order Logic* MSO over Σ. It is a special case of Second-Order logic, but where only second-order variables of arity 1 are permitted. Example 7 already gave an example of an MSO formula.

The starting point for this work is the following fundamental decidability result about MSO, which is a variant of a result of Rabin [Rabin, 1969]:

Theorem 2. *There is a computable function that takes as input an* MSO *sentence φ over Σ and decides whether φ holds in some labelled ω-tree.*

We will not need to understand the proof of this result. But the main idea of the proof is to proceed in two steps. The first, more difficult step, is to effectively translate an MSO sentence φ into a *tree automaton* T_φ: a tree automaton is a machine described using a finite set of control states Q, and a

transition function δ; it computes on an infinite tree by associating nodes to states $q \in Q$; the transition function δ restricts how the state assignment of a child relates to the state assignment of a parent. In the end it computes a single value, true or false, as its final output on each tree, depending on properties of the states assigned to each node of the tree during a computation. There are many variants of tree automata, and the details of these will be discussed later in the text. The tree automaton T_φ should be equivalent to φ, in that a tree is accepted by T_φ if and only if it satisfies φ. Thus the first step reduces satisfiability of φ to checking whether T_φ accepts some tree—the *emptiness problem for* T_φ.

The second step is an algorithm to decide the emptiness problem. For general MSO sentences, the process in the first step is very expensive. Specifically, the complexity is *non-elementary* — bigger than any tower of exponentials. And it can be shown that for general MSO sentences, one can not do better than this. When we make use of decision procedures on trees in these notes, we will be able to create the automaton T_φ directly, without going through MSO. This will allow us to get more decision procedures which are "elementary" (bounded by a fixed tower of exponentials).

There are a number of variations of Theorem 2. In particular, we will also consider *ordered trees*, where the children of a node are linear ordered by a *sibling relation*. A variant of Rabin's theorem holds for ordered ω-trees.

The *outdegree* or *rank* of a node is its number of children. If we fix the maximal rank r, then ordered trees of rank r can be considered as labelled trees, where the label of a node includes its sibling order within its parent. When we are dealing with trees of fixed rank, we often assume they are ordered (thus saying "binary trees", rather than "ordered binary trees").

3.1 Tree automata

A key tool in this will be automata over trees. These are restricted computing devices that define functions from trees to true or false.

One of the simplest kinds of tree automaton is a *nondeterministic automaton over finite complete binary trees*, or $\mathsf{NTA}_{\mathsf{FinBinary}}$. In this subsection, we will always be dealing with finite complete binary trees, so we abbreviate these to "trees" below.

Let Σ be $\{A_1, \ldots, A_n\}$. An $\mathsf{NTA}_{\mathsf{FinBinary}}$ $\mathcal{A}$ is specified by $(Q, \Sigma, I, F, \delta)$, where

- Q is a finite set of states
- Σ is a finite label set
- $\delta \subseteq Q^2 \times \Sigma \times Q$ is the *transition relation*
- I is a subset of $\Sigma \times Q$, the *initial leaf conditions*
- $F \subseteq Q$ are the *accepting (root) conditions.*

Informally, this describes a machine that processes a tree by assigning states to it. The set F restricts assignments of states to the root, while I restricts assignments of states to the leaves, based on their labels. The transition relation gives a "local restriction" on the state assignment: restricting the state assigned to the parent, the label of the parent, and the state assigned to each of its children. By referring to the leaf restriction I as "initial" and the root restriction F as "accepting", we give the impression of a machine assigning first to the leaves and then proceeding "bottom-up", arriving at a state for the root. But the computation could be equally thought of as top-down or as assigning states nondeterministically to all nodes in parallel.

Formally, an *accepting run* of the automaton on a (finite complete binary tree) t is an assignment of each node of the tree to a state $q \in Q$ such that:

- each leaf with label τ is assigned q such that $(\tau, q) \in I$;
- if a non-leaf node with label τ is assigned to state q, and the children of the node are assigned to q_1, q_2, then

$$((q_1, q_2), \tau, q) \in \delta;$$

- the root node is assigned some $q \in F$.

A tree t is *accepted* by an automaton $\mathcal{A}$ if there is an accepting run.

The *language* $L(\mathcal{A})$ of an automaton $\mathcal{A}$ is the set of trees that are accepted by $\mathcal{A}$.

Notice that in defining a tree automaton, we fixed a set of labels Σ, and dealt with trees in which each node has exactly one label from Σ. We will often apply this in the case where we have a finite set of unary predicates $P_1 \dots P_k$, and deal with trees in which each node can be labelled with a subset of the unary predicates. We can model this in the setting above by setting Σ to be the powerset of $P_1 \dots P_k$: then each node has one label, and it makes sense to run a Σ tree automaton over it.

Example 12. *Consider a tree automaton over predicates A, B with states q_A, q_B, with q_A the only accepting state. The initial labelled leaf states are $I = \{(\{A\}, q_A), (\{B\}, q_B)\}$ and the transition relation is $\delta = \{(q_A, q_B), \{A\}, q_A)\}$*

The automaton accepts exactly those trees consisting of a leftward chain of nodes labelled with $\{A\}$, with each node but the last having a right child labelled with $\{B\}$. Figure 12 shows an example of an accepted tree.

The basic computation we need on an automaton is to decide whether it accepts *some* tree. In the case of nondeterministic automata over finite trees, this is done via a very simple reachability construction [Thatcher and Wright, 1968]:

Proposition 2. *There is a polynomial time algorithm that takes as input an* $\mathsf{NTA}_{\mathsf{FinBinary}}$ $\mathcal{A}$ *and outputs true exactly when $L(\mathcal{A}) \neq \emptyset$.*

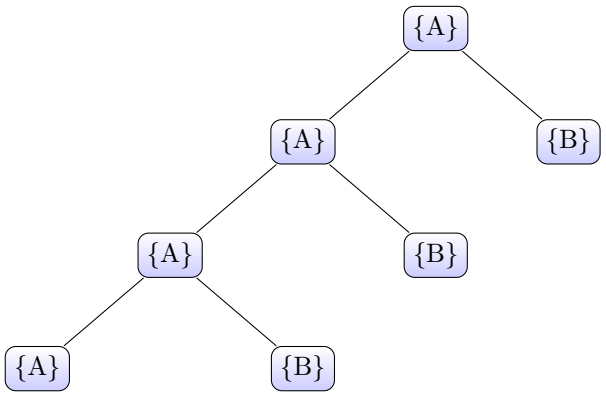

Figure 2: A tree accepted by the example automaton

Proof. We inductively determine the set ReachState of *reachable states*: the states q in Q such that there is a tree and a run of the automaton as above, but with the root labelled by q. ReachState is initialized with each q with $(\tau, q) \in I$. In the inductive step, we loop over each transition $((q_1 \dots q_r), \tau, q) \in \delta$, and if each $q_1 \dots q_r$ is in ReachState then q is added. This process is monotone and will thus terminate in a number of steps that is at most the size of Q. At the end of it we check if F overlaps with ReachState. □

Another attractive computational property of tree automata are their *closure properties*.

Proposition 3. *Given tree automata A_1 and A_2, there is a polynomial time algorithm that produces an automaton A which accepts a tree exactly when it is accepted by both A_1 and A_2. That is $L(A) = L(A_1) \cap L(A_2)$.*

Similarly, we can efficiently form an automaton that accepts $L(A_1) \cup L(A_2)$.

Given automaton A_1, we can effectively construct an automaton A that accepts all the trees that are not *accepted by A_1: the complement of $L(A_1)$. This last operation can be done in exponential time, but not in polynomial time.*

We have now discussed automata over complete finite binary trees. There is a simple extension to general finite binary trees, and also an obvious extension to finite trees of any outdegree. Since the definitions are similar but a bit more verbose, we omit them here.

In the text we will also make use of automata over *infinite* trees. The precise notions will be introduced later. The main thing the reader has to understand at this point is that the basic results listed above, like decidability of non-emptiness, will still hold for automata over infinite trees, albeit with higher complexity.

3.2 Bibliographic remarks and suggestions for further reading

A good introduction to automata on infinite trees is [Grädel et al., 2002]. The reader without familiarity with automata may want to read about finite automata on words first (e.g. see [Sipser, 1996]). The non-elementary complexity of MSO on trees was first proven in [Stockmeyer and Meyer, 1973].

4 Brief Review of Complexity

4.1 Brief Complexity Background

The text does not require extensive background in complexity theory. The main focus is to show that certain computational problems — e.g. satisfiability for formulas in some logic – are decidable. And to demonstrate decidability it suffices to exhibiting an algorithm for solving the problem that could be programmed on a computing device; these demonstrations will not be completely formal.

But in some cases we will need to argue there is a decision procedure with a certain running time. For example, for some of our logics, the decision procedure runs in "doubly-exponential time" or 2EXPTIME. The reader can think of these informally as well – there is some program in a standard programming language that solves the problem in time doubly exponential in its input. But we give a brief overview of some of the formal background here, along with some pointers to more detail.

A *Deterministic Turing Machine* over binary alphabet consists of:

- a finite set of states Q,
- an initial state $q_0 \in Q$
- a set of accepting states $A \subseteq Q$ and rejecting states $R \subseteq Q$
- a transition function $\delta : 2 \times Q \to (Q \times \mathsf{Dir} \times 2)$ where $\mathsf{Dir} = \{\mathsf{Left}, \mathsf{Right}\}$.

A *configuration* of a Deterministic Turing Machine consists of a finite binary sequence $s_1 \dots s_n$ representing the tape content, a state $q \in Q$, and a head position $h_i \leq n$. A *run* of a Turing Machine on an input string x_0 is a sequence of configurations $c_1 \dots$ with several properties described below.

- The state associated with configuration c_1 is q_0, the head position is 1 and the tape content of c_1 is x_0
- For each index i of the sequence c_i, c_{i+1} satisfies the transition function, in the following sense:

  Let h_i be the head position of c_i, $\mathsf{HeadVal}_i$ the value at index h_i in the tape content, and q_i the state in c_i. Let h_{i+1}, q_{i+1} be defined similarly for c_{i+1}, and let $\mathsf{HeadVal}_{i+1}$ be the value underneath h_i in the tape of c_{i+1}. The tape values of c_{i+1} are the same as those of c_i for cell positions other than h_i. For the cell position h_i, if $\delta(\mathsf{HeadVal}_i, q_i) = (q_{i+1}, \mathsf{Dir}, \mathsf{HeadVal}_{i+1})$, then the cell position at h_i is $\mathsf{HeadVal}_{i+1}$.

  The head position h_{i+1} must satisfy the following:

  If $\delta(\mathsf{HeadVal}_i, q_i) = (q_{i+1}, \mathsf{Right}, \mathsf{HeadVal}_{i+1})$ then $h_{i+1} = hi + 1$.
  If $\delta(\mathsf{HeadVal}_i, q_i) = (q_{i+1}, \mathsf{Left}, \mathsf{HeadVal}_{i+1})$ and $h_{i+1} = hi - 1$

An *accepting run* is one which reaches a state in A, while a run that reaches a rejecting state is a *rejecting run.*

The *time* taken by such a run is simply the number of configurations in the sequence, while the *space* is the maximal number of cells in any configuration.

A decision problem is just a function from strings over some alphabet to $\{\mathsf{True}, \mathsf{False}\}$. For a function $F(n)$ a decision problem P is said to be in $\mathsf{DTIME}(F)$ if there is a Deterministic Turing Machine that gives the correct answer to P, where for each input w, the time of the run is bounded by $F(|w|)$. The notion of a problem being in $\mathsf{DSPACE}(F)$ is defined similarly. The main example of this we will refer to is the class of problems that can be decided with *polynomial space*: PSPACE, for short.

A problem is in *polynomial time* (PTIME) if it is in $\mathsf{DTIME}(F)$ for some polynomial F. It is in *exponential time* (EXPTIME) if is in $\mathsf{DTIME}(2^F)$ for some polynomial F, and similarly for double-exponential time (2EXPTIME).

In this book we will sometimes deal with computational problems that have extremely high complexity. Let $\mathsf{EXPTOWER}(k, n)$ be defined by induction: $\mathsf{EXPTOWER}(1, n) := n$ $\mathsf{EXPTOWER}(k+1, n) = 2^{\mathsf{EXPTOWER}(k,n)}$. A problem is *elementary* if there is a k such that it is in $\mathsf{DTIME}(\mathsf{EXPTOWER}(k, n))$. That is, its complexity is bounded by a tower of k exponentials. A problem is *non-elementary* if it is non-elementary. Thus non-elementary problems are very hard indeed.

A Non-Deterministic Turing Machine is defined similarly to a deterministic one, but instead of the transition function we have a *transition relation*: $\delta \subseteq 2 \times Q \times Q \times \mathsf{Dir} \times 2)$. A run is defined as with a deterministic machine, but now we will replace statements like

$$\delta(\mathsf{HeadVal}_i, q_i) = (q_{i+1}, \mathsf{Right}, \mathsf{HeadVal}_{i+1})$$

with a statement:

$$(\mathsf{HeadVal}_i, q_i, q_{i+1}, \mathsf{Right}, \mathsf{HeadVal}_{i+1}) \in \delta$$

Given an input string, a Deterministic Turing Machine has exactly one maximal run, while a non-deterministic machine can have no runs or many runs.

A Non-Deterministic Turing Machine M *decides* a problem P if:

- every run of M is accepting or rejecting,
- on every x with $P(x) = \mathsf{True}$, there is some accepting run of M on x,
- for every x with $P(x) = \mathsf{False}$ every run of M on x is non-accepting.

For a function $F(n)$ a decision problem P is said to be in $\mathsf{NTIME}(F)$ if there is a Non-Deterministic machine M that decides P, where on every input x of size n, each run is of size at most $F(n)$.

A problem is in non-deterministic polynomial time (NP) if it is in $\mathsf{NTIME}(F)$ for some polynomial F. A fundamental problem related to logic that is in NP is the *propositional satisfiability problem*: given a formula φ built up from 0-ary relations (propositions) using the connectives $\wedge, \vee, \neg$, determine if φ is satisfiable. The NP algorithm just guesses an assignment of $\mathsf{True}, \mathsf{False}$ to each propositions and then checks whether it makes the sentence true.

In some places, we will mention that a problem is not just in a class C, but is "complete" for the class C. For example, we will say that the modal logic satisfiability problem is complete for the class PSPACE. Informally, this means that the problem really requires the full power of PSPACE: we cannot do better. We now formalize this.

A polynomial time many-one reduction from problem P to problem Q is a function R from inputs to problem P to inputs of problem Q such that:

For each x, $P(x) = \mathsf{True}$ if and only if $Q(R(x)) = \mathsf{True}$.

A problem P_0 is *hard* for a complexity class C (PSPACE, 2EXPTIME, etc.) under polynomial time many-one reductions if for any problem P in C, there is a polynomial time many-one reduction from P to P_0. A problem P_0 is complete for a complexity class C if it is in C and complete for C.

Thus we say a problem is NP-complete if it is complete for the class NP.

Above we have talked about problems whose inputs are binary strings. One can deal with problems that have as input more complex structures, by coding them as strings. For example, a first-order formula φ can be coded easily as a binary string. First we would pick some string to represent each of the symbols $\forall, \exists, \wedge, \vee, \neg$ and then representing each variable x by a unique binary string differing from each string representing a symbol. The binary representation of a formula would then just be the concatenation of the representations of its symbols, with a special separator string in between each.

Theorem 3. *Propositional satisfiability is* NP*-complete*

A number can be represented as a string in two ways: the *binary* encoding, where a string of n 1's represents 2^n, and the *unary* where a string of n 1's represents n.

4.2 Bibliographic remarks and further reading

The material in this section can be found in many textbooks, such as [Papadimitriou, 1994]. For a focus on NP-completeness and its connection to problems like propositional satisfiability, check [Garey and Johnson, 1979]

5 Modal Logic

We begin with the decidability of basic modal logic, which will set the template for our decidability arguments throughout these notes.

5.1 Modal logic basics

Basic modal logic is a logic restricted to vocabularies consisting of finitely many unary relations $U_1 \dots U_n$ and a single binary relation R. A structure for this kind of vocabulary is a *Kripke Structure*.

Every formula will have at most one free variable, which we will denote as x. Thus we can ask whether a formula is true in a given Kripke Structure and a given node of the structure.

The left side of Figure 3 shows a Kripke Structure with 4 nodes. There are predicates Orange, Green and Blue, with Green being true of two nodes, Orange and Blue holding of one node each. The binary relation between nodes is shown via the red arrows. If we want to determine whether a formula is true in this structure, we need to pick a distinguished node, such as the node highlighted with red border on the left.

The basic formulas are True, False, and atomic unary formulas $U_i(x)$.

Formulas are defined inductively by:

- if φ_1 and φ_2 are formulas, then $\varphi_1 \wedge \varphi_2$, $\neg\varphi_1$, and $\varphi_1 \vee \varphi_2$ are formulas.
- if φ is a formula then $\exists y\ R(x, y) \wedge \varphi(x := y)$ is a formula.
- if φ is a formula then $\forall y\ R(x, y) \rightarrow \varphi(x := y)$ is a formula.

where $\varphi(x := y)$ is the formula obtained from φ by substituting y for x.

In this work, we consider modal logic as a fragment of first-order logic. This is not the standard syntax for modal logic. Modal logic formulas always have one free variable, so it is possible to use a succinct notation in which the variable is omitted. The usual notation is as follows:

- A formula $U_i(x)$ is usually just written as U_i.
- if φ is a formula then $\Diamond\varphi$ is a formula (with the meaning $\exists y\ R(x, y) \wedge \varphi(x := y)$).
- if φ is a formula then $\Box\varphi$ is a formula (with the meaning $\forall y\ R(x, y) \rightarrow \varphi(x := y)$)

Example 13. $\mathsf{Blue} \wedge \Diamond\mathsf{Orange}$ *is a modal formula that holds on a node that is labeled with* Blue *and which links to a node labelled with* Orange. *The highlighted node in the Kripke Structure in the left side of Figure 3 satisfies this formula.*

$\Box\mathsf{False}$ *is a formula that holds of nodes that have no R-successors. There are no nodes in the Kripke Structure on the left side of Figure 3 that satisfy this formula.*

5.2 Decidability via the tree model property

A formula $\varphi(x)$ is satisfiable if the sentence $\exists x\ \varphi(x)$ is satisfiable in the usual sense.

Our aim is to show:

Theorem 4. *Satisfiability of a modal logic formula $\varphi(x)$ is decidable.*

In the case of modal logic, the notion that every satisfiable formula has a "tree-shaped" model has a particularly straightforward formulation.

We say that a model of a modal logic signature is a *tree structure* if the relation R forms a tree – an acyclic connected graph such that the following hold:

- for every element v' there is at most one v such that $R(v, v')$
- there is exactly one element v' for which there is no such v as above; and we refer to such an element as the root element of the structure.

Since modal logic is a fragment of MSO, Theorem 4 follows easily from Theorem 2 and:

Theorem 5. *If a formula $\varphi(x)$ of modal logic is satisfiable, then there is a tree structure M satisfying $\varphi(x)$.*

We prove this using an *unravelling* construction. The informal idea of the unravelling of a structure M at an element x_0 is that we make a copy x_0' of x_0, giving x_0' the same unary relations true at x_0; we then take every element c directly connected to x_0 (the "children" of x_0) and make a fresh copy c' of c, connecting them to x_0'; we then continue recursively on all these $c's$. Formally, the elements of the unravelling of M at x_0, denoted $\mathsf{ModalUnravel}(M, x_0)$ are finite paths through M starting from x_0, where a path is just (as usual) a sequence s of elements $x_0 \dots x_n$ in M where for $i \leq n-1$ $R(x_i, x_{i+1})$ holds in M. The relation $R(s, s')$ holds for two such paths if s' is obtained from path s by adding one additional element. Clearly this is a tree. A unary relation U holds of a sequence $s = x_0 \dots x_n$ in $\mathsf{ModalUnravel}(M, x_0)$ exactly when $U(x_n)$ holds in M.

Figure 3 shows a simple Kripke structure and the first few levels of its unravelling.

Notice that each element of $x \in M$ is associated with certain copies in the unravelling; namely, all paths s that end in x, which are labelled in $\mathsf{ModalUnravel}(M, x_0)$ exactly as x is labelled in M.

We want to show that the modal formulas that x_0 satisfies in M are the same as those satisfied by x_0 in $\mathsf{ModalUnravel}(M, x_0)$. As a step towards this we define an equivalence relation between Kripke Structures.

Given Kripke structures M, M' and elements $x_0 \in M, x_0' \in M'$, a *bisimulation relation* between x_0 and x_0' is a binary relation $B \subseteq M \times M'$ such that $B(x_0, x_0')$ and whenever $B(x, x')$ holds

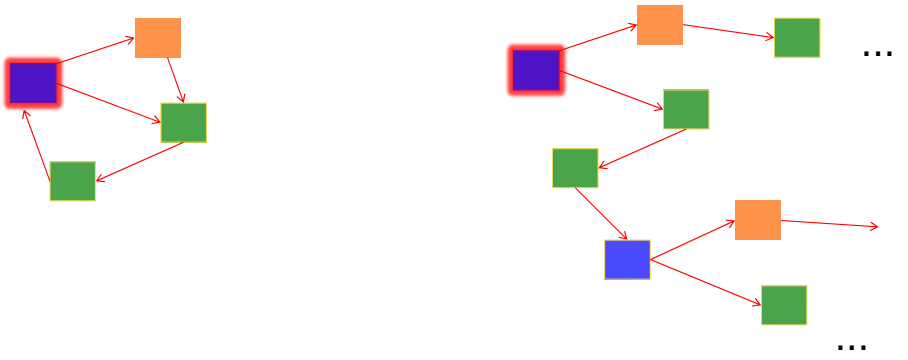

Figure 3: A structure and its unravelling

- unary predicates are preserved: $M \models U_i(x) \leftrightarrow M' \models U_i(x')$
- (Forth) for each $x_1 \in M$ with $M \models R(x, x_1)$ there is $x_1' \in M'$ with $B(x_1, x_1')$ such that $M' \models R(x', x_1')$.
- (Back) for each $x_1' \in M'$ with $M \models R(x', x_1')$ there is $x_1 \in M$ with $B(x_1, x_1')$ such that $M \models R(x, x_1)$.

(M, x) and (M', x') are *bisimilar* if there is a bisimulation relation between them.

One way to think of bisimulation that will be useful later on is as a game between two players, Spoiler (who wants to show that the structures are not bisimilar) and Duplicator (who wants to show that they are bisimilar). At any point in the game the play consists of a pair (x, x') with $x \in M$ and $x' \in M'$. If x and x' do not satisfy the same atomic formulas, then Spoiler immediately wins. Otherwise, a round of the game proceeds by Spoiler choosing one structure, say M, and then picking $x_1 \in M$ with $M \models R(x, x_1)$. Duplicator must respond in the other structure with x_1' such that $M' \models R(x', x_1')$, and then the game continues at position (x_1, x_1'). If Spoiler gets stuck or Duplicator is able to continue playing forever, then Duplicator wins.

It is easy to see that (M, x) and (M', x') are bisimilar exactly when Duplicator can win the game starting at position (x, x'). Formally, a *strategy* for the Duplicator is a function λ that inputs a position and a choice of Spoiler, and returns a valid response for Duplicator. A play of the game abides by the strategy λ if Duplicator always responds using λ. Given a position (M, x), (M', x) a strategy λ is winning starting at the position if every play of the game starting at this position and abiding by λ is winning for Duplicator. Thus the formal statement of the observation above is:

(M, x) and (M', x') are bisimilar exactly when Duplicator has a winning strategy for the game starting at position (x, x').

Consider the correspondence between elements in a structure and the corresponding elements in the unravelling. The correspondence is depicted in Fig-

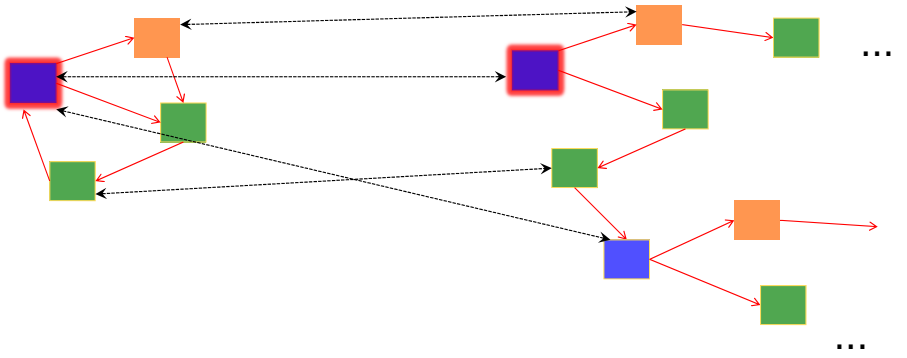

Figure 4: Relation between elements in a structure and elements in the unravelling

ure 4.

It is easy to check that this correspondence is a bisimulation:

Proposition 4. *The mapping B relating any $x \in M$ to each corresponding copy in the unravelling is a bisimulation relation.*

Further, we can show that any bisimulation preserves modal logic formulas:

Proposition 5. *If (M, x) and (M', x') are bisimilar, then the modal formulas satisfied by x in M are the same as those satisfied by x' in M'.*

Proof. The proof will have some similarity to the proof of Proposition 1, which also showed that whenever there was a winning strategy in a certain game, a class of logical formulas was preserved.

In this case, we need to show that:

if M, c is bisimilar to M', c' and $M, c \models \varphi$, then $M', c' \models \varphi$.

We can proceed by induction on the way φ is built up from the grammar of modal logic.

The base case is where φ is just a unary symbol $U(x)$. But the first property of a bisimulation is that the unary symbols are preserved, so this case follows.

The first inductive cases are for connectives, and these will be simple, proceeding exactly as in Proposition 1. Suppose φ is of the form $\varphi_1 \wedge \varphi_2$, $M, c \models \varphi$, and M, C is bisimilar to M', c'. Then $M, c \models \varphi_1$ and $M, c \models \varphi_2$. The induction hypothesis applies to φ_1 and φ_2, and thus $M', c' \models \varphi_1$ and $M', c' \models \varphi_2$. Thus $M', c' \models \varphi_1 \wedge \varphi_2$ as required. The cases for $\vee, \neg$ are similar.

Consider the case of φ built up from the existential or "diamond" quantification.

$$\varphi = \exists y \; R(x, y) \wedge \varphi'$$

If $M, c \models \varphi$, then there is d such that $R(c, d)$ holds in M and $M, d \models \varphi'$.

By the (Forth) property of bisimulation there is d' such that $R(c', d')$ and M, d is bisimilar to M', d'. By induction $(M', d') \models \varphi'$, since φ' is of lower complexity. Thus

$$M', c' \models \exists y\ R(x, y) \wedge \varphi'$$

as required.

The case of universal quantification is similar, but using the (Back) property. □

Theorem 5 follows immediately from the previous propositions.

We are now ready to prove Theorem 4. Given a modal logic formula φ, from Theorem 5 we know that it suffices to check whether φ is satisfied in a tree. This in turn is decidable via Theorem 2.

5.3 Automata for modal logic

Our next goal will be to get a better decision procedure using automata. For the case of modal logic, the kind of automata we need are *nondeterministic automata over finite ordered trees of bounded outdegree*, or $\mathsf{NRTA}_{\mathsf{FinRanked}}$, which generalize the binary tree case explained earlier.

We fix a maximal outdegree (or *rank*) of the trees r, and a set of node predicates $A_1 \ldots A_n$ for such trees. Let Σ be $\{A_1, \ldots, A_n\}$. An $\mathsf{NRTA}_{\mathsf{FinRanked}}$ $\mathcal{A}$ is specified by $(Q, \Sigma, I, F, \delta)$, where

- Q is a finite set of states
- Σ is a finite set of labels
- $\delta \subseteq \bigcup_{1 \leq i \leq r} Q^i \times \mathcal{P}(\Sigma) \times Q$ is the *transition relation*
- I is a subset of $\mathcal{P}(\Sigma) \times Q$, the *initial labelled leaf states*
- $F \subseteq Q$ are the *accepting states.*

As with binary trees, this describes a machine that processes a tree by assigning states to it. The set F restricts assignments of states assigned to the root, while I restricts assignments of states to the leaves, and the transition relation restricts the state assigned to internal nodes.

An accepting run of the automaton on a tree t is an assignment of each node of the tree to a state $q \in Q$ such that:

- each leaf with label τ is assigned q such that $(\tau, q) \in I$;
- if a non-leaf node with label τ is assigned to state q, and the children of the node are assigned to $q_1 \ldots q_r$, then

$$((q_1 \ldots q_r), \tau, q) \in \delta;$$

- the root node is assigned some $q \in F$.

A tree t is accepted by an automaton $\mathcal{A}$ if there is an accepting run, and the language $L(\mathcal{A})$ of an automaton $\mathcal{A}$ is the set of trees that are accepted by $\mathcal{A}$.

As with binary trees, non-emptiness can be tested by a very simple reachability construction [Thatcher and Wright, 1968]:

Proposition 6. *There is a polynomial time algorithm that takes as input an* $\mathsf{NRTA}_{\mathsf{FinRanked}}$ $\mathcal{A}$ *and outputs true exactly when* $L(\mathcal{A}) \neq \emptyset$.

5.4 Optimized decidability via translation to an automaton

Given a modal formula φ, we now construct a tree automaton $\mathcal{A}_\varphi$ that runs over a tree-shaped Kripke structure and checks that φ holds. We define an automaton $\mathcal{A}_\varphi$ that expects a *ranked, ordered labelled tree*, and thus one can distinguish one child from another. $\mathcal{A}_\varphi$ implementing φ will mean that for any ordered tree $t_<$, φ is satisfied in the underlying Kripke structure of $t_<$ if and only if $\mathcal{A}_\varphi$ accepts $t_<$. Note that since φ does not make use of the order, $\mathcal{A}_\varphi$ will in fact give the same result on any two ordered trees that have the same underlying labelled tree structure.

Because the tree structure is finite with degree d, we use an $\mathsf{NRTA}_{\mathsf{FinRanked}}$. The use of finite tree automata requires us to deal with finite trees. We thus need a variant of Theorem 5. The *modal depth* of a formula is the number of nestings of modalities.

Theorem 6. *Consider a formula* $\varphi(x)$ *of modal logic of modal depth* d. *If* φ *is satisfiable, then there is a model* M *which is a tree structure of depth* d *and where the maximal number of children of each node (i.e. the* degree*) is bounded by a number polynomial in the size of* φ.

For the depth bound, we can truncate the unravelling at depth d, and see that the formulas of modal depth d satisfied at the root are unaffected.

For the degree bound, we need only a bit more work. First, we should normalize the formula. A formula is in *negation normal form* (NNF) if negation only occurs on atoms. A formula can be converted into NNF in polynomial time, just by using the De Morgan rules, repeatedly replacing $\neg\forall x\ \varphi$ by $\exists x\ \neg\varphi$ and $\neg\exists x\ \varphi$ by $\forall x\ \neg\varphi$. Thus we will assume our formula is in NNF. We can now trim the number of children of each node starting at the parents of leaf nodes. The idea is that a node that is i steps from the leaf level needs only one representative of satisfiability for existential subformulas of modal depth at most i. More precisely, suppose we have a Kripke structure M in which node v satisfies φ. φ is a Boolean combination of existential (or "diamond") subformulas $\varphi_1 \ldots \varphi_n$. For each φ_j satisfied by v, we can choose a child of v, v_j that witnesses this. Now eliminate from M all the other children of v along with their subtrees. We claim that all the subformulas of φ satisfied by v before performing this transformation are still true afterwards. For atomic formulas and their negations, this is true because the labels of v were not changed. The existential formulas satisfied by v are not impacted, since there is already a

witness. The universal formulas satisfied by v are also not impacted, since we are now quantifying over fewer nodes. The induction step for $\wedge$ and $\vee$ is straightforward, and there is no induction step for negation, since the formula was in NNF. Repeating this "trimming operation" gives a degree bound that is polynomial in the formula.

Now that we know that it is enough to deal with finite trees of some fixed branching degree g, we can begin to translate formulas to automata that run over (ordered versions of) such trees.

We again assume that our modal formula φ is in NNF. Let S be a collection of subformulas of φ with some fresh c substituted for any free variables. We say S is *consistent* if:

- it does not contain both $U(c)$ and $\neg U(c)$ for some atomic formula $U(c)$;
- whenever S contains $\varphi_1 \vee \varphi_2$ it contains φ_1 or φ_2 (or both);
- whenever S contains $\varphi_1 \wedge \varphi_2$ it contains both φ_1 and φ_2.

The automaton $\mathcal{A}_\varphi$ for our modal logic formula φ will have states for every consistent subset S of the subformulas of φ (with c substituted for free variables), together with a special non-accepting sink state. The automaton will satisfy the property that if there is a run that reaches state S at some node n whose subtree has height h, then the formulas in S of modal depth at most h will hold at n within t.

The accepting states are all consistent sets containing φ. The initial labelled leaf states are all pairs (τ, S) where S is a consistent set that "does not contradict the predicates in τ": that is, $\{U(c) : U \in \tau\} \cup \{\neg U(c) : U \in \Sigma \setminus \tau\} \cup S$ is consistent.

The transition relation δ_φ describes the possible g-tuples $(S_1 \dots S_g)$ of states for the children of a node with state $S = \{\psi_1 \dots \psi_n\}$ and label τ.

If $\{U(c) : U \in \tau\} \cup \{\neg U(c) : U \in \Sigma \setminus \tau\} \cup S$ is inconsistent, then $S_1, \dots, S_g$ must be the special non-accepting sink state. Otherwise, the transition must satisfy:

- if S contains φ_0 of the form $\forall y\ R(c, y) \to \varphi_1$ then each S_i should contain $\varphi_1(c)$.
- if S contains φ_0 of the form $\exists y\ R(c, y) \wedge \varphi_1$ then some S_i should contain $\varphi_1(c)$.

We can verify that the states of the automaton correctly represent the truth of formulas:

Lemma 1. *For any tree t of height at most h, the automaton $\mathcal{A}_\varphi$ has a valid run on t where the root is associated with state S if and only if all formulas in S of modal depth at most h holds at the root of t.*

The lemma is proven by induction on h. The base case is where the tree consists only of the root, and we are dealing with formulas of modal depth 0,

which are Boolean combinations of atomic formulas. This is true by definition of the initial state. The inductive step for modalities follows easily from the transition function and the semantics of modalities.

To see that this lemma implies that the automaton has the properties we desire, let h be the modal depth of φ. Applying the lemma to h we see that the automaton reaches an accepting state on a tree of height at most h if and only if the root of the tree satisfies φ. The lemma is proven by induction

Recall that the language $L(\mathcal{A}_\varphi)$ of automaton $\mathcal{A}_\varphi$ consists of all trees that are accepted by $\mathcal{A}_\varphi$ starting from the root. By the previous lemma, this corresponds to exactly the trees where φ holds at the root. This means we have reduced satisfiability testing to a non-emptiness test for the language.

Since the size of $\mathcal{A}_\varphi$ is exponential in the size of the original modal formula φ, this gives an EXPTIME upper bound to satisfiability testing for modal logic. In fact, it is possible to do better, staying within PSPACE, as we show next.

5.5 Tight complexity bounds on modal logic satisfiability

The translation from a modal formula φ to an automaton A_φ we proposed here for deciding modal logic is still not optimal. The nondeterministic automaton has exponentially many states, since it has states for collections of subformulas of the modal formula. Hence we cannot construct the automaton explicitly without taking exponential time.

We now deal with this problem. We use two properties of the automaton:

- States and alphabet symbols can be represented with polynomially many bits. Further, one can verify that a transition is in the automaton in polynomial time, and similarly can verify that a state is accepting or is an initial state.
- The automaton can accept trees only of depth bounded by d, where d is the modal depth.

We now argue that there is a PSPACE decision procedure. We give a nondeterministic PSPACE algorithm, which suffices by Savitch's theorem [Savitch, 1970]. At any point the algorithm has a state S of the automaton and a depth d, with the goal of checking for satisfiability of the formulas of S in a node with a label τ in a tree of depth d, with a stack of subgoals for the parent nodes. The algorithm guesses an outdegree $k \leq |\varphi|$, states $S_1 \dots S_g$ and labels $\tau_1 \dots \tau_g$ for each of the children. It then verifies the transition from parent to child is valid, and then descends into each child in turn to verify satisfiability of S_i at a tree of depth $d-1$ with label τ_i, with the stack of subgoals modified accordingly.

It turns out that PSPACE is the best possible bound, since the problem is PSPACE-hard [Ladner, 1977].

5.6 Bibliographic remarks

Early results on the complexity of modal logic can be found in [Ladner, 1977]. A discussion of the complexity of modal logics with restrictions on the kinds

of Kripke structures considered can be found in [Hemaspaandra and Schnoor, 2008]. Modal logic is an extremely rich topic, which can be approached both proof-theoretically or model-theoretically, with many variations. See [Blackburn et al., 2001] for one introduction.

The outline of our decidability discussion follows the exposition in Section 4 of [Vardi, 1997].

6 The Guarded Fragment

Basic modal logic restricts the signature to have a single binary predicate. We now present an extension to an arbitrary relational signature, the *guarded fragment* GF.

6.1 Guarded Fragment basics

The atomic formulas of GF are True, False, any relational atom, and equality atoms.

Formulas are defined inductively by:

- If φ_1 and φ_2 are GF formulas, then $\varphi_1 \wedge \varphi_2$, $\neg\varphi_1$, and $\varphi_1 \vee \varphi_2$ are formulas.
- If φ is a formula of GF then $\exists \vec{y}\ R(\vec{c}, \vec{x}, \vec{y}) \wedge \varphi$ is a formula of GF where the free variables of φ are required to be contained in $\vec{x} \cup \vec{y}$. We call $R(\vec{c}, \vec{x}, \vec{y})$ the "guard atom" or "guard" of the quantification.
- If φ is a formula of GF then $\forall \vec{y}\ R(\vec{c}, \vec{x}, \vec{y}) \rightarrow \varphi$ is a formula of GF, where as above all free variables of φ must be used in the "guard atom" $R(\vec{x}, \vec{y})$.

Above we have written $R(\vec{c}, \vec{x}, \vec{y})$ to emphasize that R can contain constants $\vec{c}$ in addition to the quantified variables $\vec{y}$ and free variables $\vec{x}$. But in the future we will sometimes drop the constants $\vec{c}$ in writing guards for brevity.

It is easy to see that all modal logic formulas are actually GF formulas. However, GF allows us to go beyond modal logic in several respects:

Example 14. *The most obvious feature of* GF *compared to modal logic is that it allows us to talk about arbitrary arity relations.*

Suppose we have a ternary relation $S(x, y, z)$. Then

$$\forall xyz\ S(x, y, z) \rightarrow \exists uv\ S(z, u, v)$$

is a GF *sentence.*

But even in a signature with binary relations, GF *allows us to do some things that basic modal logic cannot do. It allows us to talk about multiple binary relations:*

$$\forall xy\ R(x, y) \rightarrow S(x, y)$$

It allows us to universally quantify over elements of a unary relation:

$$\forall x\ U(x) \rightarrow \exists y\ R(x, y) \wedge V(y)$$

It allows us to make assertions about self-edges:

$$\forall x\ U(x) \rightarrow \exists y\ R(x, y) \wedge R(y, y)$$

And it allows us to talk about a binary relation in both directions:

$$\forall x\ U(x) \rightarrow \exists y[R(x, y) \wedge (\exists z\ R(z, y) \wedge V(z))]$$

An important parameter of a GF formula φ is the *width*, the maximal arity of a relation in φ. We write width(φ) for the width of φ.

6.2 Decidability of GF via the tree model property

Our aim is to show:

Theorem 7. *Satisfiability of a* GF *sentence is decidable.*

We will do this by proving the tree-model property, applying a variation of the unravelling construction used for modal logic. Prior to defining the unravelling itself, we will have to define what "tree-like" means, since it will no longer mean that the model is literally a tree (as it did with modal logic).

For a number k and relational signature σ, the k *tree-code signature for* σ consists of a binary relation E, unary relation symbols F for every fact F that uses relations in σ and arguments in $\{1 \dots k\}$, and also symbols $\mathsf{Eq}(i,j)$ for $i,j \leq k$ and Card_i for $i \leq k$. Informally, an element of this code structure represents a tuple of at most k elements in a structure over the signature σ; the unary relations F on codes tell which relations hold of the corresponding tuple, while for two code elements connected by E, if $\mathsf{Eq}(i,j)$ holds at v then the i-th element in v is equal to the j-th element in the parent of v. Card_i asserts that the number of elements in the tuple is exactly i. Given elements v, v' in a structure for this signature, along with $i,j \leq k$ we say that (v,i) and (v',j) are *directly equivalent* in the structure if $E(v',v) \wedge \mathsf{Eq}(i,j)(v)$ hold in the structure. We call the transitive and reflexive closure of this relation *annotated node equivalence* in the structure. Omitting the dependence on the underlying structure, we let $[v,i]$ denote the equivalence class of the pair (v,i).

Example 15. *Figure 5 shows a tree code in the coding signature for $k = 3$, over a base alphabet consisting of three unary predicates (the colors orange, green, and pink in the picture along with three binary relations (the red, green, and blue) edges. L_1, L_2, and L_3 are three substructures over a domain of size 3 in this language. The root node is "labelled as representing a copy of L_1": more precisely, it is labelled with node predicates for each of the facts in L_1: e.g. $Orange(2)$ and $Red(3,2)$. The left child of the root is "labelled as representing a copy of L_3". In addition, it is labelled with the relation $\mathsf{Eq}(2,3)$, indicating that node 2 in the child structure is to be identified with node 3 in its parent. The right child of the root is similarly labelled as representing a copy of L_2, with element 2 identified with element 3 in its parent.*

The structure that the tree codes is shown on the right side of the figure. Note that the structure can be formed from taking copies of the structures L_1, L_2, L_3 and identifying nodes according to the additional Eq *labels.*

A structure for this alphabet $(V, E, \dots)$ is a *valid k tree code* if

- (V,E) is a tree
- for any i at most one of $\mathsf{Eq}(i,j)$ and at most one of $\mathsf{Eq}(j,i)$ can hold at a node
- exactly one of Card_m can hold at a node, and if Card_m holds then any coded facts must be over $\{1, \dots, m\}$ and we cannot have $\mathsf{Eq}(i,j)$ for $i > m$.

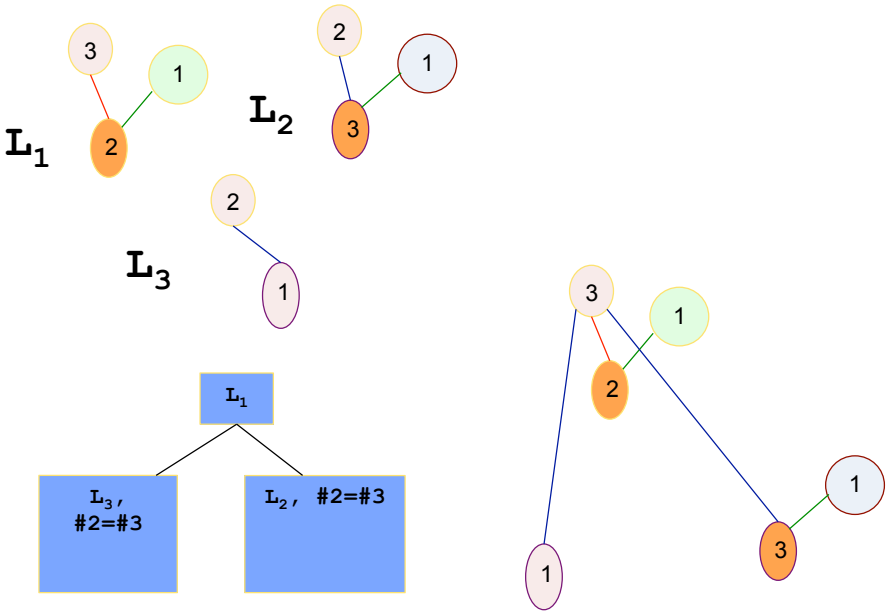

Figure 5: A tree and the structure that it codes

For a valid tree code t, the *decoding* decode(t) is the structure whose elements are annotated node equivalence classes and where $R([v, i_1], \ldots [v, i_n])$ holds exactly when v satisfies $R(i_1 \ldots i_n)$ in V.

A structure is *codable with width k* if it is the decoding of some valid k tree code. We can now make precise the tree-like model property for GF:

Alternative definition of tree-codable structure. Above we have defined k-tree codes, and explained what the decoding is of a code. That is, we have defined the structure M being coded from the tree that codes it. It is more standard to give a direct definition of what it means for a structure M to have a tree coding. A *tree decomposition* of a structure M is a tree (V, E) and a function λ assigning to each vertex $v \in V$ with a subset $\lambda(v)$ of elements in the domain of M, so that the following hold:

- For each fact $R(c_1 \ldots c_n) \in M$, there is a v such that $\lambda(v)$ includes each element of $c_1 \ldots c_n$.
- For each domain element $e \in M$, the set of nodes

  $$\{v \in V | e \in \lambda(v)\}$$

  is a *connected subset* of the tree. For any two vertices v_1, v_2 in the set, there is a path in the tree connecting them.

The *width* of a decomposition is the maximum size of $\lambda(v)$ over any element $v \in V$. The subsets of M of the form $\lambda(v)$ are called *bags* of the decomposition. It is easy to see that:

a model M is isomorphic to a decoding of a k-tree code exactly when it has a tree decomposition of width at most k.

From the tree decomposition we can define a code by labelling a tree node v according to the facts holding on $\lambda(v)$. Conversely, from a tree code for M we can form a tree decomposition of M by defining λ to associate a node in the tree with the set of elements it codes.

The tree-like model property for GF is now captured by the following result:

Theorem 8. *Any* GF *sentence φ that is satisfiable has a satisfying model which is codable with width k, where k is the width of φ.*

We now turn to the notion of bisimulation that will be used to show the tree-like model property. A tuple of elements $\vec{t}$ in a structure M is *guarded* if there is a single fact that contains every $t_i \in \vec{t}$. We similarly talking about a *guarded set* of a structure: a set whose elements co-occur in some fact.

Given arbitrary structures M, M' for the same signature and guarded tuples $\vec{x}_0 \in M, \vec{x}'_0 \in M'$ a *guarded bisimulation relation* between $(M, \vec{x}_0)$ and $(M', \vec{x}'_0)$ is a binary relation $B(\vec{x}, \vec{x}')$ between guarded tuples $\vec{x} \in M$ and $\vec{x}' \in M'$ such that whenever $B(\vec{x}, \vec{x}')$ holds:

- (Partial isomorphism) All predicates (including equalities) on the distinguished elements are preserved: $M \models S(x_{j_1} \dots x_{j_m}) \leftrightarrow M' \models S(x'_{j_1} \dots x'_{j_m})$.
- (Forth) For each guarded tuple $\vec{y} \in M$ there is guarded tuple $\vec{y}' \in M'$ with $B(\vec{y}, \vec{y}')$ such that $y_i = x_j$ implies $y'_i = x'_j$.
- (Back) For each guarded tuple $\vec{y}' \in M'$ there is a guarded tuple $\vec{y} \in M$ with $B(\vec{y}, \vec{y}')$ such that $y'_i = x'_j$ implies $y_i = x_j$.

$(M, \vec{x})$ and $(M', \vec{x}')$ are *guarded bisimilar* if there is a guarded bisimulation relation between them. We extend this notation to two models M and M' without any tuples: we say they are guarded bisimilar if $(M, \emptyset)$ is guarded bisimilar to $(M', \emptyset)$, where $\emptyset$ denotes the empty tuple.

As we did with modal logic, we can rephrase guarded bisimulation in terms of a game between two players Spoiler and Duplicator. A position of the game is a pair of guarded tuples $\vec{x}_0 \in M$ and $\vec{x}'_0 \in M'$. Spoiler can decide to pick one of the structures, say M, and replace $\vec{x}_0$ with a guarded $\vec{y}_0$ that may overlap with $\vec{x}_0$. Duplicator must then respond with $\vec{y}'_0$ that satisfies the same predicates as $\vec{y}_0$ and overlaps with $\vec{x}'_0$ whenever $\vec{y}_0$ overlaps with $\vec{x}_0$. Spoiler wins if Duplicator cannot make a play. An infinite sequence in which Spoiler never wins is a winning play for Duplicator.

Proposition 7. *Duplicator has a winning strategy in the guarded bisimulation game starting from initial position $\vec{x} \in M$ and $\vec{x}' \in M'$ if and only if $(M, \vec{x})$ and $(M', \vec{x}')$ are guarded bisimilar.*

We now argue that if $(M, \vec{x})$ and $(M', \vec{x}')$ are guarded bisimilar, then they agree on GF formulas.

Proposition 8. *If $(M, \vec{x})$ and $(M', \vec{x}')$ are guarded bisimilar, then the* GF *formulas satisfied by $\vec{x}$ in M are the same as those satisfied by $\vec{x}'$ in M'.*

Proof. Spelling out the statement as:

if $M, \vec{c}$ is guarded bisimilar to $M', \vec{c}'$ and $M, \vec{c} \models \varphi$, then $M', \vec{c}' \models \varphi$.

We can proceed by induction on the way φ is built up from the GF grammar.

The base case is φ atomic, and this holds using the partial isomorphism property. The connectives $\wedge, \vee, \neg$ are straightforward.

Consider the case of φ built up from existential quantification.

$$\varphi = \exists y_1 \dots y_m \; R(x_{j_1} \dots x_{j_n}, y_1 \dots y_m) \wedge \varphi'$$

By adding equalities into φ', we can assume that $x_{j_1} \dots x_{j_n}$ and $y_{d_1} \dots y$ are are sequences of variables, possibly with repetition. If $M, \vec{c} \models \varphi$ then there are $d_1 \dots d_m$ such that $R(c_{j_1} \dots c_{j_n}, d_1 \dots d_m)$ holds in M and $M, c_{j_1} \dots c_{j_n}, d_1 \dots d_m \models \varphi'$. Thus $c_{j_1} \dots c_{j_n}, d_1 \dots d_m$ is a guarded tuple.

By the (Forth) property of guarded bisimulation there is $d'_1 \dots d'_m$ such that $c'_{j_1} \dots c'_{j_n}, d'_1 \dots d'_m$ is guarded bisimilar to $c_{j_1} \dots c_{j_n}, d_1 \dots d_m$. By induction $M', c'_{j_1} \dots c'_{j_n}, d'_1 \dots d'_m \models \varphi'$, since φ' is of lower complexity. Thus

$$M', c'_{j_1} \dots c_{j_n} \models \exists y_1 \dots y_m R(x_{j_1} \dots x_{j_n}, y_1 \dots y_m) \wedge \varphi'$$

as required.

The case of guarded universal quantification is similar, but using the (Back) property. □

We now define the *guarded unravelling* of a structure M. We will do this by defining a k tree coding of a structure, denoted $\mathsf{GUnravelTree}(M)$, where k is the maximal arity of a relation. The guarded unravelling is the structure that this codes (and thus the unravelling is tree-like). The vertices V of $\mathsf{GUnravelTree}(M)$ are sequences of distinct tuples that are guarded, and the edges E order them by prefix. That is, the root might be a guarded tuple $\vec{t}_0$, one of its children in the tree would be a sequence $\vec{t}_0, \vec{t}_1$, one of its grandchildren would be a sequence $\vec{t}_0, \vec{t}_1, \vec{t}_2$. We refer to this tree (V, E) as the *guarded traversal tree* of M. Although each vertex v in the tree is a sequence of guarded tuples, we say that the vertex is *associated with* the guarded tuple $\vec{t}$ that is the last one in the sequence, and we write $\mathsf{GTuple}(v)$ for that tuple. There can be many sequences that end with the same tuple $\vec{t}$. Thus the same $\vec{t}$ will be $\mathsf{GTuple}(v)$ for infinitely many vertices v.

To obtain $\mathsf{GUnravelTree}(M)$, we label the guarded traversal tree with labels from the k-tree code signature. For a vertex v with associated guarded tuple $t_1 \dots t_m$, and each relation R in the signature of M, we label v with $R(j_1 \dots j_m)$ exactly when $R(t_1 \dots t_m)$ holds in M. We let the label $\mathsf{Eq}(i, j)$ hold of a vertex v' that is the child of another vertex v if and only if:

- the guarded tuple of v is $c_1 \dots c_r$
- the guarded tuple of v' is $d_1 \dots d_s$
- $d_j = c_i$.

That is, the equality labels of adjacent nodes reflect the overlap between the associated guarded tuples.

Proposition 9. *The labelled tree* $\mathsf{GUnravelTree}(M)$ *above is a valid* k*-tree code, where* k *is the maximal arity of a relation in the vocabulary.*

We can define $\mathsf{GUnravel}(M)$ to be the structure coded by the labelled tree $\mathsf{GUnravelTree}(M)$. Recall that each vertex v is associated with a subset of the domain elements of the structure being coded, and thus each v is associated with a substructure $\mathsf{GUnravel}(M)_v$ of $\mathsf{GUnravel}(M)$. If $\vec{t} = \mathsf{GTuple}(v)$, then the labelling of the tree guarantees that:

$\mathsf{GUnravel}(M)_v$ is isomorphic to $M_{\vec{t}}$, the substructure of M associated with $\vec{t}$.

Note that in the tree $\mathsf{GUnravelTree}(M)$, each each vertex v has children corresponding to every guarded tuple $\vec{t} \in M$. Thus we have the following *extension* property of the unravelling:

For a vertex v of the tree $\mathsf{GUnravelTree}(M)$, with associated guarded tuple $\vec{t} = \mathsf{GTuple}(v) \in M$, for each guarded tuple $\vec{t}' \in M$, there is a child v' of v with

$\mathsf{GTuple}(v') = \vec{t}'$. and $\mathsf{GUnravel}(M)_{v'}$ overlaps with $\mathsf{GUnravel}(M)_v$ exactly as $\vec{t}'$ overlaps with $\vec{t}$.

Note also that every element e' of $\mathsf{GUnravel}(M)$ corresponds to some element e of M, since it lies in $\mathsf{GUnravel}(M)_v$ for some $v = \mathsf{GTuple}(\vec{t})$ with $\vec{t} \in M$. Further, all guarded sets in $\mathsf{GUnravel}(M)_v$ arise as copies of guarded sets in M, a property we call *guarded coverage* property:

Every guarded tuple $u_1 \dots u_r$ in $\mathsf{GUnravel}(M)$ is contained in $\mathsf{GUnravel}(M)_v$ for some vertex v in the guarded traversal tree.

In particular, as with the bisimulation-based unravelling for modal logic, each element in the unravelling of M is a copy of some element $u \in M$, and there are many copies of each element.

Proposition 10. *There is a guarded bisimulation between* $\mathsf{GUnravel}(M)$ *and* M.

Proof. To prove the proposition, we give a strategy for Duplicator in the guarded bisimulation game. We will enforce the following invariant:

If $\vec{t} \in M, \vec{u} \in \mathsf{GUnravel}(M)$ are positions achieved by playing our strategy, then there is a vertex v in $\mathsf{GUnravelTree}(M)$ with $\mathsf{GTuple}(v) = \vec{t}$, and $\vec{u}$ is in $\mathsf{GUnravel}(M)_v$ with each u_i being the copy of t_i in $\mathsf{GUnravel}(M)_v$.

Assuming that we achieve this invariant, we have satisfied the partial isomorphism property required of a guarded bisimulation, since $\mathsf{GUnravel}(M)_v$ is isomorphic to $M_{\vec{t}}$.

We need to show that we can maintain the invariant in response to plays of Spoiler. If Spoiler plays a guarded tuple $\vec{t}'$ in M, we know by the extension property that v has a child v' with $M_{v'}$ where $\mathsf{GTuple}(v') = \vec{t}'$ and the overlap of M_v and $M_{v'}$ is exactly as $\vec{t}$ overlaps with $\vec{t}'$. Taking the copies of $\vec{t}'$ in $M_{v'}$ gives us a response of Duplicator in $\mathsf{GUnravel}(M)$.

If Spoiler plays a guarded tuple $u'_1 \dots u'_r$ in $\mathsf{GUnravel}(M)$, by the guarded coverage property this must be in a $\mathsf{GUnravel}(M)_v$, and v corresponds to a guarded tuple of M that has a subtuple that can be ordered as $t'_1 \dots t'_r$ with the mapping taking t'_i to u'_i a partial isomorphism of M to $\mathsf{GUnravel}(M)$. We play this $\vec{t}'$ in M. The overlaps of $\vec{t}'$ with $\vec{t}$ may not exactly match those of $\vec{u}'$ with $\vec{u}$: indeed $\vec{u}'$ may be a copy of the same $\vec{t}$ that is disjoint from $\vec{u}$. However, it is still the case that if $u'_i = u_j$, then $t'_i = t_j$, as required. One can check that the invariant is preserved by this choice, which completes the argument that Duplicator can win the game. □

The previous propositions together prove Theorem 8.

Note that we can say a bit more.

Theorem 9. *Any* GF *sentence* φ *that is satisfiable has a satisfying model that is codable by a countable tree with width* $k = \mathrm{width}(\varphi)$.

Proof. If φ has a model M, then the model can be taken to be countable, by the Lowenheim-Skolem theorem in classical logic [Chang and Keisler, 1990]. The unravelling of M is then countable. □

Using Theorem 9, we see that to decide satisfiability of a GF formula φ, it suffices to decide whether there is a model that is *codable by a tree*. But to apply our tree decidability results, we need to reduce to decidability of a logic over trees, not models that are tree-like. This "gap" is bridged by a *forward mapping theorem*, which states that we can reduce reasoning about tree-like models to reasoning about their codes.

The forward mapping holds for a very powerful logic, which we introduce here.

Guarded Second-Order logic (GSO) extends first-order logic in a similar way as Second-Order logic, but with a semantic restriction on second-order quantification. The syntax is the same as SO, but the semantics of second-order quantifiers is different. If S is a second-order quantifier of arity n, then

$$\exists S\ \varphi$$

holds in a structure M under a valuation exactly when: there is a set S_0 of n-tuples such that φ holds with the valuation extended by S_0, *and* for each n-tuple $t_1 \dots t_n \in S_0$ there is a fact in M containing each t_i. The semantics of $\forall S\ \varphi$ in GSO is defined analogously.

GSO formulas can be thought of as a syntactic restriction of SO. That is, one can think of $\exists S\ \varphi$ in GSO as a "shorthand" for $\exists S[\forall \vec{x}\ (S(\vec{x}) \rightarrow \bigvee_i \exists \vec{y}_i R_i(\vec{y}_i, \vec{x})) \wedge \varphi]$. The translation adds an additional conjunct (the first conjunct above) that restricts S to be guarded.

Example 16. *All first order formulas are in* GSO*, since* GSO *only restricts second order quantification.*

Recall the first example of second order logic formulas from Example 7:

$$\begin{aligned} \exists x_0\ x_1 U(x_0) \wedge V(x_1) \wedge \exists S \\ S(x_0) \wedge S(y_1) \wedge \\ [\forall x\ y(\ (S(x) \wedge G(x,y)) \rightarrow S(y))] \wedge \\ \neg \exists S_2\ (S_2(x_0) \wedge \exists z\ S(z) \wedge\ \neg S_2(z)) \wedge (\forall x\ y(\ (S(x) \wedge G(x,y)) \rightarrow S(y))) \end{aligned}$$

The sentence holds exactly when there is a path from a node labelled U to a node labelled V.

This property can be expressed in GSO*, since it is enough to look for a set S that is contained in the set of nodes.*

A similar observation shows that the sentence from Example 7 expressing that a graph G has a 3-coloring is expressible in GSO*. Another example would be a sentence asserting that a bi-partite graph has a matching. This could be expressed in* GSO*, since the matching is a selection of the edges in the original graph.*

For an example of a property not in GSO*, consider a signature with unary relations U, V, and the following formula with a binary second-order variable*

$B(x, y)$.

$$\exists B$$
$$[\forall xy\ B(x,y) \to U(x) \wedge V(y)] \wedge [\forall x\ U(x) \to \exists y\ B(x,y)] \wedge$$
$$[\forall y\ V(y) \to \exists x\ B(x,y)] \wedge [\forall x \neg \exists y_1\ y_2\ (y_1 \neq y_2 \wedge B(x,y_1) \wedge B(x,y_2))] \wedge$$
$$[\forall y \neg \exists x_1\ x_2\ (x_1 \neq x_2 \wedge B(x,y_1) \wedge B(x,y_2))]$$

The sentence asserts that there is a bijection from U to V. The elements of the bijection clearly will not be guarded, since there are no relations of arity above 1 *to guard it.*

The following result, due to Courcelle [Courcelle, 1990], states that evaluation of a GSO sentence over a tree-like structure reduces to evaluating an MSO sentence over the corresponding tree code:

Theorem 10. *For every* GSO *sentence φ and number k, we can construct in polynomial time an* MSO *sentence φ' in the language of k-codes such that for any model M that has a k-tree code t,*

$$M \models \varphi \leftrightarrow t \models \varphi'$$

The forward mapping that proves the theorem is a simple induction. Recall that nodes in a code represent at most k elements in the coded structure. The idea is that an element e of a model M can be represented by some element v_e of its code. For example, consider a formula

$$\varphi = R(x, y) \wedge T(y, z)$$

For a code to satisfy this, there must be a node v_1 representing the fact $R(x, y)$ and another node v_2 representing the fact $T(y, z)$. In node v_1, the value y will correspond to some index $h \leq k$. While in node v_2, y will correspond to some index $i \leq k$ possibly different from h. Further there must be a path between v_1 and v_2 whose pattern of equality facts carries along the index i to the index h. We can express this as a formula:

$$\exists v_1\ v_2 \bigvee_{g,h \leq k} \bigvee_{i,j \leq k} R(g,h)(v_1) \wedge T(i,j)(v_2)$$
$$\wedge \exists P_1 \ldots P_k\ [Path(P_1 \ldots P_k) \wedge P_h(v_1) \wedge P_i(v_2)] \wedge$$
$$(\bigwedge_{a,b \leq k} \forall v\ v'\ P_a(v) \wedge E(v,v') \wedge \mathsf{Eq}(a,b)(v') \to P_b(v'))$$

where $Path(P_1 \ldots P_k)$ is a formula expressing that the union of $P_1 \ldots P_k$ is a path in the tree.

Figure 6 gives an illustration of how a simple formula translates into an assertion about paths in a tree code.

Although the forward mapping result holds for GSO, we discuss it in more detail for a first-order formula. Our mapping will take as input a first-order

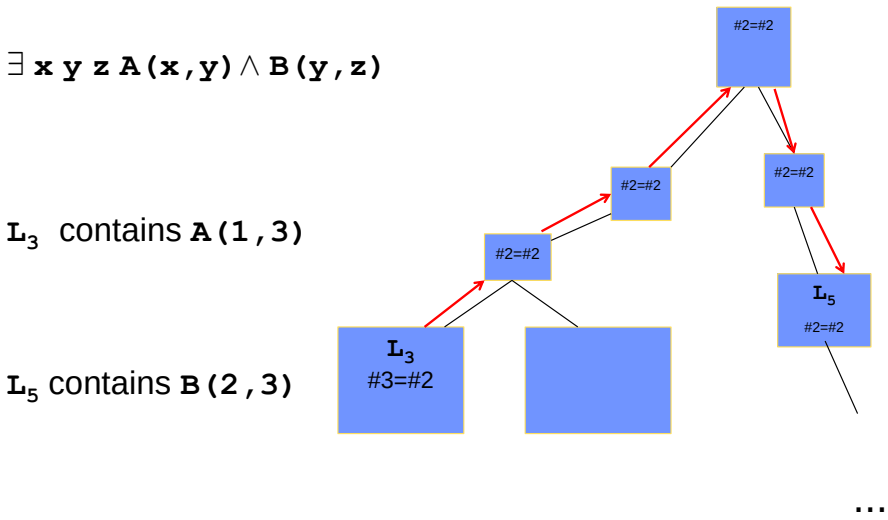

Figure 6: A formula and a path in the tree code witnessing that it holds in the tree code.

formula φ and a mapping μ taking each free variable x_i of φ to a local name. Our translation ForMap will produce a Monadic Second Order φ' with free variables $X_{x_1} \dots X_{x_k}$.

- For an atomic formula $\varphi = R(v_1 \dots v_j)$,

$$\mathsf{ForMap}(\mu, \varphi) = \exists w \bigwedge_i X_{v_i}(w) \wedge R(\mu(v_1), \dots \mu(v_j))(w)$$

- Boolean operators are pushed through the translation. E.g. $\mathsf{ForMap}(\mu, \neg\psi) = \neg\mathsf{ForMap}(\mu, \psi)$.

- For $\varphi = \exists x_{n+1}\ \psi(\vec{x}, x_{n+1})$

  $$\mathsf{ForMap}(\mu, \varphi) = \bigvee_j \exists X_{x_{n+1}}\ \mathsf{MaxConnected}(X_{x_{n+1}}, j) \wedge \mathsf{ForMap}(\mu; x_{n+1} \mapsto j, \psi)$$

  where $\mathsf{MaxConnected}(Z, j)$ expresses that Z is a connected subset of the tree such that each node in the set contains local name j in its domain, and Z is maximal with respect to this property. We can easily describe this formula in MSO, using the predicates $\mathsf{InDomain}_k$.

- Universal quantification can be reduced to negation and existential quantification.

We now state the correctness criterion for this mapping:

Lemma 2. *Let M be a structure that has a tree code t_m. Then each $c \in M$ corresponds to a connected subset* $\mathsf{subtree}(c)$ *of t_M and a local name* $\mathsf{localname}(c)$. *Fix a binding μ of free variables $\vec{x}$ of a formula φ to local names, and consider tuple $\vec{c}$ of elements of M such that* $\mathsf{localname}(c_i) = \mu(x_i)$. *Let σ be a variable binding for φ taking x_i to c_i, and σ' the corresponding binding for* $\mathsf{ForMap}(\mu, \varphi)$ *taking variable X_{x_i} to* $\mathsf{subtree}(c_i)$. *Then*

$$M, \sigma \models \varphi \text{ if and only if } t_M, \sigma' \models \mathsf{ForMap}(\mu, \varphi)$$

The lemma is proven by structural induction. We sketch here only the atomic case. Suppose $M, \sigma \models R(v_1 \dots v_j)$, and let $\sigma(v_i) = c_i$ for each i. Then there is a vertex w in the tree code that is labelled with $R(\mu(v_1), \dots \mu(v_j))$, and we can use this as a witness for $\mathsf{ForMap}(\mu, \varphi)$. Conversely, suppose there is a w satisfying $\bigwedge_i \ X_{v_i}(w) \wedge R(\mu(v_1), \dots \mu(v_j))(w)$ in the tree code of M. Using the fact that X_{v_i} is bound to $\mathsf{subtree}(c_i)$ in σ and $\mathsf{localname}(c_i) = \mu(v_i)$ in the tree code, the definition of tree code tells us that $R(c_1, \dots, c_j)$ must hold in M.

Note that GF is a subset of first-order logic. Hence we can decide if $\varphi \in \mathsf{GF}$ is satisfiable as follows:

- Applying Theorem 8, it suffices to determine if φ has a k tree-like model, where k is the width of φ.
- We can construct φ' as in Theorem 10, reducing to determining whether φ' is satisfied in a tree. This in turn can be decided using the tree satisfiability procedure of Theorem 2.

This completes the argument for the decidability of GF satisfiability, Theorem 7.

6.3 Automata for GF

In the previous subsection, we proved GF satisfiability is decidable by reducing to tree satisfiability. As with our initial approach for modal logic, this approach does not give an elementary bound on complexity, since it goes via Rabin's theorem. We will now look at a more efficient decidability technique using automata, mimicking what we did for modal logic.

For the case of GF, we need a more powerful automaton model. Since the tree models guaranteed by Theorem 8 are infinite trees, we need automata that can process such trees. In inductively translating the boolean operators from logic it will also be convenient to deal with automata that have not just nondeterminism (an analog of $\vee$), but some dual notion which is an analog of $\wedge$. Finally, since GF formulas can move backwards and forwards in a structure, it will be useful to have an automaton that can move up and down as it processes the tree.

We will define *two-way alternating automata over infinite trees*, again fixing the outdegree of nodes r and a set of node predicates $A_1 \dots A_n$ for such trees. Let Σ be $\{A_1 \dots A_n\}$. Let $\mathsf{Direction}_r$ be the set of (movement) *actions*: Stay, Down_i for $i \leq r$, and Up.

For any set J, let $B^+(J)$ be the set of positive boolean combinations of propositions in J. Given a set I of elements from J and a formula $\varphi \in B^+(J)$, the notion of φ holding in I ($I \models \varphi$) is defined as usual in propositional logic: a single element $j \in J$ holds in I if $j \in I$, a disjunction holds in I if one of its disjuncts holds, while a conjunction holds if all of its conjuncts hold. We will be interested in positive boolean combinations over $\mathsf{Direction}_r \times Q$; these formulas will be used to describe possible moves of the automaton. We will describe such a formula by talking about choices of two different players: Eve controls the nondeterministic choices ("existential" player), and Adam controls the universal choices ("for all" player). The nondeterministic choices by Eve correspond to disjunctions in the formula, and the universal choices by Adam correspond to conjunctions in the formula. For instance, we will describe a formula like $(\mathsf{Stay}, q') \vee ((\mathsf{Down}_1, q) \wedge (\mathsf{Down}_2, q))$ as saying that Eve chooses between staying in the current node and switching to q', or moving downward; if she decides to move downward, then Adam chooses whether to move downward on the left child or right child, and then switches to state q.

A two-way alternating automata over infinite trees is specified as $(Q, \Sigma, q_0, \delta, \Omega)$, where

- Q is a finite set of states
- Σ is a finite label set.
- $q_0 \in Q$ is the *initial state*
- $\delta \in Q \times \Sigma \to B^+(\mathsf{Direction}_r \times Q)$ is the *transition relation*
- Ω is an acceptance condition, which we discuss below.

The automaton can be run starting at the root of the tree, but it can also be run from some interior vertex of the tree. We thus define a *run of the automaton on a vertex v in a tree t*. This is another tree t' whose labelling function $\lambda_{t'}$ labels nodes n of t' with a vertex v_n of t and a state $q_n \in Q$. We now describe further properties that are required for the run to be *accepting*.

First we require that the root of t' is assigned to v and state q_0. That is, the computation starts at the initial state and the initial vertex.

Second, we require that the relationship between parent and children labels in t' be consistent with the transition function δ. Suppose a vertex n of t' is associated by $\lambda_{t'}$ to a vertex v_n of t with label σ_n, and also to a state q_n, and let C_n be the children of n in t'. Then we require that $\lambda_{t'}$ associate each $c' \in C_n$ with a vertex of t that is either v_n, a parent of v_n, or a child of v_n.

Given the above requirement we can associate each child $c'_i \in C_n$ with a direction $d'_i \in \mathsf{Direction}_r$ as well as a state $q'_i \in Q$. Let P_n be the set of pairs (d'_i, q'_i) associated with some child of n. We require that $P_n \models \delta(q, \sigma_n)$.

Finally, we require that every path of t' obey the acceptance condition Ω. There are a number of different acceptance conditions defined for automata over infinite trees. For GF we will need only the *Büchi acceptance condition*, which is specified by a subset F of Q. The corresponding requirement is that along

each path of a run t', there are infinitely many nodes that are associated with states in F.

We say that an automaton $\mathcal{A}$ *accepts* a tree t if there is an accepting run of $\mathcal{A}$ starting from the root of t. We let $\mathsf{2ABT}^{\omega}$ denote the class of two-way alternating automata over infinite trees with the Büchi condition. Note that a 1-way nondeterministic Büchi automaton, denoted $\mathsf{1NBT}^{\omega}$, is just a $\mathsf{2ABT}^{\omega}$ where every transition function formula is of the form $\bigvee_i(\mathsf{Down}_1, q_1)\wedge\cdots\wedge(\mathsf{Down}_r, q_r)$.

As with other automata, we can apply this to trees with finitely many predicates by considering the label alphabet as a powerset.

The main computation we will need on these automata is a non-emptiness test:

Proposition 11 ([Vardi, 1998])**.** *There is an algorithm for testing non-emptiness of a* $\mathsf{2ABT}^{\omega}$ $\mathcal{A}$ *that runs in exponential time. More specifically, if* $\mathcal{A}$ *has size* n *and* s *states, the running time is bounded by* $f(n)^{f(s)}$ *where* f *is a polynomial.*

6.4 Optimized decidability of GF via translation to an automaton

As a warm-up, let us first return to the case of modal logic, and consider how we could construct an alternating automaton $\mathcal{A}_\varphi$ for a modal logic formula φ.

Let φ be a formula in modal logic. Assume φ is in negation normal form, so negations are pushed inside as far as possible. Then we define the automaton $\mathcal{A}_\varphi$ for φ as follows:

- The state set is the set of all subformulas of φ with a fresh c substituted for any free variable, together with $\{\mathsf{True}, \mathsf{False}\}$.
- The initial state is φ itself.
- The set of accepting states is $\{\mathsf{True}\}$.
- The transition function δ is defined as follows:

$$\delta(P(c), \tau) := \begin{cases} (\mathsf{Stay}, \mathsf{True}) & \text{if } P(c) \in \tau \\ (\mathsf{Stay}, \mathsf{False}) & \text{if } P(c) \notin \tau \end{cases}$$

$$\delta(\neg P(c), \tau) := \begin{cases} (\mathsf{Stay}, \mathsf{True}) & \text{if } P(c) \notin \tau \\ (\mathsf{Stay}, \mathsf{False}) & \text{if } P(c) \in \tau \end{cases}$$

$$\delta(\mathsf{True}, \tau) := (\mathsf{Stay}, \mathsf{True})$$

$$\delta(\mathsf{False}, \tau) := (\mathsf{Stay}, \mathsf{False})$$

$$\delta(\psi_1 \vee \psi_2, \tau) := (\mathsf{Stay}, \psi_1) \vee (\mathsf{Stay}, \psi_2)$$

$$\delta(\psi_1 \wedge \psi_2, \tau) := (\mathsf{Stay}, \psi_1) \wedge (\mathsf{Stay}, \psi_2)$$

$$\delta(\exists y\ R(c, y) \wedge \psi(y), \tau) := \bigvee_{1\le i\le r}(\mathsf{Down}_i, \psi(c))$$

$$\delta(\forall y\ R(c, y) \rightarrow \psi(y), \tau) := \bigwedge_{1\le i\le r}(\mathsf{Down}_i, \psi(c))$$

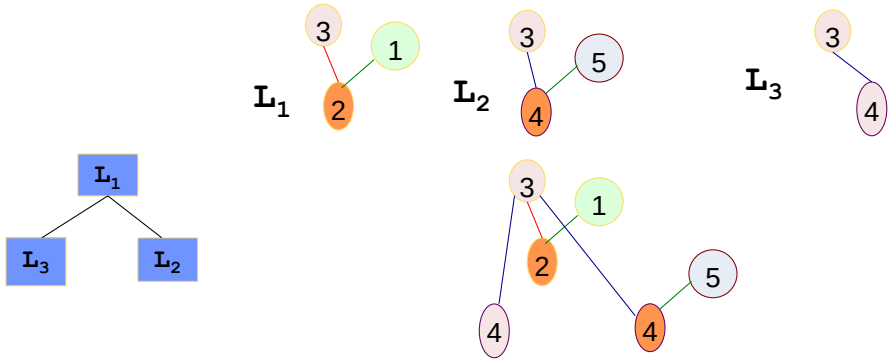

Figure 7: Implicit coding

The idea is that if the automaton is in state ψ at node v in the input tree t, then the existential player Eve is trying to show that ψ holds at v, and the universal player Adam is trying to show otherwise. Note that the number of states of this automaton is only polynomial in the size of the input φ; however, we pay an exponential price to test for non-emptiness of a $\mathsf{2ABT}^{\omega}$ like this, so we get an EXPTIME upper bound on satisfiability testing for modal logic, as we did in our previous procedure.

The automaton for a GF sentence φ is a generalization of this. As before, we assume that φ is in negation normal form. For ease of exposition, we also assume that there is no use of equality.

It is convenient to code relational structures by trees in a slightly different way. We will still have a node in a tree code represent k elements in a structure, for some fixed k. Recall that previously we labelled nodes of a tree code with special predicates to say how the k elements associated to that node relate to the k elements associated to the parent of the node. We call this *explicit equality coding*. An alternative is *implicit equality coding*; we associate each node with at most k numbers within the set $N_k = \{1, \ldots, 2 \cdot k\}$; the *local names* of the node. We write names(v) for the local names used in node v in a tree code. We label the node with relations mentioning those local names. If a node and its parents overlap on their local names, this means the two elements are the same.

Figure 7 gives an example of implicit coding. We have $k = 3$, so each of L_1, L_2, and L_3 is describing a structure with 3 elements, and each of these elements is described using a number from 1 to 6. The tree at the right codes a structure. To figure out what structure is being coded by this tree (the structure on the bottom right), we glue the structures L_i for adjacent nodes together, using the common numbers as gluing points.

Note that if we choose a node in a tree code and a local name, we have described exactly one element of the coded structure. The corresponding tree code signature for implicit coding is similar to the one for explicit coding, in that we have unary predicates for each fact over the local names. We do not

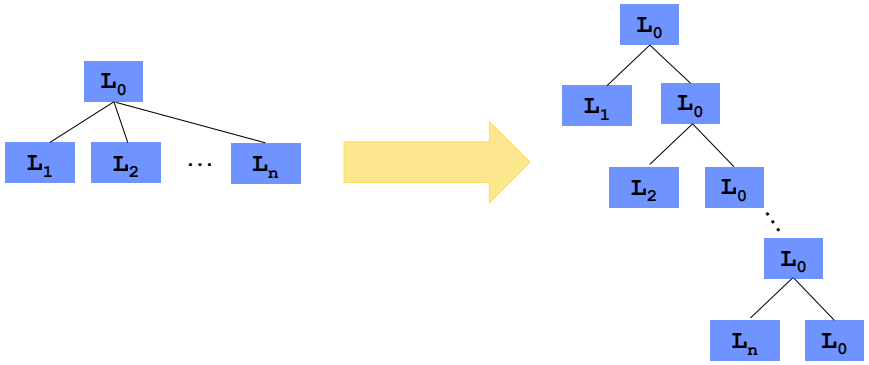

Figure 8: Conversion to binary trees

have equalities any more, and we will need to have unary predicates $\mathsf{InDomain}_k$ for k a local name $\mathsf{InDomain}_k(v)$ indicates that local name k is in the domain of v. This predicate will be use to determine which local names are identified in the decoding.

Tree codes can generally have unbounded (possibly infinite) degree. More precisely, Theorem 9 showed that a code can have at most countably many children. For technical reasons, it is more convenient to use binary trees for our encodings. Any tree code where each node has at most countably many children can be converted to a binary tree code in the following way: starting from the root, replace each node u with children $v_1, v_2, \ldots$ with the subtree consisting of $v_1, v_2, \ldots$ and new nodes $u_1, u_2, \ldots$ such that the label at each u_i is the same as the label at u, the left child of u_i is v_i and the right child of u_i is u_{i+1}.

We can also make the code into a full binary tree by adding extra copies of what was previously a leaf node.

After doing this, each node in a binary tree can be identified with a finite string over $\{1, 2\}$, with ϵ identifying the root, and $u1$ and $u2$ identifying the left child and right child of u.

Figure 8 illustrates the conversion to binary tree codes.

We write $\Sigma^{\text{code}}_{\sigma,k}$ for the signature of these tree codes, and write $\Sigma^{\text{code}}_{\sigma,k}$-tree to mean a labelled binary tree over this signature.

We are now ready to describe the automaton construction for a GF sentence φ.Let $\mathsf{cl}(\varphi, N_k)$ be the set of subformulas of φ with names from N_k substituted in for any free variables, together with True and False. The automaton $\mathcal{A}_\varphi$ for φ is defined as follows:

- The state set is $\mathsf{cl}(\varphi, N_k)$.
- The initial state is φ.
- The transition function δ is defined below.

- The set of accepting states consists of all states of the form True, $\neg R(\vec{a})$, or $\forall \vec{y}\ R(\vec{a},\vec{y}) \to \psi(\vec{a},\vec{y})$.

We now describe the transition function.

$$\delta(R(\vec{a}),\tau) := \begin{cases} (\mathsf{Stay},\mathsf{False}) & \text{if } \vec{a} \text{ not represented in } \tau \\ (\mathsf{Stay},\mathsf{True}) & \text{if } R(\vec{a}) \in \tau \\ \bigvee_{d\in\mathsf{Dir}}(d, R(\vec{a})) & \text{otherwise} \end{cases}$$

$$\delta(\neg R(\vec{a}),\tau) := \begin{cases} (\mathsf{Stay},\mathsf{True}) & \text{if } \vec{a} \text{ not represented in } \tau \\ (\mathsf{Stay},\mathsf{False}) & \text{if } R(\vec{a}) \in \tau \\ \bigwedge_{d\in\mathsf{Dir}}(d, \neg R(\vec{a})) & \text{otherwise} \end{cases}$$

$$\delta(\mathsf{True},\tau) := (\mathsf{Stay},\mathsf{True})$$

$$\delta(\mathsf{False},\tau) := (\mathsf{Stay},\mathsf{False})$$

$$\delta(\psi_1 \vee \psi_2,\tau) := (\mathsf{Stay},\psi_1) \vee (\mathsf{Stay},\psi_2)$$

$$\delta(\psi_1 \wedge \psi_2,\tau) := (\mathsf{Stay},\psi_1) \wedge (\mathsf{Stay},\psi_2)$$

$$\delta(\exists \vec{y}\ R(\vec{a},\vec{y}) \wedge \psi(\vec{a},\vec{y}),\tau) := \begin{cases} (\mathsf{Stay},\mathsf{False}) & \text{if } \vec{a} \text{ not represented in } \tau \\ \bigvee_{R(\vec{a},\vec{b})\in\tau}(\mathsf{Stay},\psi(\vec{a},\vec{b}))\vee & \\ \quad \bigvee_{d\in\mathsf{Dir}}(d, \exists \vec{y}\ R(\vec{a},\vec{y}) \wedge \psi(\vec{a},\vec{y})) & \text{otherwise} \end{cases}$$

$$\delta(\forall \vec{y}\ R(\vec{a},\vec{y}) \to \psi(\vec{a},\vec{y}),\tau) := \begin{cases} (\mathsf{Stay},\mathsf{True}) & \text{if } \vec{a} \text{ not represented in } \tau \\ \bigwedge_{R(\vec{a},\vec{b})\in\tau}(\mathsf{Stay},\psi(\vec{a},\vec{b}))\wedge & \\ \quad \bigwedge_{d\in\mathsf{Dir}}(d, \forall \vec{y}\ R(\vec{a},\vec{y}) \to \psi(\vec{a},\vec{y})) & \text{otherwise} \end{cases}$$

For the case of a positive atomic formula, notice that Eve is given the ability to search for a node in the tree code that witnesses the desired fact. Likewise, for guarded existential quantification formulas, Eve tries to find a witness satisfying the body of the quantified formula. This witness can come from elements that are represented in the current node, or elements that are represented elsewhere in the tree. This leads to the disjuncts in $\bigvee_{R(\vec{a},\vec{b})\in\tau}(\mathsf{Stay},\psi(\vec{a},\vec{b})) \vee \bigvee_{d\in\mathsf{Dir}}(d, \exists \vec{y}\ R(\vec{a},\vec{y}) \wedge \psi(\vec{a},\vec{y}))$. While searching, she is in a non-accepting state, so she cannot cheat and forever defer her choice of witness. Likewise, if she moves into part of the tree where $\vec{a}$ is not represented, then she immediately loses (she must move to a non-accepting sink state False).

Negated atomic formulas or guarded universal quantification can be seen as the dual of this, with Adam controlling the search for a counterexample.

This completes the construction. The automaton satisfies the following property:

Lemma 3. *Consider any $\rho \in \mathsf{cl}(\varphi, N_k)$, any consistent tree code t which codes model M, and any node $v \in t$ containing the names in ρ. Let $l_1 \dots l_i$ be the local names in ρ and $e_1 \dots e_i$ the corresponding elements of M. Then ρ holds of $e_1 \dots e_i$ in M if and only if $\mathcal{A}_\varphi$ accepts t when run starting from v in using ρ as the initial state.*

In particular if φ is a sentence of GF*. then for all consistent trees t the* $2ABT^{\omega}$ *automaton $\mathcal{A}_\varphi$ accepts t from the root starting at state φ iff* $\text{decode}(t) \models \varphi$*.*

The lemma is proven using induction.

Combining Lemma 3 with Theorem 8 we see that:

A guarded fragment sentence φ is satisfiable if and only if there is a tree that is a valid k-code of a structure which is accepted by the automaton $\mathcal{A}_\varphi$ referred to in the lemma.

We can easily write an automaton $\mathcal{A}_{consistent}$ that accepts a tree exactly when it is the code of some structure: Thus combining with the above, we obtain that satisfiability testing for GF sentences φ reduces to a non-emptiness test of $L(\mathcal{A}_\varphi \cap \mathcal{A}_{consistent})$. Since the intersection of two automata can be converted to a single automaton, we have reduced satisfiability to automaton non-emptiness.

What is the cost for testing satisfiability using this automaton approach? The number of node label predicates in the code alphabet will be exponential when the arity of the relations is not fixed. Recalling that the transition relation of these automata work over $\mathcal{P}(\Sigma)$ we can see that the transition relation will be doubly-exponential. Thus the size of the automata is doubly exponential in the size of φ, since the alphabet is already of doubly exponential size. This would suggest that when we apply the non-emptiness test of Proposition 11, we would get a triply exponential upper bound. However, recall from Proposition 11 that the emptiness test for $2ABT^{\omega}$ is actually exponential only in the number of states. The automaton $\mathcal{A}_\varphi$ has only exponentially many states. Specifically, there are at most $f(|\varphi|) \cdot 2^{f(\text{width}(\varphi))}$ states for f some polynomial function independent of φ. Hence, satisfiability is in 2EXPTIME (and can be even be shown to be 2EXPTIME-complete).

When we fix the maximum arity of relations, the number of node label predicates becomes polynomials, and the size of the automaton drops to this drops to singly-exponential. Since the number of states of this automaton is polynomial under this restriction, we can use Proposition 11 to get a single-exponential bound.

Theorem 11. *Satisfiability for* GF *is* 2EXPTIME*-complete in general, and* EXPTIME*-complete when the maximum arity of relations is fixed.*

6.5 Bibliographic remarks

GF was introduced in a series of articles by Andréka, van Benthem, and Németi, leading to [Andréka et al., 1998]. The complexity of satisfiability was isolated by Grädel in [Grädel, 1999b]. Our exposition roughly follows [Grädel, 1999a].

Guarded bisimulations and the corresponding unravellings are discussed in [Grädel, 1999a] and [Grädel and Otto, 2014].

A good introduction to alternating automata on infinite trees can be found in [Löding, 2011]. More background on automata and logic connections can also be found in [Thomas, 1997a].

Grädel proved that the guarded fragment has the *finite model property*: whenever a sentence is satisfiable, it is satisfied by a finite structure [Grädel, 1999b]. It follows from this that finite satisfiability of GF is decidable. The proof of the finite model property for GF requires techniques beyond the automata-to-logic connection, and we will not discuss it here.

7 The Guarded Negation Fragment

7.1 GNF basics

We will now extend the argument for GF to a richer fragment called the *guarded negation fragment* GNF.

The atomic formulas of GNF are True, False, relational atoms, and equality atoms.

Formulas are defined inductively by:

- if φ_1 and φ_2 are formulas of GNF, then $\varphi_1 \wedge \varphi_2$, and $\varphi_1 \vee \varphi_2$, and $\exists x\ \varphi$ are formulas of GNF;
- if φ is a formula then $R(\vec{x}) \wedge \neg\varphi$ is a formula, where the free variables of φ are required to be contained in the variables $\vec{x}$ of the atom $R(\vec{x})$. R is the *guard* of the negated formula;
- if φ is a formula with at most one free variable then $\neg\varphi$ is a formula.

The last item is sometimes phrased as a special case of the second item, using trivial guards: $\mathsf{True} \wedge \neg\varphi$ for a sentence φ, and "an equality guard" $x = x \wedge \neg\varphi(x)$ for a formula φ with one free variable x.

Syntactically, GNF does not extend GF, since it does not allow universal quantification. However, every GF *sentence* can be easily rewritten in GNF:

Proposition 12. *Every* GF *sentence is equivalent to a* GNF *sentence.*

Consider the first example GF sentence presented in the previous section:

$$\forall xyz\ S(x,y,z) \rightarrow \exists uv\ S(z,u,v)$$

We would express this in GNF as:

$$\neg\exists xyz\ [S(x,y,z) \wedge \neg\exists uv\ S(z,u,v)]$$

We have translated universal quantification and implication into existential quantification, conjunction, and negation in the usual way. The outermost negation is vacuously guarded, since there are no free variables. The inner negation has the atom $S(x,y,z)$ that guards the free variables of the negated formula.

The proof of Proposition 12 is an inductive algorithm. For every GF formula $\varphi(\vec{x})$ and every guard $R(\vec{x},\vec{y})$ of the free variables of φ it produces a GNF formula $\varphi'(\vec{x},\vec{y})$ that is equivalent to $R(\vec{x},\vec{y}) \wedge \varphi(\vec{x})$.

Unsurprisingly, GNF allows one to express many things that are not expressible in GF.

Example 17. *Every positive existential formula (one built up using only* $\vee, \wedge, \exists$*) is expressible in* GNF*. For example, the formula*

$$\exists x_1\ x_2\ x_3\ R(x_1,x_2) \wedge R(x_2,x_3) \wedge R(x_3,x_1)$$

is in GNF *but is not expressible in* GF.

Since GNF *is closed under boolean combinations of sentences, it can also express that one positive existential formula follows from another using a set of* GF *axioms. For example, the sentence:*

$$\begin{aligned}\exists x_1\ x_2\ x_3\ (R(x_1,x_2) \wedge R(x_2,x_3) \wedge R(x_3,x_1)) \wedge \\ \forall xy\ (R(x,y) \rightarrow S(x,y)) \wedge \\ \neg\exists x_1\ x_2\ x_3\ (S(x_1,x_2) \wedge S(x_2,x_3) \wedge S(x_3,x_1))\end{aligned}$$

is in GNF. *It is unsatisfiable if and only if the positive existential sentence* $\exists x_1\ x_2\ x_3\ S(x_1,x_2) \wedge S(x_2,x_3) \wedge S(x_3,x_1)$ *follows from the positive existential sentence* $\exists x_1\ x_2\ x_3\ R(x_1,x_2) \wedge R(x_2,x_3) \wedge R(x_3,x_1)$ *assuming the* GF *sentence* $\forall xy\ (R(x,y) \rightarrow S(x,y))$.

Generalizing the last example, consider a set of GF sentences Σ and two positive existential sentences φ_1, φ_2. Consider the problem of deciding whether $\Sigma \wedge \varphi_1 \rightarrow \varphi_2$ is valid. This is sometimes expressed by saying: φ_1 implies φ_2 relative to Σ. $\Sigma \wedge \varphi_1 \rightarrow \varphi_2$ is valid exactly when $\Sigma \wedge \varphi_1 \wedge \neg\varphi_2$ is unsatisfiable, and thus this problems reduces to satisfiability of a GNF sentence.

Recall from Section 2 that positive existential formulas of the form $\exists\vec{x}\,\psi(\vec{x},\vec{y})$ where $\psi(\vec{x})$ is a conjunction of atoms are known as *conjunctive queries*, or CQs. *Unions of conjunctive queries* (UCQs) are disjunctions of formulas like this. UCQs are expressible in GNF, but are not in general expressible in GF.

It is often helpful to consider the formulas of GNF in a certain normal form that comes from nesting UCQs using guarded negation. The *normal form for* GNF over a signature σ can be defined recursively to be formulas of the form $\delta[Y_1 := \alpha_1 \wedge \neg\varphi_1, \ldots, Y_n := \alpha_n \wedge \neg\varphi_n]$ where

- δ is a UCQ over $\sigma \cup \{Y_1, \ldots, Y_n\}$ for some fresh relations $Y_1, \ldots, Y_n$,
- $\varphi_1, \ldots, \varphi_n$ are in normal form GNF,
- $\alpha_1, \ldots, \alpha_n$ are guards for the free variables in $\varphi_1, \ldots, \varphi_n$ such that the number of free variables in each $\alpha_i \wedge \neg\varphi_i$ matches the arity of Y_i, and
- $\delta[Z := \psi]$ is the result of replacing every occurrence of $Z(\vec{x})$ in δ with $\psi(\vec{x})$.

Note that the base case of this recursive definition is a UCQ over σ (take $n = 0$ above).

Every GNF formula can be converted into this form in a canonical way with an exponential blow-up in size.

We refer to formulas built up like this as *UCQ-shaped formulas*. Likewise, we say a formula is a *CQ-shaped formula* if it is of the form $\delta[Y_1 := \alpha_1 \wedge \neg\varphi_1, \ldots, Y_n := \alpha_n \wedge \neg\varphi_n]$ for δ a CQ, and α_i and φ_i as above.

As with GF, we will need a notion of the *width* of a GNF formula, which will relate to the notion of tree-like model that is appropriate. The width of a GNF formula φ in the normal form above is the maximum number of free variables in

any subformula, and we extend this to non-normal form formulas via converting to normal form: that is, we denote by GNF^k the set of GNF-formulas that are of width k when they are brought into this normal form.

We will also keep track of two other parameters, CQ-rank and negation depth.

The *CQ-rank* of a formula φ in normal form is the maximum number of conjuncts in any CQ-shaped subformula of φ that begins with an existential quantifier.

Let us consider the distinction between width and CQ-rank. We note first that a formula may have low CQ-rank but still have high width. This clearly can happen if the maximal arity of the relations in the schema is not fixed, since CQ-rank counts only the number of conjuncts, not the number of variables. But even for a schema where relations all have arity 2, the width can be much higher than the CQ-rank, since the formula may have UCQ-shaped formulas that have many variables, but where every CQ has a small number of conjuncts. Conversely, if we do not fix the schema, a formula may have low width but still have high CQ-rank. For example, the formula

$$U_1(x) \wedge U_2(x) \wedge \ldots U_n(x)$$

has width only 1, since we have only one variable. But its CQ-rank is n. In GF, we saw the width of a formula has an impact on the size of the "bags" in a tree-like model of the formula. Through this, the width impacts the size of the *alphabet* in a tree automaton accepting tree-like models of the formula. The same will hold for GNF. We will see in Section 7.4 and Section 7.5 that the CQ-rank has an important impact on the *number of states* of the automata for a GNF formula.

The *negation depth* is the maximal nesting depth of negations in the syntax tree of φ. This parameter will simply be useful for doing inductions within proof.

7.2 Decidability of GNF via the tree model property

Our aim is to show:

Theorem 12. *Satisfiability of a* GNF *sentence is decidable.*

We will proceed similarly to GF, by showing the tree-like model property. Let us first talk about what kind of trees we will need. As for GF, we will look at trees that code structures; a node in the tree will code several elements in the structure. As with GF some tree nodes code the behavior of a guarded set. But we will also need tree nodes that code the existence of a witness to a positive existential sentence. Such nodes may need to be big enough to contain witnesses for the existential quantifiers in formulas of the form φ in the normal form grammar above. This is related to the *width* of a formula, defined previously.

We can now state the tree-model property for GNF:

Theorem 13. *Any* GNF^k *sentence* φ *that is satisfiable has a satisfying model which is codable with width* k*, where* k *is the maximal number of free variables in any subformula of* φ*.*

In order to define the notion of bisimulation that is relevant for GNF, we will use a game. Each position in the game is a partial homomorphism h from M to M', or vice versa, where the domain of h has size at most k. The *active structure* in position h is the structure containing the domain of h. The GNF^k-game starts from the empty partial map from M to M'. In each round of the game, Spoiler chooses between one of the following moves:

- *Modify up to k:* Spoiler can delete an element of the domain of the active structure, resulting in a mapping h' with a smaller domain. In this case Duplicator need do nothing. Spoiler can also add an element to the domain of the active structure, as long as the size of the resulting domain of the is no more than k; if Spoiler makes such a move, Duplicator must extend to a homomorphism h' that includes the new element. Duplicator loses if this is not possible. Otherwise, the game proceeds from the position h'.
- *Guarded Switch:* If $\mathsf{domain}(h)$ is guarded, then Spoiler can choose to switch active structure. If h is not a partial isomorphism, then Duplicator loses. Otherwise, the game proceeds from the position h^{-1}.

Duplicator wins if she can continue to play indefinitely.

We say that $(M, \vec{x})$ and $(M', \vec{x}')$ are GNF^k-bisimilar if Duplicator can win the GNF^k-game starting from a position where the partial homomorphism maps x_i to x'_i, and the active structure is M. Note that in the case $\vec{x}$ is guarded, then Duplicator can win from position where the active structure is M and the partial homomorphism maps x_i to x'_i if and only if Duplicator wins from the position where the active structure is M' and the partial homomorphism maps x'_i to x_i. That is, for guarded tuples this notion of bisimilarity is an equivalence relation.

Proposition 13. *If* $(M, \vec{x})$ *and* $(M', \vec{x}')$ *are* GNF^k*-bisimilar and* $\vec{x}$ *is guarded, then the* GNF *formulas of width* k *satisfied by* $\vec{x}$ *in* M *are the same as those satisfied by* $\vec{x}'$ *in* M'*.*

Proof. To prove the proposition, we show a more general statement:

If there is a winning strategy for Duplicator on a position of GNF^k game on $(M, \vec{c})$ and $(M', \vec{c}')$ with active structure M, then every GNF^k formula $\varphi(\vec{x})$ holding at $M, \vec{c}$ holds at $M', \vec{x}$.

Notice that this is a "uni-directional" statement, stating only that formulas are preserved from M to M'. But it implies Proposition 13, since if the positions are guarded then Spoiler is free to switch structures, so if Duplicator can win from M she can also win from M'.

The statement is proven by a simple structural induction on GNF^k formulas. As usual, the atomic cases follow from the Partial Isomorphism requirement in the game, and the cases for $\vee$ and $\wedge$ follow straightforwardly from the definitions.

Consider the inductive case of a UCQ-shaped subformula φ that holds in $(M, \vec{m})$ and we want to show holds in $(M', \vec{m}')$. There is an extension of $\vec{m}$ to $\vec{n}$ that has at most k elements that witnesses the satisfaction of the formula. We can consider a set of Modify moves of Spoiler that add these additional elements. By GNF^k-bisimilarity there is a response of Duplicator $\vec{n}'$. For any guarded subset $\vec{n}_0$ of $\vec{n}$, we know by induction that the corresponding set $\vec{n}_0$ will satisfy the same subformulas of our formula. Using this we can see that $\vec{n}'$ witnesses that φ holds in $(M', \vec{m}')$.

Consider the inductive case for a guarded negation formula $R(\vec{x}) \wedge \neg\varphi(\vec{x})$ holding of $\vec{c}$ in M. Then $R(\vec{c})$ must hold in M, and thus by Partial Isomorphism (or induction) $R(\vec{c}')$ must hold in M'. Assume by way of contradiction that φ did not hold of $\vec{c}'$ in M'. Since $\vec{c}$ is guarded, Spoiler could switch structures, and by induction we would know that $\varphi(\vec{x})$ must hold of $\vec{c}$ in M, a contradiction.

This completes the argument for Proposition 13. □

We will now define an unravelling based on this notion of GNF^k-bisimulation. As before, we will do this by defining a k tree coding of a structure, denoted $\mathsf{GNUnravel}_k$. The GNF^k-unravelling is the decoding of this structure. Recall that our goal is to create a tree-like structure GNF^k-bisimilar to M. The intuition will be that we anticipate possible plays of the Spoiler within M, and we create the unravelling so that there is a response to these plays.

Given a structure M, the underlying tree of our tree code will have vertices V that are sequences of tuples where each tuple is of size at most k. Observe that these are exactly what we get by looking at plays in the game and restricting to the sequences in M. The tree will be ordered by an edge relation E that is just the prefix ordering. We refer to (V, E) as the *k-tuple traversal of M*. Note that each vertex $v \in V$ is associated to a tuple $\mathsf{GTuple}(v)$ of size at most k, the final tuple in the sequence.

We label the tree just as in the case of the guarded traversal. That is, $\mathsf{Eq}(i, j)$ holds of a child s' of a node s if and only if:

- s ends with tuple $c_1 \dots c_r$
- s' extends s with a single tuple $d_1 \dots d_s$
- $d_j = c_i$.

$R(m_1, \dots, m_j)$ holds at a sequence s if and only if the final tuple of s is $\vec{c}$ and $R(c_{m_1}, \dots, c_{m_j})$ holds in M.

Proposition 14. $\mathsf{GNUnravelTree}_k(M)$ *is a valid k tree code.*

Again analogously to what we did for GF, we can define $\mathsf{GNUnravel}_k(M)$ to be the structure coded by the labelled tree $\mathsf{GNUnravelTree}_k(M)$. Again we have each vertex v is associated with a tuple $\vec{t} = \mathsf{GTuple}(v)$ of domain elements of M. The substructure $\mathsf{GNUnravel}_k(M)_v$ of $\mathsf{GNUnravel}_k(M)$ is isomorphic to the substructure of M over $\vec{t}$, $M_{\vec{t}}$. $\mathsf{GTuple}(v)$ is no longer a guarded tuple, but its size is bounded by k.

As in the guarded unravelling, we have the following *extension* property:

For a vertex v of the tree $\mathsf{GNUnravelTree}_k(M)$, with associated tuple $\vec{t} = \mathsf{GTuple}(v) \in M$, for each tuple $\vec{t'} \in M$ of size at most k, there is a child v' of v with $\mathsf{GTuple}(v') = \vec{t'}$. and $\mathsf{GNUnravel}_k(M)_{v'}$ overlaps with $\mathsf{GNUnravel}_k(M)_v$ exactly as $\vec{t'}$ overlaps with $\vec{t}$.

We still have the *guarded coverage* property:

Every guarded tuple $u_1 \dots u_r$ in $\mathsf{GNUnravel}_k(M)$ is contained in $\mathsf{GNUnravel}_k(M)_v$ for some vertex v in the guarded traversal tree.

Using these properties we can show that $\mathsf{GNUnravel}_k(M)$ is GNF^k-bisimilar to M:

Proposition 15. *$\mathsf{GNUnravel}_k(M)$ and M are GNF^k-bisimilar.*

Proof. To prove the proposition, we give a strategy for Duplicator in the GNF^k-bisimulation game. We will enforce a weakening invariant as in the GF case:

If $\vec{t} \in M, \vec{u} \in \mathsf{GNUnravel}_k(M)$ are positions achieved by playing our strategy, then

- each u_i is a copy of t_i; that is, u_i is in $\mathsf{GNUnravel}_k(M)_v$ where $\vec{w} = \mathsf{GTuple}(v)$ contains t_i, and the isomorphism of $M_{\vec{w}}$ to $\mathsf{GNUnravel}_k(M)_v$ takes t_i to u_i.
- if $\vec{u}$ is guarded, or if the active structure is M, then there is a vertex v in $\mathsf{GNUnravelTree}_k(M)$ with $\mathsf{GTuple}(v) = \vec{t}$, and $\vec{u}$ is in $\mathsf{GNUnravel}_k(M)_v$ with each u_i being the copy of t_i in $\mathsf{GNUnravel}_k(M)_v$.

We need to show that we can maintain the invariant in response to plays of Spoiler.

If the active structure is M, Spoiler can modify the k-tuple in M to get another k-tuple $\vec{t'}$ in M. We know by induction that the second invariant holds of the position before Spoiler's move, and thus there is v such that Duplicator's position $\vec{u}$ is contained in $\mathsf{GNUnravel}_k(M)_v$. By the extension property, v has a child v' where $\mathsf{GTuple}(v') = \vec{t'}$ and the overlap of $\mathsf{GNUnravel}_k(M)_v$ and $\mathsf{GNUnravel}_k(M)_{v'}$ is exactly as $\vec{t}$ overlaps with $\vec{t'}$. Taking the copies of $\vec{t'}$ in $\mathsf{GNUnravel}_k(M)_{v'}$ gives us a response of Duplicator in $\mathsf{GNUnravel}_k(M)$.

If the active structure is $\mathsf{GNUnravel}_k(M)$, Spoiler modifies $\vec{u}$ to get a tuple $u'_1 \dots u'_r$ in $\mathsf{GNUnravel}_k(M)$. Each $u'_1 \dots u'_r$ is a copy of some $t'_1 \dots t'_r$ in M, and Duplicator responds by playing $\vec{t'}$. Clearly the first invariant is preserved. Further, the guarded coverage property implies that the second invariant is preserved.

Spoiler can also move to switch structures. If the active structure is M, this move does not impact the invariant. If the active structure is $\mathsf{GNUnravel}_k(M)$, we need to be sure that the second invariant holds. But the rules of the game say that Spoiler can only switch when the positions are guarded, and hence the second invariant is preserved. This completes the argument that Duplicator can win the game. □

The previous propositions together prove Theorem 13.

Now we note that GNF is also a subset of GSO, thus the forward mapping result, Theorem 10 applies to GNF sentences. Hence we can decide satisfiability for $\varphi \in \mathsf{GNF}$ using the same technique as Theorem 7: apply a forward mapping on φ to get a φ' on tree codes, and check satisfiability of φ' using Rabin's theorem. This completes the proof of the decidability of GNF, Theorem 12.

7.3 Automata for GNF

As before, we have shown decidability of GNF via the tree model property and tree satisfiability. Now we will look for a more efficient procedure using automata. We will need some additional machinery from automata theory to help with our construction of an automaton from a GNF sentence, which we explain here.

First, we will make use of another common acceptance condition called a *parity condition*. This is specified by a mapping Ω that assigns a priority (natural number) to each state. The requirement is that along each path of a run, the maximum priority occurring infinitely often is even. This is a generalization of the Büchi condition seen earlier: a Büchi condition based on some set of states F can be converted to a priority mapping Ω that assigns priority 2 to states in F and priority 1 to everything else. Let $\mathsf{2APT}^\omega$ denote 2-way alternating tree automata equipped with this parity acceptance condition. Similarly, let $\mathsf{1NPT}^\omega$ denote 1-way nondeterministic parity tree automata.

Our construction will proceed in a more general and modular way, taking advantage of *closure properties* of $\mathsf{2APT}^\omega$ automata:

Proposition 16. $\mathsf{2APT}^\omega$ *automata are closed under union, intersection, and complementation. The constructions can be done in time polynomial in the size of the input, and the number of priorities is linear in the number of priorities in the original.*

Another important closure property is closure under projection. Let $\Sigma' := \Sigma \cup \{U_1, \ldots, U_j\}$, and let L' be a language of Σ'-trees. The *projection* of L' to Σ is the language L consisting of trees t over Σ such that there is some $t' \in L'$ that agrees exactly with t on Σ (i.e. there is some annotation of t with relations in $\Sigma' \setminus \Sigma$ that is in L'). Projection is easy starting with a 1-way nondeterministic automaton $\mathcal{A}'$ for L' — the automaton $\mathcal{A}$ for the projected language L just nondeterministically guesses the missing information in $\Sigma' \setminus \Sigma$ while simulating $\mathcal{A}$. This straightforward construction does not work directly for alternating automata. However, it turns out that we can always convert between 2-way alternating and 1-way nondeterministic versions of the automata, and then do the projection. In particular, [Vardi, 1998] showed that any $\mathsf{2APT}^\omega$ automaton can be converted to an equivalent 1-way nondeterministic version, with an exponential blow-up:

Theorem 14 ([Vardi, 1998])**.** *Let $\mathcal{A}$ be a $\mathsf{2APT}^\omega$. We can construct in* EXPTIME *a $\mathsf{1NPT}^\omega$ $\mathcal{A}'$ such that $L(\mathcal{A}) = L(\mathcal{A}')$. The number of states of $\mathcal{A}'$ is exponential*

in the number of states of $\mathcal{A}$, and the number of priorities of $\mathcal{A}'$ is linear in the number of states and priorities of $\mathcal{A}$.

One reason this result is interesting is that we can test for non-emptiness of a $\mathsf{1NPT}^\omega$:

Proposition 17. *[Kupferman and Vardi, 1998] There is an algorithm that tests for emptiness of a $\mathsf{1NPT}^\omega$ in time $O(poly(s)2^{poly(p)})$, where s is the size of the automaton and p is the number of priorities.*

Given an automaton A over trees whose nodes are labeled with unary predicates $S_0 \ldots S_n$, accepting language L_A. The projection of L_A by predicate S_0 is the set of trees t labeled with $S_1 \ldots S_n$ such that for some expansion t' of t with an interpretation for S_0, t' is in L_A. We say a class of automata is "closed under projection" if given an automaton A in the class as above and a predicate S_0, there is another automaton in the class accepting the projection of L_A by S_0. The notion of a class being closed under complement, intersection, union, is similar. By combining this conversion to a nondeterministic automaton with the straightforward closure of nondeterministic automata under projection, we get closure of $\mathsf{2APT}^\omega$ under projection.

Proposition 18. *$\mathsf{2APT}^\omega$ automata are closed under projection. The construction can be done in $\mathsf{EXPTIME}$ and results in an automaton for the projection language where the number of states is exponential in the original number of states, and the number of priorities is linear in the original number of states and priorities.*

Thus alternating automata with the parity condition are closed under all the operations used not only in first-order logic but in Monadic Second Order Logic. Given $\mathsf{2APT}^\omega$ automata A_1 representing φ_1 and A_2 representing φ_2, we can construct an automaton representing $\varphi_1 \wedge \varphi_2$: just create a new start state which has transition function that conjoins the transitions out of the start states of A_1 and A_2. Similarly for $\varphi_1 \vee \varphi_2$. If we have an automaton A for $\varphi(S_0 \ldots)$, we can construct an automaton for $\exists S\ \varphi$ by just projecting A by S_0. These closure properties can thus be used to convert any MSOL sentence over trees into a $\mathsf{1NPT}^\omega$, which can be tested for non-emptiness using Proposition 17, giving a proof of decidability of MSO over trees, Theorem 2.

Example 18. *We form an automaton checking whether the formula $\exists x\ C(x) \wedge D(x)$ holds on a tree using the method above. We first form an automaton A_C checking the formula $C(x)$. This will be an automaton over trees with labels C, D and Isx, where Isx represents the node x. A_C will be the "conjunction" of two automata. The first automaton searches the tree for a node with labels A and Isx, while the second checks that there is exactly one node labelled Isx. By conjunction here we mean that we use the closure of conjunction of alternating automata, described above. The automaton for A_D is defined analogously. We then form an automaton $A_{C(x)\wedge D(x)}$ checking the formula $C(x) \wedge D(x)$ by applying closure under conjunction again. The final automaton we want is formed by projecting*

the automaton $A_{C(x)\wedge D(x)}$ under the predicate Isx*, which we can do while staying within alternating automata by Proposition 18.*

However, iteratively applying Theorem 14 leads to a tower of exponential blow-up. Thus if we want to get a better bound for a logic like GNF, we will need to do something better than just applying these operations naïvely.

We need one final conversion, going in this case in the other direction, from 1-way nondeterministic to 2-way alternating automata. We call this *localization*. This process takes a 1-way nondeterministic automaton $\mathcal{A}$ that runs on trees with extra information about some predicate (annotated on the tree), and converts it to an equivalent 2-way alternating automaton $\mathcal{A}'$ (the "locally-launched automaton" for $\mathcal{A}$) that operates on trees without these annotations, but is launched from an interior node v. The automaton $\mathcal{A}'$ launched at v should verify that $\mathcal{A}$ accepts on the tree in which only v is annotated with the additional predicate.

Theorem 15. *Let $\Sigma' := \Sigma \cup \{U_1, \ldots, U_j\}$. Let $\mathcal{A}'$ be a $\mathsf{1NPT}^\omega$ on Σ'-trees. We can construct a $\mathsf{2APT}^\omega$ $\mathcal{A}$ on Σ-trees such that for all Σ-trees t and nodes v in the domain of t,*

$$\mathcal{A}' \text{ accepts } t' \text{ from the root } \textit{iff } \mathcal{A} \text{ accepts } t \text{ from } v$$

where t' is the Σ'-tree obtained from t by setting $U_1, \ldots, U_j$ to hold at v (and only at v).

The number of states of $\mathcal{A}$ is linear in the number of states of $\mathcal{A}'$, the number of priorities of $\mathcal{A}$ is linear in the number of priorities of $\mathcal{A}'$, and the overall size of $\mathcal{A}$ is linear in the size of $\mathcal{A}'$.

Proof sketch. The subtlety is that the automaton $\mathcal{A}$ is reading a tree without the singleton valuations for $U_1, \ldots, U_j$ so once the automaton leaves node v, if it were to cross this node again, it would be unable to correctly simulate $\mathcal{A}'$. To deal with this, $\mathcal{A}$ simulates $\mathcal{A}'$ by concurrently doing two things:

- guessing the state $\mathcal{A}'$ would reach at the launch node v, and simulating what $\mathcal{A}'$ does moving downward from v
- guessing in a backwards fashion an initial part of a run of $\mathcal{A}'$ on the path from v to the root and then processing "the rest of the tree" – the part outside of the path from the root to v and the subtree of v, in a normal downwards fashion. When the automaton moves upward it can remember which child it came from, so as to avoid going down this path again.

□

It can shown that this localization process even preserves *monotonicity*. This is not important at the moment, but will play a role later. For now, just note that an automaton $\mathcal{A}$ is *monotonic with respect to* Σ'' if for all states q in the state set of $\mathcal{A}$ and for all $\tau' \subseteq \{P : P \in \Sigma''\}$, $\delta(q, \tau)$ entails $\delta(q, \tau \cup \tau')$. In other

words, monotonicity says that if the automaton is able to do something without τ' in the label, then it should be able to do it with τ' in the label.

It can be checked that if $\mathcal{A}$ is monotonic with respect to Σ'', then its localization $\mathcal{A}'$ is also monotonic with respect to Σ''; this follows from the fact that the transition function of $\mathcal{A}'$ is designed to imitate a run of $\mathcal{A}$.

7.4 Inductive translation of GNF to an automaton

One approach to translating GNF to automata is to work inductively, getting an automaton for each subformula, and using closure properties of automata in building up the automata bottom-up. In the previous subsection, we mentioned that this is the approach used to show decidability of full MSO over trees, but using this method (on arbitrary MSO) gives a non-elementary algorithm, since with every quantifier alternation we will need to apply some automaton operation that induces an exponential blow-up.

Outline of an approach via Scott Normal Form. One way to approach this is by rewriting the GNF formulas to reduce quantifier alternation. This is the approach taken in the original GNF paper [Bárány et al., 2011]. The idea is to introduce new relation symbols for intermediate formulas, an approach that is often called "Scott Normal Form", after its usage in earlier decidable logics by Dana Scott [Scott, 1962].

The Scott Normal Form of a GNF formula in normal form will be constructed as the conjunction of two formulas, the "skeleton" and the conjunction of another set of formulas, the "definitions", both created by induction. The skeleton will always be a UCQ, while the definitions will be a formula with one quantifier alternation. For atomic formulas the skeleton is the formula itself, and the definitions are empty. For positive boolean operations the skeletons are just take the corresponding combination of the inductively-defined formulas, and the definitions are just the definitions that we get by induction.

Suppose we have a UCQ-shaped formula $\chi = \delta[Y_1 := \alpha_1 \wedge \neg\psi_1, \ldots, Y_s := \alpha_s \wedge \neg\psi_s]$ where δ is a UCQ, then the skeleton of χ is just δ. Let ψ_i' be the skeleton of ψ_i formed by induction. Then the definitions of χ are all the definitions of the ψ_i, along with the definitions:

$$\forall \vec{x}\ Y_i(\vec{x}) \rightarrow (\alpha_i(\vec{x}) \wedge \neg\ \psi_i')$$
$$\forall \vec{x}\ \alpha_i(\vec{x}) \wedge \neg\ \psi'(\vec{x}) \rightarrow\ Y_i(\vec{x})$$

Note that the skeleton and each definition formed by this conversion is a GNF formula, and thus the conjunction of the skeleton and each definition is in GNF.

Example 19. *Let* path3(x_1, x_2) *be the formula*

$$\exists y_1\ y_2\ y_3 R(x_1, y_1) \wedge R(y_1, y_2) \wedge R(y_2, y_3) \wedge R(y_3, x_2)$$

So path3 *says there is a path with 3 intermediate nodes leading from* x_1 *to* x_2.

Consider the formula

$$\exists x_1\ x_2\ x_3\ x_4(S(x_1,x_2) \wedge \neg \mathsf{path3}(x_1,x_2)) \wedge (T(x_2,x_3) \wedge \neg \mathsf{path3}(x_2,x_3))$$

Then the Scott Normal Form of the formula is:

$$\begin{aligned}
&(\exists x_1\ x_2\ x_3\ x_4(G_1(x_1,x_2) \wedge G_2(x_2,x_3)) \wedge \\
&(\forall x_1\ x_2\ G_1(x_1,x_2) \leftarrow S(x_1,x_2) \wedge \mathsf{path3}(x_1,x_2)) \wedge \\
&(\forall x_1\ x_2\ S(x_1,x_2) \wedge \mathsf{path3}(x_1,x_2) \rightarrow G_1(x_1,x_2)) \wedge \\
&(\forall x_1\ x_2\ G_2(x_1,x_2) \leftarrow T(x_1,x_2) \wedge \mathsf{path3}(x_1,x_2)) \wedge \\
&(\forall x_1\ x_2\ T(x_1,x_2) \wedge \mathsf{path3}(x_1,x_2) \rightarrow G_2(x_1,x_2))
\end{aligned}$$

The key point about the Scott normal form is that the quantifier alternation is bounded: if put in prenex normal form, the formula we be a conjunction of $\forall \vec{x}\ \exists y\ \gamma$ where γ is quantifier-free. When we translate these formulas to codes, we will then get a restricted quantifier alternation within MSO. And after that, a standard algorithm for converting MSO formulas to automata will give us an elementary blow-up.

Inductive construction without Scott normal form. Above we sketched an approach that went first into the normal form of GNF, to simplify the grammar, and then into Scott Normal Form, to get rid of quantifier alternation. We now give another inductive construction, which does not explicitly go through Scott Normal Form, but uses the same idea of breaking up a GNF formula into UCQs and GF formulas. The idea will be that the only expensive operation in alternating automata is projection, and projection only occurs only in dealing with UCQ-shaped formulas. There is an exponential blow up in converting a UCQ-shaped formula to an automaton. But this blow-up is *only in terms of the "UCQ-skeleton"* (described below): there is no blow-up in terms of the size of the automata for the lower-level guarded negation formulas that are embedded in the UCQ-shaped formula. Hence if we have many levels of nesting of UCQ-shaped subformulas, we only get a single exponential blow-up, not a tower of exponentials. We make this more precise below.

Recall that in translating from GF to automata, we only had to deal with *answer-guarded subformulas*: formulas φ that are logically equivalent to $R(\vec{x}) \wedge \varphi$ where $R(\vec{x})$ is a guard for the free variables in φ. Since every subformula $\varphi(\vec{x})$ is only used within a guard for $\vec{x}$, we only needed to care about answer-guarded subformulas. This meant that we could restrict our attention to bindings of the free variables that lie within a single node of the tree. The ability to deal only with "local properties" was crucial to the construction, since it meant we could inductively create automata that run from an interior node in the tree that has all the bound values in it. In contrast, GNF is built up from formulas that may not be guarded, and we have to decide how to deal with these.

To get around the need to worry about free variable assignments that are not local, we can make use of the normal form for GNF. We take advantage of the fact that the nesting of UCQs allowed in GNF is restricted: the free variables in

a nested UCQ-shaped formula can be assumed to be guarded, and hence must be represented locally in a single node in the tree code.

Recall that a UCQ-shaped formula in GNF with negation depth greater than 0 is of the form $\delta[Y_1 := \alpha_1 \wedge \neg\psi_1, \ldots, Y_s := \alpha_s \wedge \neg\psi_s]$ where δ is a UCQ over the extended signature σ' obtained from σ by adding fresh relations $Y_1, \ldots, Y_s$, each ψ_i is a UCQ-shaped formula in GNF, and each α_i is a guard in σ for the free variables of ψ_i. We proceed by first constructing an automaton for the UCQ "skeleton" δ, over the extended signature σ'. The automaton for $\delta[Y_1 := \alpha_1 \wedge \neg\psi_1, \ldots, Y_s := \alpha_s \wedge \neg\psi_s]$ can then be obtained by "plugging in" the complement of the inductively-defined automata for each ψ_i into the automaton for δ. The closure properties of parity automata are used to perform the complementation efficiently.

Less formally this construction amounts to simulating the automaton for δ while allowing Eve to guess the valuations for each Y_i (i.e. valuations for each subformula $\alpha_i \wedge \neg\psi_i$). In order to prevent Eve from cheating and just guessing that every tuple satisfies these subformulas, Adam is allowed to challenge her on these guesses by launching an inductively defined automaton for these subformulas.

The only remaining issue is how to efficiently create an automaton for the UCQ skeleton. That is, we need to say how to get an automaton for a UCQ δ with guarded free variables. Formally, a *local assignment* $\vec{a}/\vec{x}$ for $\vec{a} = a_1 \ldots a_n \in N_k^n$ and $\vec{x} = x_1 \ldots x_n$ is a mapping such that $x_i \mapsto a_i$. A node v in a valid tree code t with $\vec{a} \subseteq \text{names}(v)$ and a local assignment $\vec{a}/\vec{x}$ describes a valuation for $\vec{x}$. We want an automaton A_δ whose states include, as one component in a product, a local assignment σ. The correctness condition is that A_δ runs on t from a state that is associated with σ accepts if and only if δ holds in $\text{decode}(t)$ on the valuation associated with σ. This is basically the same correctness condition as we used for translating GF to automata. But the automata we created for GF did not need the additional "local assignment" to be explicitly a separate component of the state, since for GF our states were formulas of the closure, and these local assignments were already baked into the closure.

Returning to how to construct A_δ: Intuitively we can just "program the automaton for A_δ directly". We create an automaton that starts at an interior node with an initial state telling how the free variables of δ map to local names of the node. We then search the tree, guessing witnesses to the existential quantifiers and verify each atomic fact.

One way to formalize this is to explicitly describe the states of the automata. The states will have to encode information about which variables we currently have in scope and which subformula of the UCQ we are currently trying to verify. In the next subsection we will define a notion called "specialization" that will help us describe these states.

An alternative to explicitly writing out the states of A_δ will be to make use of localization. The idea is to first write a 1-way non-deterministic automaton for δ that assumes that all the variables are marked on the tree. This can be done for a UCQ by induction, using the closure properties of 1-way non-deterministic automata. We can apply Theorem 15 to convert this 1-way automaton over the

extended signature with free variable markings to the 2-way automaton that we need in order to do the "plugging operation" above. Note that Theorem 15 applies to a UCQ, since such formulas are monotone. This idea of starting with an automaton that runs on trees with free variables and then localizing it when we are able to get to an answer-guarded formula will be useful in dealing with more powerful logics in Section 11 later on.

Using our construction for UCQ-shaped formula we can create the desired automaton for GNF by induction on the negation depth of the formula.

Lemma 4. *Given a normal form formula $\psi(\vec{x})$ in* GNF *of width at most k over signature σ, we can construct a* 2APT$^\omega$ *$\mathcal{A}_\psi$ such that for all valid k tree codes t, for all local assignments $\vec{a}/\vec{x}$, and for all nodes v in t with $\vec{a} \subseteq \text{names}(v)$,*

$$\mathcal{A}_\psi^{\vec{a}/\vec{x}} \text{ accepts } t \text{ starting from } v \quad \textit{iff} \quad \text{decode}(T) \text{ satisfies } \psi([v, \vec{a}]).$$

Further, there is a polynomial function f independent of ψ such that the number of states of $\mathcal{A}_\psi$ is at most $N_\psi := f(m_\psi) \cdot 2^{f(kr_\psi)}$ and the number of priorities is at most $f(km_\psi)$ where $m_\psi = |\psi|$, r_ψ is the CQ-rank of ψ. The overall size of the automaton and the running time of the construction is at most exponential in $|\sigma| \cdot N_\psi$.

By combining this with an automaton that checks that the input tree is a valid k tree code, we can test satisfiability of sentences from GNF in 2EXPTIME.

Theorem 16. *There is a* 2EXPTIME *procedure that decides satisfiability of* GNF *formulas.*

With a little more effort, we could even show that we can construct a 2ABT$^\omega$ for ψ in GNF rather than a parity automaton, but this is not important for achieving the 2EXPTIME bound on satisfiability.

7.5 "Holistic" translation from GNF to automata

Recall that in the GF automaton construction, we can explicitly say what the states of the automaton are: they correspond to formulas in the subformula closure. The same was true in the earlier μ-calculus approach. This "holistic approach" is different from the inductive approach that we just overviewed for GNF, which applies automata operations to get the final automaton.

We can also extend the holistic approach to GNF, giving an alternative translation. In doing so we need to include more than just the usual subformula closure in order to be able to correctly handle the CQ-shaped formulas. Below we will explain this, focusing only on GNF without equality.

Before we define the relevant closure, we need to think more carefully about CQ-shaped formulas, and how they can be satisfied in a tree-like structure. This fine-grained analysis of how these formulas can be satisfied could be used as an alternative way to get automata for UCQ-skeletons, as in the approach in the previous section, but we will use it here as a way of describing the entire automata for a GNF sentence directly. Our analysis of how CQ-shaped formula

can be satisfied revolves around the notion of *specialization*, which we describe next.

Consider a CQ-shaped normal form GNF formula

$$\rho(\vec{x}) = \exists \vec{y} \bigwedge_{j \in \{1,\ldots,r\}} \psi_j(\vec{x}, \vec{y}).$$

A *specialization* of ρ is a formula ρ' obtained from ρ by the following operations:

- select a subset $\vec{y}_0$ of $\vec{y}$ (call variables from $\vec{x} \cup \vec{y}_0$ the *inside variables* and variables from $\vec{y} \setminus \vec{y}_0$ the *outside variables*);
- select a partition $\vec{y}_1, \ldots, \vec{y}_s$ of the outside variables, with the property that for every ψ_j, either ψ_j has no outside variables or all of its outside variables are contained in the partition element $\vec{y}_j$;
- let χ_0 be the conjunction of the ψ_j whose free variables are among $\vec{x}$ and the inside variables $\vec{y}_0$, and let χ_i for $i \in \{1, \ldots, s\}$ be the conjunction of the ψ_j using outside variables and satisfying $\mathsf{Free}(\psi_j) \subseteq \vec{x} \cup \vec{y}_0 \cup \vec{y}_i$;
- set $\rho'(\vec{x}, \vec{y}_0)$ to be

$$\chi_0(\vec{x}, \vec{y}_0) \wedge \bigwedge_{i \in \{1,\ldots,s\}} \exists \vec{y}_i \ \chi_i(\vec{x}, \vec{y}_0, \vec{y}_i).$$

Roughly speaking, each specialization of ρ describes a different way that a CQ-shaped formula could be satisfied by elements $\vec{x}$ occurring in a bag of a tree-like structure. The inside variables represent witnesses for the existential quantifiers that are found in the bag itself. The partition of the outside variables represent the different directions from the bag where the additional non-local witnesses are to be found: moving either to an ancestor or to one of the children. Since each atom of the CQ shape formula must be realized in a single bag, the atoms must be "homogeneous" with respect to the partition, as captured in the second item above.

It is easy to see that if a specialization is realized, then so is the original formula, since the realization of the specialization gives witnesses for all the existential quantifiers:

Lemma 5. *Let $\rho(\vec{x}) \in$ GNF be a CQ-shaped formula $\exists \vec{y} \ \bigwedge_j \psi_j(\vec{x}, \vec{y})$. For all structures M and for all specializations $\rho'(\vec{x}, \vec{y}_0)$ of ρ, if $M \models \rho'(\vec{a}, \vec{b})$, then $M \models \rho(\vec{a})$.*

Since a formula is vacuously a specialization of itself, the converse direction is vacuously true. What is more useful is that whenever a formula is realized, it is realized by a specialization that is "simpler" than the original formula it specializes. We say a specialization is *non-trivial* if either χ_0 is non-empty or the partition of the outside variables is non-trivial ($s > 1$). The following result captures the idea that in realizing a formula we need to realize some simpler specialization:

Lemma 6. *Let $\rho(\vec{x}) \in \mathsf{GNF}$ be a CQ-shaped formula $\exists \vec{y} \bigwedge_j \psi_j(\vec{x}, \vec{y})$. Given a structure M and its tree code $\mathfrak{t}$, if there exists a node v that includes names $\vec{a}$ and $M \models \rho([v, \vec{a}])$, then there is a non-trivial specialization $\rho'(\vec{x}, \vec{y}_0)$ of ρ and a node w with $\vec{a}$ and additional names $\vec{b}_0$ in its domain such that $[w, \vec{a}] = [v, \vec{a}]$ and $M \models \rho'([w, \vec{a}], [w, \vec{b}_0])$.*

The idea behind the lemma is that if the formula holds at a bag with certain witnesses for the free variables, we can traverse the bags of the tree code, traversing only nodes that preserve all those witnesses, until we arrive at a bag B where either some of the witnesses are found locally in B or the witnesses are found in different directions from B. In the first case we have realized a specialization in which χ_0 is non-empty, and in the second case we have realized a specialization in which the partition of the outside variables is non-trivial.

We will write $\mathsf{Spec}(\exists \vec{y}\, \eta(\vec{a}, \vec{y}), A)$ for the set of all specializations of $\exists \vec{y}\, \eta(\vec{a}, \vec{y})$ with names from A substituted for any new inside variables. For convenience in the construction below, each specialization $S \in \mathsf{Spec}(\exists \vec{y} \bigwedge_j \psi_j(\vec{x}, \vec{y}), A)$ will be represented as a set: that is, the specialization $\chi_0(\vec{a}, \vec{b}_0) \wedge \bigwedge_{i \in \{1,\ldots,s\}} \exists \vec{y}_i\, \chi_i(\vec{a}, \vec{b}_0, \vec{y}_i)$ of $\exists \vec{y} \bigwedge_{j \in \{1,\ldots,r\}} \psi_j(\vec{a}, \vec{y})$ is represented as the set:

$$\{\psi_j(\vec{a}, \vec{b}_0) : j \in \{1, \ldots, r\}, \psi_j(\vec{a}, \vec{b}_0) \text{ in } \chi_0\} \cup \{\exists \vec{y}_i\ \chi_i(\vec{a}, \vec{b}_0, \vec{y}_i) : i \in \{1, \ldots, s\}\}.$$

Again, each formula in the set describes how a piece of the CQ-shaped formula is satisfied.

Fix some GNF sentence φ in normal form that we are interested in testing satisfiability for. The closure $\mathsf{cl}_{\mathrm{GN}}(\varphi, N_k)$ that is relevant for the GNF automaton construction consists of the subformulas of φ along with specializations of any CQ-shaped formulas. Formally, elements of $\mathsf{cl}_{\mathrm{GN}}(\varphi)$ will be of the form $\langle \psi, p \rangle$ where ψ is a formula and $p \in \{+, -\}$ is a polarity to indicate whether ψ comes from a positive or negative part of φ.

Let $\mathsf{cl}_{\mathrm{GN}}(\varphi, N_k)$ be the smallest set C of formulas containing $\langle \varphi, + \rangle$, $\langle \mathsf{True}, + \rangle$, $\langle \mathsf{True}, - \rangle$, $\langle \mathsf{False}, + \rangle$, $\langle \mathsf{False}, - \rangle$ and satisfying the following closure conditions:

- if $\langle \alpha \wedge \neg\psi, + \rangle \in C$, then $\langle \alpha, + \rangle, \langle \psi, - \rangle \in C$;
- if $\langle \alpha \wedge \neg\psi, - \rangle \in C$, then $\langle \alpha, - \rangle, \langle \psi, + \rangle \in C$;
- if $\langle \bigvee_i \psi_i, + \rangle \in C$, then $\langle \psi_i, + \rangle \in C$ for all i;
- if $\langle \bigvee_i \psi_i, - \rangle \in C$, then $\langle \psi_i, - \rangle \in C$ for all i;
- if $\langle \exists \vec{y}\, \eta(\vec{a}, \vec{y}), + \rangle \in C$, then $\langle \psi', + \rangle \in C$ for all $\psi' \in \mathsf{Spec}(\exists \vec{y}\, \eta(\vec{a}, \vec{y}), N_k)$;
- if $\langle \exists \vec{y}\, \eta(\vec{a}, \vec{y}), - \rangle \in C$, then $\langle \psi', - \rangle \in C$ for all $\psi' \in \mathsf{Spec}(\exists \vec{y}\, \eta(\vec{a}, \vec{y}), N_k)$.

We are now ready to describe the $\mathsf{2ABT}^{\omega}$ automaton construction for a GNF sentence φ. The automaton $\mathcal{A}_\varphi$ for φ is defined as follows:

- The state set is $\mathsf{cl}_{\mathrm{GN}}(\varphi, N_k)$.

- The initial state is $\langle \varphi, + \rangle$.
- The transition function δ is defined below.
- The set of accepting states consists of all states of the form $\langle \mathsf{True}, + \rangle$, $\langle \mathsf{False}, - \rangle$, $\langle R(\vec{a}), - \rangle$, or $\langle \exists \vec{y}\ \eta(\vec{a}, \vec{y}), - \rangle$.

We now describe the transition function. Below we use Dir to abbreviate the set $\mathsf{Direction}_2$ of movement actions in a binary tree. That is, Dir consists of Stay, $\mathsf{Down}_1, \mathsf{Down}_2$, and Up.

$$\delta(\langle R(\vec{a}), + \rangle, \tau) := \begin{cases} (\mathsf{Stay}, \langle \mathsf{False}, + \rangle) & \text{if } \vec{a} \text{ not represented in } \tau \\ (\mathsf{Stay}, \langle \mathsf{True}, + \rangle) & \text{if } R(\vec{a}) \in \tau \\ \bigvee_{d \in \mathsf{Dir}} (d, \langle R(\vec{a}), + \rangle) & \text{otherwise} \end{cases}$$

$$\delta(\langle R(\vec{a}), - \rangle, \tau) := \begin{cases} (\mathsf{Stay}, \langle \mathsf{True}, + \rangle) & \text{if } \vec{a} \text{ not represented in } \tau \\ (\mathsf{Stay}, \langle \mathsf{False}, + \rangle) & \text{if } R(\vec{a}) \in \tau \\ \bigwedge_{d \in \mathsf{Dir}} (d, \langle R(\vec{a}), - \rangle) & \text{otherwise} \end{cases}$$

$$\delta(\langle \mathsf{True}, + \rangle, \tau) := (\mathsf{Stay}, \langle \mathsf{True}, + \rangle)$$
$$\delta(\langle \mathsf{False}, - \rangle, \tau) := (\mathsf{Stay}, \langle \mathsf{False}, - \rangle)$$
$$\delta(\langle \mathsf{True}, - \rangle, \tau) := (\mathsf{Stay}, \langle \mathsf{True}, - \rangle)$$
$$\delta(\langle \mathsf{False}, + \rangle, \tau) := (\mathsf{Stay}, \langle \mathsf{False}, + \rangle)$$
$$\delta(\langle \bigvee_i \psi_i, + \rangle, \tau) := \bigvee_i (\mathsf{Stay}, \langle \psi_i, + \rangle)$$
$$\delta(\langle \bigvee_i \psi_i, - \rangle, \tau) := \bigwedge_i (\mathsf{Stay}, \langle \psi_i, - \rangle)$$
$$\delta(\langle \alpha \wedge \neg \psi, + \rangle, \tau) := (\mathsf{Stay}, \langle \alpha, + \rangle) \wedge (\mathsf{Stay}, \langle \psi, - \rangle)$$
$$\delta(\langle \alpha \wedge \neg \psi, - \rangle, \tau) := (\mathsf{Stay}, \langle \alpha, - \rangle) \vee (\mathsf{Stay}, \langle \psi, + \rangle)$$

$$\delta(\langle \exists \vec{y}\ \eta(\vec{a}, \vec{y}), + \rangle, \tau) := \begin{cases} (\mathsf{Stay}, \langle \mathsf{False}, + \rangle) & \text{if } \vec{a} \text{ not represented in } \tau \\ \bigvee_{S \in \mathsf{Spec}(\exists \vec{y}\ \eta(\vec{a}, \vec{y}), \mathrm{names}(\tau))} \bigwedge_{\psi \in S} (\mathsf{Stay}, \langle \psi, + \rangle) \vee \\ \quad \bigvee_{d \in \mathsf{Dir}} (d, \langle \exists \vec{y}\ \eta(\vec{a}, \vec{y}), + \rangle) & \text{otherwise} \end{cases}$$

$$\delta(\langle \exists \vec{y}\ \eta(\vec{a}, \vec{y}), - \rangle, \tau) := \begin{cases} (\mathsf{Stay}, \langle \mathsf{True}, + \rangle) & \text{if } \vec{a} \text{ not represented in } \tau \\ \bigwedge_{S \in \mathsf{Spec}(\exists \vec{y}\ \eta(\vec{a}, \vec{y}), \mathrm{names}(\tau))} \bigvee_{\psi \in S} (\mathsf{Stay}, \langle \psi, - \rangle) \wedge \\ \quad \bigwedge_{d \in \mathsf{Dir}} (d, \langle \exists \vec{y}\ \eta(\vec{a}, \vec{y}), - \rangle) & \text{otherwise} \end{cases}$$

We can show that:

> A formula $\varphi(\vec{x})$ holds in a tree code t at node v with names $\vec{a}$ if and only if the automaton above accepts when launched at t from v with initial state $\langle \varphi(\vec{a}), + \rangle$.

This is proven by induction, and Lemma 6 is utilized in the inductive case for CQ-shaped formulas.

It is useful to compare this to the GF construction on page 44. There is an obvious difference here in that we are carrying around the polarity in the state, whereas with GF we were working with formulas in NNF. Ignoring this

minor technical difference, the construction here is the same as for GF except for the quantifier case. For GF, we could always find the witnesses for the quantified variables in a single node (because the quantification was guarded). For GNF, the witnesses for a CQ-shaped formula might not be in a single node. Hence, Eve guesses some specialization, which is a declaration of the how to find witnesses—some of these witnesses might be in the current node, but some of the witnesses might be in different parts of the tree, and require breaking-up the CQ into pieces that are satisfied in these different directions. Once she declares this specialization, Adam can then challenge her on any part of this specialization. Notice that any state $\langle \exists \vec{y}\ \eta(\vec{a}, \vec{y}), + \rangle$ is not accepting, to ensure that Eve cannot forever avoid finding the witnesses.

Because the number of specializations is exponential in the size of the formula (in fact polynomial in the size of the signature, and exponential in the width and CQ-rank), the number of states is still singly-exponential as before. Hence, this still leads to a 2EXPTIME satisfiability testing algorithm for GNF.

7.6 Bibliographic remarks and suggestions for further reading

The Guarded Negation Fragment was introduced in [Bárány et al., 2011], which also presented the main decidability and complexity results about the logic. The best reference for the language is probably [Bárány et al., 2015], although the syntax we used has some minor distinctions from that presented in [Bárány et al., 2015]. Follow-up papers studying further properties of the logic are [Bárány et al., 2012, Bárány et al., 2013]. The decidability argument for GNF given in [Bárány et al., 2015] proceeds differently than the direct automata-theoretic approaches given here. They first eliminate nesting of CQ-shaped formulas by introducing new relations. This simplification reduces to the case of a boolean combination of GF sentences and UCQs: essentially, the Open World Query Answering problem for GF sentences. This special case can be solved by a more straightforward automata-theoretic translation [Bárány et al., 2010].

An important result about GNF is that it has the finite model property: whenever a sentence is satisfiable, there is a finite structure that satisfies it. This result, due to Bárány, Gottlob, and Otto [Bárány et al., 2010], makes use of automata-theoretic techniques, but uses other ideas for avoiding the use of infinite structures.

Frontier-guarded TGDs were studied in [Baget et al., 2011b], and the results on open world query answering for these TGDs were derived in [Baget et al., 2011b].

8 Applications to Open World Query Answering

We have seen that the Guarded Negation Fragment contains allows us to combine GF statements with positive existential sentences in a decidable logic. In this section, we give an application of GNF decidability to the open world query answering problem, a problem coming from databases and knowledge representation. For this problem the ability to express positive existential sentences is crucial.

Open World Query answering will also introduce us to two ways of refining the technique of tree-like models. We will look at situations where we can come up with a single "most general" tree-like model. Open World Query Answering also gives us a more refined complexity analysis for GNF, by splitting the input up into two parts.

8.1 Open World Query Answering Basics

Recall from Section 2 that first-order logic formulas are evaluated over a structure M, consisting of a domain and interpretations of each relation. But if we are interested in formulas φ that are domain-independent, we need not concern ourselves with the domain, and can consider the structures M to be defined just by the interpretations of the relations.

In some situations we have a set of information, but it is incomplete. For example, we may have a ternary relation $\mathsf{MarriotHotels}(id, address, phone)$ and a relation $\mathsf{Hotels}(address, phone)$. We know about some information concerning it; for example, we may know $\mathsf{MarriotHotels}(2456, 123\text{ Main Street}, 222-2222)$ holds, and we know some logical rules satisfied by the data, such as:

$$\forall xyz\ \mathsf{MarriotHotels}(x, y, z) \rightarrow \mathsf{Hotels}(y, z)$$

But additional facts may hold as well. Given a finite structure M_0, and a set of sentences Σ in some logic, the *possible worlds for* M_0, Σ are the set of structures M such that M satisfies Σ and the interpretation of each relation R in M contains the interpretation in M_0. We say "M contains the facts of M_0" below. We can consider M_0 and Σ as a way of representing the infinitely many possible worlds.

Given another sentence φ, we want to know if it holds in all possible worlds for M_0, Σ. In the example above, $\mathsf{Hotels}(123\text{ Main Street}, 222-2222)$ holds in all possible worlds.

Formally, the *Open World Query Answering Problem* (OWQ) is the problem that takes as input a finite structure M_0, sentences Σ, a sentence φ, and outputs yes exactly when φ holds in all possible worlds for M_0, Σ. We write $M_0 \wedge \Sigma \models \varphi$ for short.

8.2 Decidability of Open World Query Answering

An easy corollary of our results on GNF is that the Open World Query Answering Problem is decidable for Σ in GF and φ a UCQ.

Corollary 1. *Open World Query Answering is decidable in* 2EXPTIME *as* M_0 *varies over finite structures,* Σ *over* GNF*,* φ *over UCQs.*

Proof. The negation of OWQ is the satisfiability of $\bigwedge_{F \in M_0} F \wedge \Sigma \wedge \neg\varphi$, where F ranges over quantifier-free atomic formulas $R(c_1 \dots c_n)$ that hold in M_0. This is in GNF, and thus we can decide it using Theorem 16. □

8.3 Open World Query Answering and Canonical tree-like models

The OWQ problem has most often been studied in the case where the sentences Σ are Tuple-Generating Dependencies (TGDs), sentences of the form:

$$\forall x_1 \dots x_j \bigwedge_{m \leq q} A_m \rightarrow$$
$$\exists y_1 \dots y_k \bigwedge_{n \leq r} B_n$$

where $A_1 \dots A_q$ and $B_1 \dots B_r$ are relational atoms. The conjunction of A_m is referred to as the *body* of the TGD while the conjunction of the B_n is referred to as the *head*. The variables x_i that also occur in some B_n are the *exported variables* of the TGD.

Corollary 1 tells us that OWQ is decidable when φ is a UCQ and the TGDs are in GNF. A TGD is *Guarded* if there is an atom A_g in the body that contains all the variables of the body. A TGD is *Frontier-Guarded* if there is an atom A_g in the body that contains all the exported variables. We claim that Frontier-Guarded TGDs can be rewritten in GNF. This can be seen by rewriting universal quantification and implication using negation and existential quantification in the usual way:

$$\neg[\exists x_1 \dots x_j \bigwedge_{m \leq q} A_m \wedge$$
$$\neg \exists y_1 \dots y_k \bigwedge_{n \leq r} B_n]$$

and then observing that the outer negation is vacuously guarded (since there are no free variables), while the distinguished atom A_g serves as a guard for the inner negation. Thus OWQ is decidable for Frontier-Guarded TGDs. Note that Guarded TGDs can be rewritten in GF. However, applying the reduction of Corollary 1, we see that OWQ for Guarded TGDs reduces to a GNF satisfiability problem (not a GF satisfiability problem): this is because OWQ involves the CQs, which are not generally in GF. However, what we can say using the same reduction as in Corollary 1 is that OWQ for an atomic CQ and Guarded TGDs reduces to satisfiability of GF.

It follows that if Σ consists of Frontier-Guarded TGDs, M_0 is a finite structure, Q is a CQ, and $M_0 \wedge \Sigma \wedge \neg Q$ is satisfiable, then there is an M that witnesses

this that has a k-tree code: that is, an M that has a k-tree code that preserves all the facts of M_0, satisfies Σ and fails to satisfy Q.

In fact, for any set of TGDs Σ, there is a "canonical choice" for such an M, and the same choice of M works for every CQ Q such that $M_0 \wedge \Sigma \wedge \neg Q$ is satisfiable. We can build such a structure as the union of structures M^i with $M^0 = M_0$ an M^{i+1} built from M^i by "firing each rule in M^{i+1}".

Formally, for each TGD $\sigma \in \Sigma$

$$\forall x_1 \dots x_j \bigwedge_{m \leq q} A_m \rightarrow$$
$$\exists y_1 \dots y_k \bigwedge_{n \leq r} B_n$$

A *trigger* for σ is a substitution $c_1 \dots c_j$ such that $M^i, c_1 \dots c_j \models \bigwedge_m A_m$. "Firing a rule on this trigger" (often called "performing a chase step with this trigger") means generating new elements $d_1 \dots d_k$ and adding facts $B_n(d_1 \dots d_k)$ for $n \leq r$ to M^{i+1}. The structure M^{i+1} is formed from M^i by performing chase steps for each trigger in M^i.

The structure constructed by unioning the structures M_i define above is often referred to as the *chase of* M_0 *under* Σ, denoted $\mathsf{Chase}_\Sigma(M_0)$. [1]

Example 20. *Consider* Σ *consisting of the following two TGDs:*

$$\forall x\ y\ [R(x,y) \wedge U(y) \rightarrow \exists z\ R(y,z) \wedge U(z) \wedge V(y)]$$
$$\forall x\ y\ [R(x,y) \wedge V(y) \rightarrow \exists z\ R(z,x) \wedge V(x)]$$

and the initial structure M_0 *in which the only ground atoms holding are* $R(x_0, y_0)$ *and* $U(y_0)$ *for some elements* x_0, y_0.

In the first "chase step" we would fire the first rule with the "trigger" $x = x_0, y = y_0$. *This creates a new element* z_0 *and the additional facts*

$$R(y_0, z_0), U(z_0), V(y_0)$$

After this step, another chase step could fire on the second rule, with trigger $x = x_0, y = y_0$. *This step creates a new element* z_1 *and adds facts*

$$R(z_1, x_0), V(x_0)$$

At this point, we have fired every chase step in M_0, *giving us a new structure* M^1. *We should now fire every chase step in* M^1. *For example, rule* 1 *can then fire on* $x = y_0, y = z_0$, *and rule* 2 *can fire on* $x = z_1\ y = x_0$.

The process will go on for an infinite number of rounds, leading to an infinite structure, the union of the structures M^i *formed in each round, in which the sentences in* Σ *are satisfied.*

[1] There are several variations of the chase [Fagin et al., 2005, Onet, 2013], but the other variations will not be relevant here.

M' clearly satisfies Σ and is thus a possible world for M_0. Further we claim that:

Proposition 19. *For any boolean conjunctive query Q with constants from M_0, $M_0 \wedge \Sigma \models Q$ exactly when Q holds in the chase of M_0 under Σ.*

Proof. It is easy to see that $\mathsf{Chase}_\Sigma(M_0)$ contains M_0 and satisfies Σ. That is, $\mathsf{Chase}_\Sigma(M_0)$ is a possible world for M_0, Σ. Thus if $M_0 \wedge \Sigma \models Q$, then Q holds in $\mathsf{Chase}_\Sigma(M_0)$. In the other direction, suppose Q holds in $\mathsf{Chase}_\Sigma(M_0)$, and consider an arbitrary possible world M. We will show that Q holds in M. To do this we will build a homomorphism h from $\mathsf{Chase}_\Sigma(M_0)$ to M. We build h inductively, as the union of homomorphisms h^i on each M^i. For elements of M_0 it is the identity The new element of M^{i+1} are generated from a chase step using some trigger τ in M^i for some TGD. That is there is a TGD

$$\forall x_1 \dots x_j \bigwedge_{m \leq q} A_m \rightarrow \exists y_1 \dots y_k \bigwedge_{n \leq r} B_n$$

in Σ, and a $c_1 \dots c_j \in M^i$ satisfying the left hand side of the TGD, with new elements $m_1 \dots m_k$ generated as witnesses for $y_1 \dots y_k$ in the head. The homomorphism h^i maps $c_1 \dots c_j$ to some trigger in M. But since M satisfies Σ, there must be some witnesses $d_1 \dots d_k$ for $y_1 \dots y_k$ in M. We set $h(m_i) = d_i$ for each i. If we do this for each fresh elements generated in M^{i+1} we can check that this is a homomorphism. □

Example 21. *An illustration of the proof of Proposition 19 can be found in Figure 9. The figure shows the first few levels of $\mathsf{Chase}_\Sigma(M_0)$ on the left, and then an arbitrary possible world M on the right. The colors represent unary relations and the edges a binary relation. The figure highlights the difference between $\mathsf{Chase}_\Sigma(M_0)$ and an arbitrary possible world: in the chase the witnesses needed for some constraint are fresh elements. In M we must still have such a witness, but it may not be a fresh element. The homomorphism just maps the fresh elements in $\mathsf{Chase}_\Sigma(M_0)$ to an arbitrary witness in M.*

A structure M that satisfies the conclusion of Proposition 19 is called a *universal model* for Σ and M_0. The relationship between an arbitrary possible world M for M_0 and $\mathsf{Chase}_\Sigma(M_0)$ has some resemblance to the relationship between an arbitrary structure M and its unravelling (see Figure 3). In both cases, there is a relationship between M and the other structure that can map a single element of M to many elements in the other structure. But the chase is defined for TGDs, which are incomparable in expressiveness with guarded logics. And for arbitrary TGDs that are not guarded or Frontier-Guarded, the resulting structure may not be tree-like. However, regardless of tree-likeness, for any collection of TGDs Σ, the chase is always a "canonical counterexample" for the OWQ problem. This is what Proposition 19 says.

We claim that if Σ consists of Frontier-Guarded TGDs, then for any finite M_0, the chase of M_0 is tree-like: that is, it has a k-code, where k is bounded by the size of M_0 plus the size of Σ. The root vertex is associated with all elements

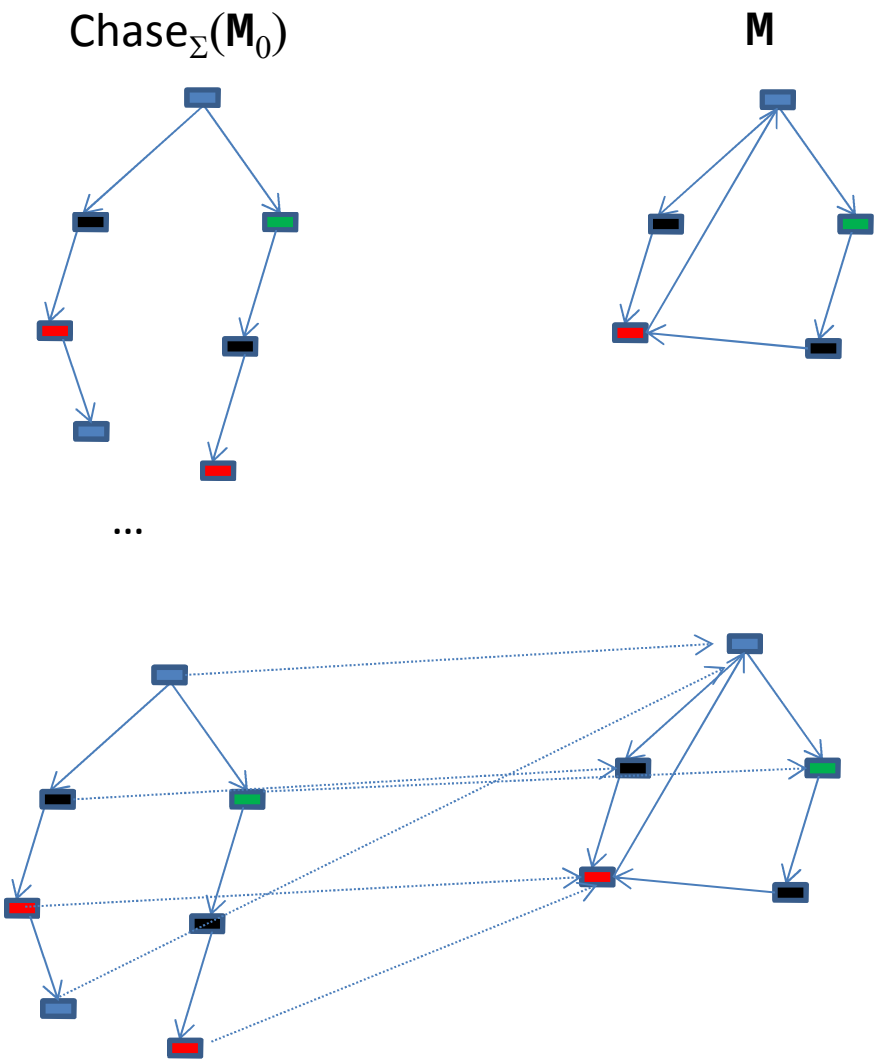

Figure 9: The chase, an arbitrary possible world M, and a homomorphism from the chase to M

of M_0. The other vertices are associated with a rule firing f in the formation of M'. If v_f is the vertex corresponding to f, then $\lambda(v)$ associates v to every element that is generated in firing f. Each rule firing f is associated to some fact $R(e_1 \dots e_k)$ that guards the mapping to the rule body, and there is a rule firing f' that generated this fact; we make v_f the child of v'_f. We need to check that:

- For every fact $F = R(c_1 \dots c_k)$ in M, the set of arguments $\{c_1 \dots c_k\}$ is contained in $\lambda(v)$ for some v
- For every element $e \in M$, the set of vertices v associated with e is connected.

For the first, note that the arguments for a fact F of M_0 are contained in the root. The arguments for a fact generated by a firing f are contained in the corresponding vertex v_f.

Example 22. *Recall the TGDs Σ of Example 20, and the initial structure M_0 in which the only ground atoms holding are $R(x_0, y_0)$ and $U(y_0)$ for some elements x_0, y_0.*

Figure 10 shows the firing of rules on the initial structure M_0, and the grouping of the resulting structure into a tree code. Note that the root consists of the facts in M_0 itself. All other nodes represent structures with at most w elements, where w is the width of Σ. In fact, since these are Guarded TGDs, we obtain that all non-root nodes are guarded.

For the second item, it suffices to show that if e is not present in $\lambda(v)$ for a vertex v, it is not present in $\lambda(v')$ for any descendant v' of v. But the only elements that are introduced in a descendant of v are elements generated freshly by rule firings, and these can not be in v since they are fresh.

Note that gives an alternative proof that counterexamples to OWQ for Frontier-Guarded TGDs can be made tree-like, without going through Theorem 13, the tree-like model result for GNF. This gives a direct argument for decidability of OWQ for Frontier-Guarded TGDs.

In fact, one can go further and devise a 2EXPTIME algorithm to conclude decidability of OWQ for Guarded TGDs directly, without referring to automata at all. This is what is accomplished in [Calì et al., 2008, 2012]. The idea is that one can truncate the chase process any time a chase step produces a bag that is "similar" to an ancestor bag. We explain the idea for the simple case of $\mathsf{OWQ}(M_0, \Sigma, Q)$ where Σ consists of guarded TGDs and Q consists of a single ground fact $G()$. Let M be the chase of M_0 under Σ, with n, n' nodes in the tree-like representation. A *local homomorphism* from n to n' is a mapping from the values occurring in n to the values occurring in n' that preserves all facts of n. We say that a node n is *ancestor-blocked* if it has some strict ancestor n' where there is a local homomorphism from n to n'. Let M^- by a finite partial chase under Σ whose tree representation has the property that *every leaf node is ancestor-blocked*. That is, from each leaf node n of M^- there is a homomorphism of n to some strict ancestor n' of n in M^-.

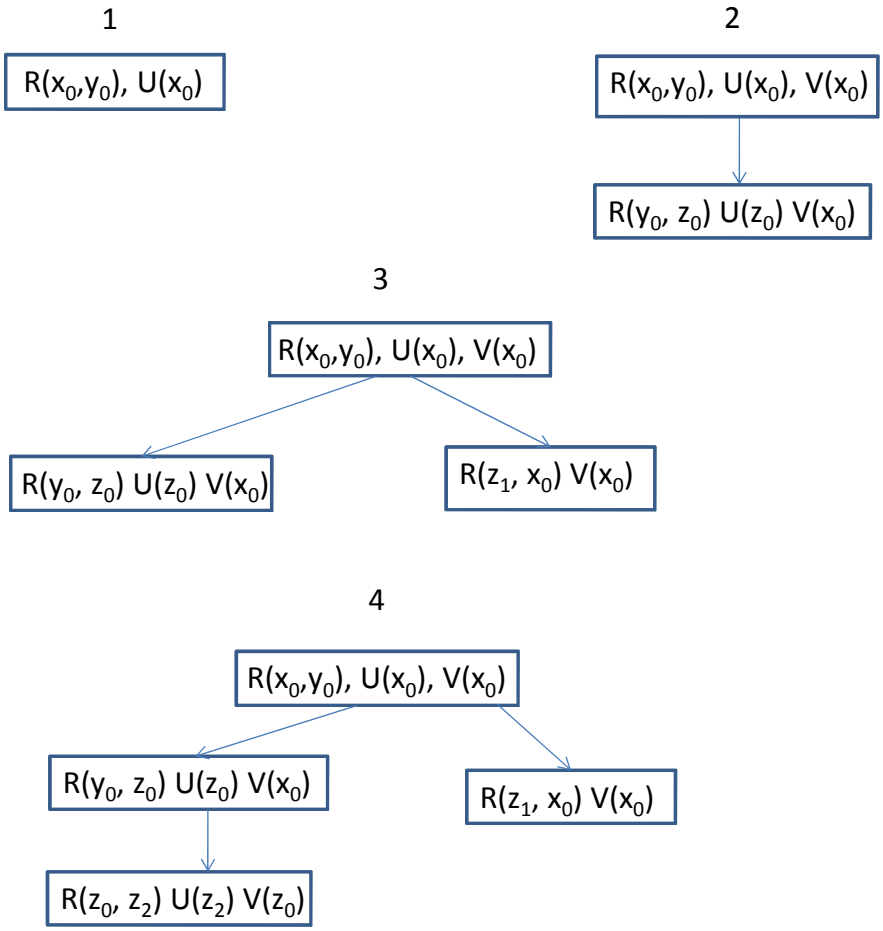

Figure 10: The chase process with guarded TGDs, with each step creating a tree-like structure

We claim that such an M^- is already universal for atomic queries. That is, if a boolean atomic query $G()$ does not hold in M^-, then it cannot hold in M, and hence we cannot have $M_0 \wedge \Sigma \models G()$. The reason is simple: if $G()$ held in M, this would be witnessed in some node m of M. Taking such a node m of minimal depth, it would have an ancestor in M^-, say n. But letting p be the path from n to m in M, a homomorphic image of p must exist going from n' to some node m' satisfying $G()$ in M, since every trigger that can fire from n, its homomorphic image will fire from n'. Thus we have contradicted the minimality of m.

Now we note that there is a double-exponential function f of M_0 and Σ such that: any path of size at least f has a node that is ancestor-blocked. Thus within M there is an M^- of depth at most doubly-exponential. Using this fact, it is not difficult to obtain a doubly-exponential complexity bound for OWQ for atomic queries under Guarded TGDs. The same technique generalizes to arbitrary conjunctive queries [Calì et al., 2012, 2008], and to Frontier-Guarded TGDs [Baget et al., 2011b].

Although this direct method can be seen as alternative presentations without automata, the notions of "similar bag" could be seen as just special cases of state repetition in automata.

8.4 Refining the complexity of open world query answering using tree-like models

Recall that open world query answering problem has 3 inputs: an initial structure M_0, a set of sentences Σ and a sentence Q. If Q is a conjunctive query and Σ is in GNF, our satisfiabiilty procedures based on tree-like models give us a way of deciding OWQ. But tree-like models also give us information about how the complexity behaves when Σ and Q are fixed and only M_0 varies. This is the "data complexity of OWQ" and it is significant because one expects Σ and Q to be small (e.g. a set of sentences that a person could write down) while M may be a very large dataset.

Theorem 17. *For any fixed set of* GNF *sentences* Σ *and UCQ* Q*, there is a* CoNP *algorithm that takes a finite structure* M_0 *and decides* $\mathsf{OWQ}(M_0, \Sigma, Q)$.

A *counterexample to* $\mathsf{OWQ}(M_0, \Sigma, Q)$ is a possible world M' for M_0 and Σ that satisfies $\neg Q$. Clearly $\mathsf{OWQ}(M_0, \Sigma, Q)$ holds exactly when there is no counterexample. To prove the theorem, we need a refinement of the unravelling process that unravels a counterexample M' but leaves M_0 in tact. For a finite structure M_0 an M_0, k*-rooted structure* is one which is M_0 unioned with a structure $T_{\vec{c}}$ for $\vec{c} \in \mathsf{domain}(M_0)^k$ where the domain of $T_{\vec{c}}$ overlaps with the domain of M only in $\vec{c}$, and for two k-tuples $\vec{c}$ and $\vec{c}'$, the domain of $T_{\vec{c}}$ overlaps with the domain of $T_{\vec{c}'}$ only within $\vec{c} \cap \vec{c}'$.

One can picture such a set as a squid, with M at the root and the $T_{\vec{c}}$ hanging off as tentacles. See Figure 8.4. We can show that if there is a witness to satisfiability of a GNF sentence in a structure that contains the facts of M_0, then there is such a structure that forms an M_0, k-rooted structure.

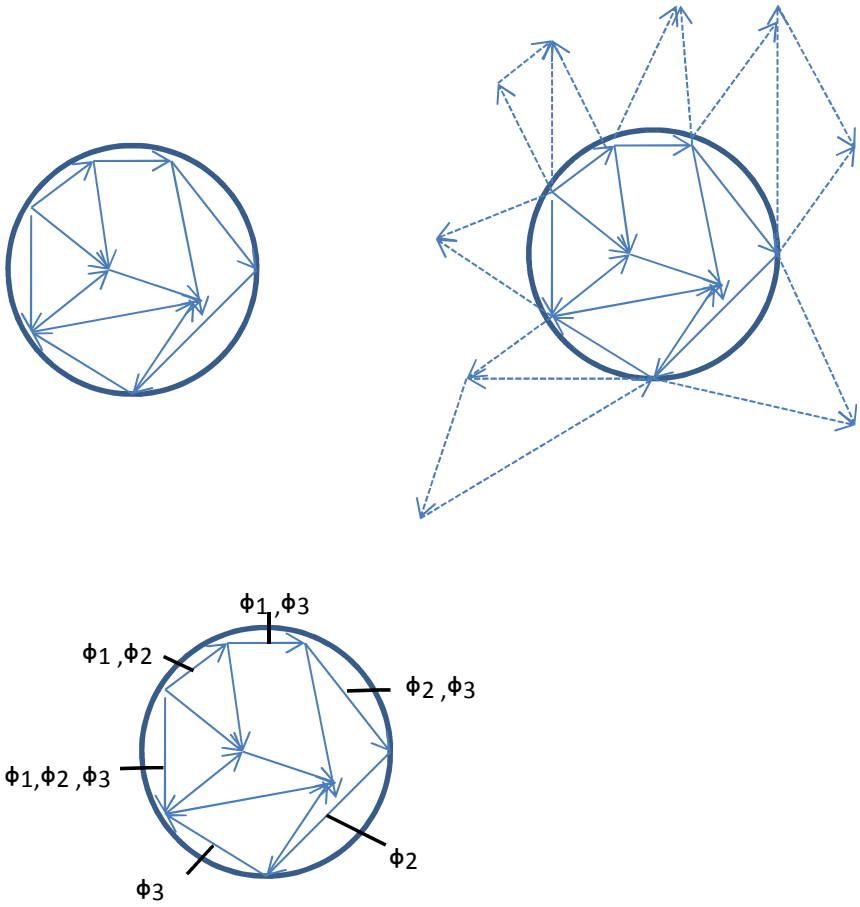

Figure 11: A structure M_0 (left top), an $M_0, 2$-rooted structure (right top) , and the abstraction of the extension (bottom)

Proposition 20. *For any finite structure M_0, if a* GNF *sentence Σ is satisfiable by a structure in which the interpretation of each relation R contains the interpretation of R in M_0, then Σ is satisfied in a structure which has an M_0, k-rooted structure, where k is at most $|\Sigma|$.*

Proof. One proof of Proposition 20 proceeds by taking a counterexample M and modifying the unravelling of M for GNF to preserve M_0. That is, we define the unravelling as before, but we make an exception for the root, making the root M_0. Recall that this procedure produces a tree-like structure where the bags correspond to k-tuples from M, for k the width of Σ. Thus the modified unravelling procedure will produce a structure that is structured in a tree, where the root is M_0 and all other nodes correspond to k-tuples. The distinct subtrees of the root would form the structures $T_{\vec{c}}$.

There is also a simpler construction that gives an M_0, k-rooted counterexample, without invoking the unravelling construction. For each guarded set $\vec{c}$ of M, let $T_{\vec{c}}$ be a copy of M where all elements are fresh except for $\vec{c}$. Let $M' = \bigcup_{\vec{c}} T_{\vec{c}}$. Clearly $T_{\vec{c}}$ and $T_{\vec{c}'}$ only overlap within $\vec{c} \cap \vec{c}'$. We claim that Duplicator has a winning strategy in the GNF^k game between M and M', for *any* k. This will imply that M' is still a counterexample to OWQ. Note that every element v of M' is associated with an element $\rho(v)$ of M In our strategy for Duplicator will ensure that if the active structure is M', then the homomorphism h will just be ρ. If the active structure is M, the homomorphism h will just be the identity. One can easily check that this determines a successful strategy for Duplicator. □

Proposition 20 shows that it suffices to examine witnesses consisting of a root that is a copy of M_0 and a collection of tentacles indexed by k-tuples of the domain of the root. We have control over the size of the root, and also over the index set. But the size of the tree-like tentacles is unbounded. We now show a decomposition result stating that to know what happens in a tree-like set, we will not need to care about the details of the tentacles, but only a small amount of information concerning the sentences that the tentacle satisfies in isolation.

Recall that $\mathsf{FO}(\sigma)$ denotes first-order logic over the signature σ with equality. Let $\mathsf{FO}(\sigma \cup \{d_1 \dots d_k\})$ denote first-order logic over the signature σ with equality and k constants, which will be used to represent the overlap elements.

Recall that the quantifier-rank of a formula is the maximal number of nested quantifiers. For any fixed signature ρ, if we fix the quantifier-rank j, we also fix the number of variables that may occur in a formula, and thus there are only finitely many sentences up to logical equivalence. Thus we can let $\mathsf{FO}_j(\rho)$ denote a finite set containing a sentence equivalent to each sentence of quantifier-rank at most j. Given an M_0, k-rooted structure M, and number j, the *j-abstraction of M* is the expansion of M_0 with relations $P_\tau(x_1 \dots x_k)$ for each $\tau \in \mathsf{FO}_j(\sigma \cup \{d_1 \dots d_k\})$. We interpret $P_\tau(x_1 \dots x_k)$ by the set of k-tuples $\vec{c}$ such that $T_{\vec{c}}$ satisfies τ.

That is, a j-abstraction of an M_0, k-rooted structure M is an annotation of each k-tuple $\vec{c}$ in M_0 with some formulas summarizing the structure of the

tentacle $T_{\vec{c}}$ of M. The bottom of Figure 8.4 gives the idea of a j-abstraction of the $M_0, 2$-rooted structure M shown in the top right. Each 2-tuple of the rooted structure M is annotated by the formulas $\varphi_1, \varphi_2, \varphi_3 \ldots$ that hold in the corresponding tentacle of M.

We let $\sigma_{j,k}$ be the signature of the j-abstraction of such structures. We now present a lemma capturing the idea that we can reduce reasoning about M_0, k rooted structures M to reasoning about the j-abstractions of such extensions.

Lemma 7. *For any sentence φ of $\mathsf{FO}(\sigma)$ and any $k \in \mathbb{N}$, there is a $j \in \mathbb{N}$ having the following property:*

Let N_1 be an M_1, k-rooted structure for some σ-structure M_1. Let N_2 be an M_2, k-rooted structure for some σ-structure M_2. If the j-abstractions of N_1 and N_2 agree on all $\mathsf{FO}(\sigma_{j,k})$ sentences of quantifier-rank at most j, then N_1 and N_2 agree on φ.

Proof. To prove this we need to make use of some results on games for first-order logic.

Let j_φ be the quantifier-rank of φ. We choose $j := j_\varphi \cdot k$. We will show that N_1 and N_2 agree on all formulas of quantifier-rank j_φ. Recall now Proposition 1 in Section 2: to show that N_1 and N_2 agree on these formulas, it is sufficient to give a strategy for Duplicator in the j_φ-round standard pebble game for $\mathsf{FO}(\sigma)$ over N_1 and N_2. With i moves left to play, we will ensure the following invariants on a game position consisting of a sequence $\vec{p}_1 \in N_1$ and $\vec{p}_2 \in N_2$:

- Let $\vec{p_1}'$ be the subsequence of $\vec{p}_1$ that comes from M_1 and let $\vec{p_2}'$ be defined similarly for $\vec{p}_2$ and M_2. Then $\vec{p_1}'$ and $\vec{p_2}'$ should form a winning position for Duplicator in the $(i \cdot k)$-round $\mathsf{FO}(\sigma_{j,k})$ game on the j-abstractions.
- Fix any k-tuple $\vec{c}_1 \in M_1$ and let $P^1_{\vec{c}_1}$ be the subsequence of $\vec{p}_1$ that lies in $T_{\vec{c}_1}$ within N_1. Then if $P^1_{\vec{c}}$ is non-empty, $\vec{c}_1$ also lies in $\vec{p}_1$. Further, letting $\vec{c}_2$ be the corresponding k-tuple to $\vec{c}_1$ in $\vec{p}_2$, and letting $P^2_{\vec{c}_2}$ be the subsequence of $\vec{p}_2$ that lies in $T_{\vec{c}_2}$ within N_2, then $P^1_{\vec{c}_1}$ and $P^2_{\vec{c}_2}$ form a winning position in the i-round pebble game on $T_{\vec{c}_1}$ and $T_{\vec{c}_2}$.

  The analogous property holds for any k-tuple $\vec{c}_2 \in M_2$.

We now explain the strategy of Duplicator, focusing for simplicity on moves of Spoiler within N_1, with the strategy on N_2 being similar. If Spoiler plays within M_1, Duplicator responds using her strategy for the games on the j-abstractions of M_1 and M_2. It is easy to see that the invariant is preserved.

If Spoiler plays an element within a substructure $T_{\vec{c}_1}$ within N_1 that is already inhabited, then by the inductive invariant, $\vec{c}_1$ is pebbled and there is a corresponding $\vec{c}_2$ in N_2 with substructure $T_{\vec{c}_2}$ of N_2 such that the pebbles within $T_{\vec{c}_2}$ are winning positions in the game on $T_{\vec{c}_1}$ and $T_{\vec{c}_2}$ with i moves left to play. Thus Duplicator can respond using the strategy in this game from those positions.

Now suppose Spoiler plays an element e_1 within a substructure $T_{\vec{c}_1}$ within N_1 that is not already inhabited. We first use $\vec{c}_1$ as a sequence of plays for

Spoiler in the game on the j-abstractions of N_1 and N_2, extending the positions given by $\vec{p_1}'$ and $\vec{p_2}'$. By the inductive invariant, responses of Duplicator exist, and we collect them to get a tuple $\vec{c}_2$. Since a winning strategy in a game preserves atoms, and we have a fact in the j-abstraction corresponding to the j-type of $\vec{c}_1$ in $T_{\vec{c}_1}$, we know that $\vec{c}_2$ must satisfy the same j-type in $T_{\vec{c}_2}$ that $\vec{c}_1$ does in $T_{\vec{c}_1}$. Therefore $\vec{c}_1$ must satisfy the same $\mathsf{FO}(\sigma \cup \{d_1 \dots d_k\})$ sentences of quantifier-rank at most j in $T_{\vec{c}_1}$ as $\vec{c}_2$ does in $T_{\vec{c}_2}$. Thus Duplicator can use the corresponding strategy to respond to e_1 with an e_2 in $T_{\vec{c}_2}$ such that $\{e_1\}$ and $\{e_2\}$ are a winning position in the $(i-1)$-round pebble game on $T_{\vec{c}_1}$ and $T_{\vec{c}_2}$.

Since the response of Duplicator corresponds to k moves in the game within the j-abstractions, one can verify that the invariant is preserved.

We must verify that this strategy gives a partial isomorphism. Consider a fact F that holds of a tuple $\vec{t}_1$ within N_1, and let $\vec{t}_2$ be the tuple obtained using this strategy in N_2.

- If $\vec{t}_1$ lies completely within some $T_{\vec{c}_1}$, then the last invariant guarantees that $\vec{t}_2$ lies in some $T_{\vec{c}_2}$. The last invariant also guarantees that σ-facts of N_1 are preserved since such facts must lie in $T_{\vec{c}_1}$, and the corresponding positions are winning in the game between $T_{\vec{c}_1}$ and $T_{\vec{c}_2}$.
- If $\vec{t}_1$ lies completely within M_1, then the first invariant guarantees that the fact is preserved.

By the definition of a rooted structure, the above two cases are exhaustive. □

From Lemma 7 we easily obtain:

Corollary 2. *For any sentence φ and $k \in \mathbb{N}$, there is $j \in \mathbb{N}$ and a sentence φ' in the language $\sigma_{j,k}$ of j-abstractions over σ such that for all sets of σ-facts M_0, an M_0, k-rooted structure satisfies φ iff its j-abstraction satisfies φ'.*

Recall that by Proposition 20, we know it suffices to check for a counterexample to entailment that is an M_0, k-rooted structure. Corollary 2 allows us to do this by guessing an abstraction and checking a first-order property of it. This allows us to finish the proof of Theorem 17.

Proof of Theorem 17. Fixing Q and Σ, we give an NP algorithm for the complement. Let $\varphi = \Sigma \wedge \neg Q$, and $k = |\varphi|$. Let j and φ' be the number and formula guaranteed for φ by Corollary 2.

Recall that $\mathsf{FO}(\sigma \cup \{d_1 \dots d_k\})$ denotes first-order logic over the signature σ of $\Sigma \wedge \neg Q$ with equality and with k constants, and $\mathsf{FO}_j(\sigma \cup \{d_1 \dots d_k\})$ denotes a finite set containing a sentence equivalent to each sentence of quantifier-rank at most j. Let Types_j be the collection of subsets τ of $\mathsf{FO}_j(\sigma \cup \{d_1 \dots d_k\})$ sentences such that the conjunction of sentences in τ is satisfiable. Note that for any fixed j, the size of $\mathsf{FO}_j(\sigma \cup \{d_1 \dots d_k\})$ is finite, hence the size of Types_j is finite. An element of Types_j can be thought of as a description of a tentacle, telling us everything we need to know for the purposes of the j-abstraction.

Given M_0, guess a function f mapping each k-tuple over M_0 to a $\rho \in \mathsf{Types}_j$. We then check whether for two overlapping k-tuples $\vec{c}$ and $\vec{c}'$, the types $f(\vec{c})$ and

$f(\vec{c}')$ are consistent on the atomic formulas that hold on overlapping elements, and whether the atomic formulas of $f(\vec{c})$ contain each fact over $\vec{c}$ in M_0. Finally, for each $\tau \in \mathsf{FO}(\sigma \cup \{d_1 \dots d_k\})$ of quantifier-rank at most j, we form the expansion I by interpreting P_τ by the set of tuples $\vec{c}$ such that $\tau \in f(\vec{c})$, and we check whether the expansion satisfies φ' with these interpretations, and if so return true.

We argue for correctness. If the algorithm returns true with I as the witness, then create an M_0, k-rooted structure N by picking for each $\vec{c}$ a structure satisfying the sentences in $f(\vec{c})$ with distinguished elements interpreted by $\vec{c}$. Such a structure exists by satisfiability of $f(\vec{c})$. It assigns atomic formulas consistently on overlapping tuples, by hypothesis, and the atomic formulas it assigns contain each fact of M_0. We let the remaining domain elements be disjoint from the domain of M_0. Note that by construction, N has M as its j-abstraction. By the choice of j and φ', and the observation above, N satisfies $\Sigma \wedge \neg Q$. Thus N witnesses that $\mathsf{OWQ}(M_0, \Sigma, Q)$ is false.

On the other hand, if $\mathsf{OWQ}(M_0, \Sigma, Q)$ is false, then by Proposition 20 we have an M_0, k-rooted structure N that satisfies $\Sigma \wedge \neg Q$. By the choice of j and φ', the j-abstraction of N satisfies φ'. For each tuple $\vec{c}$ from M_0, the set of formulas of quantifier-rank holding of $\vec{c}$ in the tentacle of $\vec{c}$ must be in Types_j. Hence we can guess f that assigns $\vec{c}$ to this set, and with this f as a witness the algorithm returns true. □

8.5 Variations of OWQ decidable via Guarded Logics

A variation of the OWQ problem is the *hybrid open and closed world query answering* problem, which we denote as HOCWQ. Here the inputs are a finite structure M_0, a set of sentences Σ, a formula Q, and additionally a subset C of the signature of the relations used in Σ: the *closed relations*. Recall that in OWQ we consider as possible worlds for M_0 and Σ structures in which every relation is interpreted by a superset of the interpretation in M_0. In hybrid open and closed world query answering, we consider only extensions in which the closed relations are interpreted *exactly* as in M_0.

The *possible worlds for* M_0, Σ, C are the set of structures M such that M satisfies Σ, the interpretation of each relation R in M contains the interpretation in M_0 and for each relation $R \in C$, the interpretation of R in M is the same as the interpretation of R in $M_0>$ Given a sentence φ, we say that $\mathsf{HOCWQ}(M_0, \Sigma, C, \varphi)$ is true exactly when φ holds in all possible worlds.

Example 23. *Consider the following sentences* Σ*:*

$$\forall x\ y\ U(x) \rightarrow \exists y\ T(x, y)$$
$$\forall x\ y\ T(x, y) \rightarrow U(y)$$

Consider a structure M_0 *with only a single fact,* $U(0)$*. Suppose that we are interested in the query* $Q = \exists x\ T(x, x)$*.*

The possible worlds for Open World Querying are any structures M where $U(0)$ holds and Σ holds. There are many such worlds where Q fails: for example we could have M containing $U(0), T(0,1)$. Thus $\mathsf{OWQ}(M_0, \Sigma, Q)$ is false.

On the other hand, suppose we consider hybrid open and closed world querying, with U a closed relation. Then in any possible world M we must have U containing only 0. By the first sentence in Σ, M must contain $T(0, y_0)$ for some y_0. But by the second constraint and the closedness of U, we must have $y_0 = 0$. Thus we can see that $\mathsf{HOCWQ}(M_0, \Sigma, Q)$ is true.

Analogously to what we did with OWQ, We can see that HOCWQ is decidable when the constraints are in GNF.

Theorem 18. *We can decide $\mathsf{HOCWQ}(M_0, \Sigma, Q, C)$ where Σ ranges over GNF sentences and Q over boolean conjunctive queries.*

Proof. $\mathsf{HOCWQ}(M_0, \Sigma, Q, C)$ can be restated as unsatisfiability of the following sentence $\psi_{M_0,\Sigma,Q,C}$

$$\bigwedge_{F \in M_0} F \wedge \Sigma \wedge \neg Q \wedge \bigwedge_{U \in C} \forall \vec{x}\ [U(\vec{x}) \rightarrow \bigvee_{\vec{c} \in M_0(U)} \vec{x} = \vec{c}]$$

If U is a relation of arity n and $\vec{x}, \vec{c}$ are n-tuples of variables and constants, respectively, then $\vec{x} = \vec{c}$ is an abbreviation for a conjunction $\bigwedge_{i \leq n} x_i = c_i$

We claim that for any M_0, Σ, Q, and C, with Σ in GNF and Q a boolean CQ, $\psi_{M_0,\Sigma,Q,C}$ is expressible in GNF. $\bigwedge_{F \in M_0} \wedge \Sigma \wedge \neg Q$ is clearly in GNF, since Σ is in GNF, all ground sentences and boolean conjunctive queries are in GNF, and GNF is closed under conjunction. The final conjunct is a conjunction over all $U \in C$. Fix a closed relation U, let n be the arity of U, and consider the formula:

$$\forall x_1 \ldots x_n\ [U(x_1 \ldots x_n) \rightarrow \bigvee_{c_1 \ldots c_n \in M_0(U)} \bigwedge_{i \leq n} x_i = c_i]$$

We can rewrite this as $\psi'_{M_0,\Sigma,Q,C}$

$$\neg \exists x_1 \ldots x_n\ [U(x_1 \ldots x_n) \wedge \neg \bigwedge_{c_1 \ldots c_n \in M_0(U)} \bigvee_{i \leq n} \neg x_i = c_i]$$

Each formula $\neg x_i = c_i$ is in GNF, since it has only one free variable x_i. Thus the formula

$$\bigwedge_{c_1 \ldots c_n \in M_0(U)} \bigvee_{i \leq n} \neg x_i = c_i]$$

is in GNF, since GNF is closed under disjunction and conjunction. We can then conclude that:

$$U(x_1 \ldots x_n) \wedge \neg \bigwedge_{c_1 \ldots c_n \in M_0(U)} \bigvee_{i \leq n} \neg x_i = c_i]$$

is in GNF, since the negation is guarded by $U(x_1 \ldots x_n)$. Finally we can conclude that $\psi'_{M_0,\Sigma,Q,C}$ is in GNF, since GNF is closed under existential quantification

and negation of sentences. Since $\psi'_{M_0,\Sigma,Q,C}$ is equivalent to $\psi_{M_0,\Sigma,Q,C}$, this completes the argument.

Thus we can check satisfiability of $\psi_{M_0,\Sigma,Q,C}$ in 2EXPTIME using the GNF satisfiability procedure of Theorem 16. □

8.6 Further reading and bibliographic remarks

Open world query answering appears under many names. In database theory it is often referred to as the "certain answer problem" [Fagin et al., 2005]. Work in database theory has focused on the case of TGDs. There are decidability results for special classes of TGDs based on "acyclicity conditions" in [Fagin et al., 2005]. Decidability for guarded TGDs was shown in [Calì et al., 2008, 2012], and for frontier-guarded TGDs in [Baget et al., 2011b]. The direct argument for decidability of OWQ for Guarded TGDs using the tree-like structure of the chase is given in [Calì et al., 2008]. The papers [Calì et al., 2008, 2012] analyze the complexity of OWQ for Guarded TGDs and several related classes, using the analysis of repetitions in the chase, rather than automata. The chase can be seen as a special case of the tableau proof systems that are heavily studied in classical proof theory [D'Agostino et al., 2001] and in Description Logics [Baader and Sattler, 2001]. The idea of "cutting off a proof when it repeats" described for the chase is closely related to the idea of *blocking* used in tableau algorithms for Description logics [Baader and Sattler, 2001].

The application of GNF to OWQ was outlined in the papers where GNF was introduced [Bárány et al., 2011, Bárány et al., 2015]. The application was extended in [Bárány et al., 2012], which introduces other connections between GNF and database query languages. For example [Bárány et al., 2012] gives a fragment of the database language Relational Algebra that is equivalent in expressiveness to GNF. The analysis of the data complexity of OWQ for GNF given in this section derives from [Bárány et al., 2012] as well. The data complexity analysis is closely-related to the "composition method" for analyzing logics over trees [Thomas, 1997b, Rabinovich, 2005]. The extension to Hybrid Open and Closed World Query Answering is based on [Benedikt et al., 2016a].

9 Fixpoint Logics

Our prior examples of decidable logics have all remained within first-order logic. We now consider extensions with fixpoints, a feature beyond first-order logic.

9.1 GNFP basics

Recall that least fixpoint logic, LFP, extends first-order logic by having:

- additional atomic formulas, $X(t_1 \dots t_k)$ where X is a second-order variable of arity k
- a new way to form formulas inductively, using a least fixpoint operator

Recall also that a formula defined using a fixpoint is true exactly when one of its approximants is true.

Example 24. *Let our vocabulary consist of a graph relation $G(x, y)$ and unary relation $U(x)$. We can consider the following* LFP *formula $\varphi(x)$:*

$$\mu_{X,x}.U(x) \vee \exists y(G(x, y) \wedge X(y))](z)$$

This defines all the elements that can reach an element in U using a path of G-edges.

Let us look at the set of approximations to the fixpoint in this formula. The initial approximation, denoted φ_0 is formed by setting $X = \emptyset$ in the body of the fixpoint. This gives the set of elements in U itself.

The next approximation φ_1 is formed by setting $X = \varphi_0$: this gives the set of elements that reach U in one step.

Similarly we can see that the n^{th} approximation is the set of elements reaching U in n-steps. φ represents the union of all these approximations.

The least fixpoint operator thus represents a kind of infinite disjunction. We can use it to derive a corresponding infinitary conjunction. We can add to LFP the new formation rule:

If $\varphi \in$ LFP and φ contains a second-order variable X of arity k along with k first-order variables $\vec{x}$, and X only occurs positively in φ then:

$$[\nu_{X,\vec{x}}.\varphi(\vec{x}, X, \vec{Y})](\vec{t})$$

is a formula of LFP.

To evaluate a greatest fixpoint on a structure M, we start with X being *all tuples from* M, and then iterate until a fixpoint is reached. That is we can define a *decreasing* sequence of approximations: start with φ_0 being all tuples from M, form $\varphi_{\alpha+1}$ by setting X to be φ_α, and then intersect at limit ordinals.

The greatest fixpoint operator does not add any expressiveness to LFP; it can be seen as an abbreviation for

$$\neg[\mu_{X,\vec{x}}.\varphi(\vec{x}, \neg X, , \vec{Y})](\vec{t})$$

Example 25. *We give an example of what can be expressed by alternating greatest and least fixpoints.*

Let $ReachesU(z)$ be the formula from Example 24 specifying that z reaches a U element. Consider the formula

$$[\nu_{Y,x}.ReachesU(x) \wedge (\forall y\ G(x,y) \rightarrow Y(y))](x)$$

To understand its meaning, consider the first approximation of it. That is, what happens if Y is all values. Then the second conjunct is always true, and thus the approximation specifies elements that reach an element in U via a G path. In the second approximation, we set Y to be the first approximation, and obtain the elements that reach a U element, and all their G-successors reach a U element. Iterating this reasoning, we see that the n^{th} approximation represents elements e such that: every element e' reachable in n steps from them has a path to a U element. Since a greatest fixpoint is a conjunction of its approximants, the formula represents elements e such that every element e' reachable from e has a path to a U element.

A natural example of LFP formulas comes from *Datalog*, a fixpoint logic in which formulas are written as a collection of rules. In Datalog, relation symbols are divided up into *extensional* and *intensional* relations, and a rule is of the form:

$$R(\vec{x}) := \varphi$$

where $R(\vec{x})$ is an atom using an intensional relation R, φ is a conjunction of atoms using either intensional or extensional atoms, and every variable in $R(\vec{x})$ occurs in φ. A Datalog program is a finite collection of rules, and a Datalog query is a program along with a distinguished intensional relation, called the *goal relation*. Datalog programs can be considered as "shorthand" for LFP formulas over the extensional relations. The intensional relations are fixpoint variables: they are originally assigned to the empty set, and then the rules are iteratively fired to increase the intensional relations until a fixpoint is reached.

Example 26. *Consider the following Datalog program with intensional relation Reach and extensional relations $U(x)$ and $R(x,y)$:*

$$Reach(x) := U(x)$$
$$Reach(x) := R(y,x) \wedge Reach(y)$$

This computes the set of elements that are reachable by a collection of R edges from the elements satisfying U.

Datalog is a subset of LFP. For example, the set of elements z satisfying the fixpoint formula $[\mu_{X,x}.U(x) \vee \exists y(R(y,x) \wedge X(y))](z)$ is the same as the set computed by the Datalog program above.

Note that in the Datalog rule

$$Reach(x) := R(y,x) \wedge Reach(y)$$

the variable y is implicitly existentially quantified. Another notational shorthand often used in Datalog is to replace the $\wedge$ by a comma. Thus in the usual syntax, the rule above would be written:

$$Reach(x) := R(y, x), Reach(y)$$

Guarded Negation Fixpoint Logic (denoted GNFP) and *Guarded Fixpoint Logic* (denoted GFP) over a signature σ can be defined as the extensions of GNF and GF, respectively, with formulas

$$[\mu_{X,\vec{x}}.\alpha(\vec{x}) \wedge \varphi](\vec{y})$$

where (i) α is a guard from σ for the free first-order variables of φ, (ii) X only appears positively in φ, and (iii) second-order variables like X cannot be used as guards.

This is a subset of LFP. We have the restrictions on negation or quantification inherited from GNF or GF. Additionally, we note two restrictions on the use of fixpoint operators in GNFP. First, the fixpoint variables are guarded, and thus in building up fixpoints, we are only adding guarded elements. Further, there are no additional first-order variables other than those used in the fixpoint.

Consider the following example, adapted from [Bárány et al., 2015].

Example 27. *Let φ be the* GNF *formula*

$$\exists y_1 y_2[(R_1(x_1, y_1) \wedge R_2(x_2, y_2) \wedge X(y_1, y_2)) \vee (R_1(x_1, x_1) \wedge R_2(x_2, x_2))]$$

where R_1 and R_2 are two binary relations, and X is a fixpoint variable. Then $[\mu_{X,x_1,x_2}.S(x_1, x_2) \wedge \varphi](z_1, z_2)$ is in GNFP *and expresses the existence of a "ladder" consisting of R_1 and R_2 paths of the same length starting from z_1 and z_2, ending in self-loops, and such that the pair of elements on each rung are guarded by S.*

Guarded Datalog allows allows the use of intensional predicates with unrestricted arities, but for each rule $R(x_1 \dots x_n) := \psi(\vec{x}, \vec{y})$, the variables $\vec{x} \cup \vec{y}$ in the body ψ of the rule must appear in a single atom over an extensional relation within ψ. *Frontier-guarded Datalog* relaxes this by requiring only that the variables $x_1 \dots x_n$ in the head of the rule, must appear in a single EDB atom appearing in the body ψ. This subsumes monadic Datalog, since the single head variable in the monadic Datalog rules can be trivially guarded.

Frontier-guarded Datalog, and hence Guarded Datalog and Monadic Datalog, can be expressed in GNFP. We explain this on an example of a generic Frontier-guarded Datalog program with single intensional predicate goal

$$\begin{aligned} \text{goal}(\vec{x}) &:= \varphi_1(\vec{x}, \vec{y}_1) \\ \text{goal}(\vec{x}) &:= \varphi_2(\vec{x}, \vec{y}_2) \\ &\dots \\ \text{goal}(\vec{x}) &:= \varphi_n(\vec{x}, \vec{y}_n) \end{aligned}$$

This can be expressed in fixpoint logic as:

$$[\mu_{\text{goal},\vec{x}}. \bigvee_i \exists \vec{y}_i \ \varphi_i(\vec{x}, \vec{y}_i)](\vec{t})$$

Since each φ_i is guarded, and GNF is closed under existential quantification and disjunction, the disjunction is in GNF. Since the fixpoint variables are guarded, the fixpoint is also in GNFP.

Example 28. *Recall the property discussed in Example 7 and Example 8. The fixpoint logic description of this property from Example 8 required a subformula representing elements that are G-reachable from a node labelled U. Previously we wrote this in fixpoint logic as:*

$$[\mu_{S,y}.U(y) \vee \exists x(G(x,y) \wedge S(x))](z).$$

But the formula can be expressed in Guarded Datalog:

$$\begin{aligned} \mathit{goal}(x) &:= U(x) \\ \mathit{goal}(x) &:= G(y,x) \wedge \mathit{goal}(y) \end{aligned}$$

Note that in defining GNFP, we have restricted the use of negation as in GNF, but also restricted the use of the fixpoint operator, requiring it to be guarded. We can explain the need for this second restriction using Datalog.

Given a Datalog program P_1 and instance I for the extensional relations, the *output* of P_1 on I, denoted $P_1(I)$, is the set of tuples for the goal relation of P_1 in the fixpoint. If we translate P_1 into an LFP formula $\varphi(x_1 \dots x_n)$, then the output is simply the evaluation of φ in I.

A Datalog program P_1 is *contained in* a Datalog program P_2 having the same extensional relations, if for every input instance I for the extensional relations, $P_1(I) \subseteq P_2(I)$. P_1 and P_2 are *equivalent* if they are mutually contained; put another way, P_1 and P_2 are equivalent if they are logically equivalent as LFP formulas.

Even though Datalog is a simple fragment of LFP, without any negation, equivalence of Datalog programs is not decidable:

Theorem 19. *[Shmueli, 1993] The problem of determining whether two Datalog programs are equivalent is undecidable, even if we restrict to programs with* 0*-ary Goal predicates (i.e. sentences).*

However, Datalog programs P_1 and P_2 are equivalent if both $P_1 \wedge \neg P_2$ and $P_2 \wedge \neg P_1$ are not satisfiable. Thus Theorem 19 implies that if we drop the requirement that fixpoints are guarded in GNFP, the satisfiability problem becomes undecidable.

On the other hand, reasoning problems about *Monadic Datalog*, such as determining whether two Monadic Datalog queries are equivalent, are reducible to GNFP satisfiability. Thus from our decidability results for GNFP it will follow that equivalence and containment of Monadic Datalog programs is decidable.

The μ-calculus. Recall from Section 5 that the basic modal logic defined there considers only unary relations $U_1 \dots U_n$ and a single binary relation R. We can add fixpoints to basic modal logic. That is, we allow atoms $X(x)$ where X is a second order unary variable, and we have a new formation rule saying that if $\varphi(x, X \dots)$ is a formula and X is positive in φ, then $\mu_{X,x}\ \varphi$ is a formula. This language is called the *μ-calculus*.

If we take Example 24 and consider G as the distinguished binary relation, we see that the formula φ is actually in the μ-calculus.

Recall also that in basic modal logic, we allow formulas to be built up using the "box modality"

$$\exists y\ R(x, y) \wedge \varphi(y)$$

and the "diamond modality"

$$\forall y\ R(x, y) \rightarrow \varphi(y)$$

We can consider a variant of basic modal logic where we allow the role of the bound and free variables to be reversed. That is, we have a "backward box modality":

$$\exists y\ R(y, x) \wedge \varphi(y)$$

as well as a "backwards diamond modality"

$$\forall y\ R(y, x) \rightarrow \varphi(y)$$

If we take this bi-directional variant of modal logic and add fixpoints, we get the *two-way μ-calculus*.

Both the μ-calculus and the two-way μ-calculus are contained in GNFP, since every formula is monadic.

Normal forms. The normal form for GNF can be extended naturally to GNFP. The *normal form for* GNFP over a signature σ can be defined recursively as formulas of the form

$$\delta[Y_1 := \psi_1, \dots, Y_n := \psi_n] \quad \text{or} \quad [\mu_{X,\vec{x}}.\psi](\vec{x})$$

where

- δ is a UCQ over $\sigma \cup \{X_1, \dots, X_m, Y_1, \dots, Y_n\}$ for some second-order variables $X_1, \dots, X_m$ and fresh relations $Y_1, \dots, Y_n$,
- each ψ_i is of the form $\alpha_i \wedge \neg\varphi_i$ or $\alpha_i \wedge \varphi_i$ such that $\varphi_1, \dots, \varphi_n$ are in normal form GNFP, $\alpha_1, \dots, \alpha_n$ are guards from σ for the free first-order variables in $\varphi_1, \dots, \varphi_n$, and the number of free first-order variables in each ψ_i matches the arity of Y_i,
- ψ is of the form $\alpha(\vec{x}) \wedge \varphi$ such that φ is in normal form GNFP, φ uses the second-order variable X only positively, and α is a guard from σ for the free first-order variables in φ.

As before, the base case for this definition is a UCQ over σ (take $n = 0$).

In other words, we can see normal form GNFP formulas as being built up from atomic formulas using guarded negation, unions of conjunctive queries, and guarded fixpoint operators.

9.2 Decidability via the tree model property

A GNFP formula is associated with a *width* just as with GNF: the width is the largest number of first-order variables in any subformula, after being converted to normal form.

As with GNF, we will related the width of a GNFP formula to the kind of tree-like structures that we will need to look for in seeking a satisfying model. Recall the notion of a k-code of a structure from the earlier sections: we have trees whose nodes code at most k elements in a structure.

Our first goal will be to show the following result:

Theorem 20 ([Bárány et al., 2015])**.** GNFP *has the tree-like model property: if φ is satisfiable, then φ is satisfiable in a structure that has a k-code, where k is the width of φ.*

Recall the GNF^k-game, and the corresponding notion of unravelling from Section 7. Given a satisfying model M of a GNFP sentence φ, we can consider its k-unravelling U as before. We want to claim that U also satisfies φ. For this it suffices to show that GNFP sentences are preserved when we pass from a structure to its unravelling. More generally, it suffices to show:

Proposition 21. *Suppose there is a winning strategy for Duplicator in the GNF^k-game between M and M' starting from the empty partial homomorphism. Then for any* GNFP *sentence φ of width at most k, $M \models \varphi$ if and only if $M' \models \varphi$.*

Notice that the proposition above easily implies Theorem 20: given a GNFP sentence φ of width k, if it is satisfiable in a structure M, then we take the GNF^k unravelling of M, $\mathsf{GNUnravel}_k(M)$. By Proposition 15, $\mathsf{GNUnravel}_k(M)$ is GNF^k-bisimilar to M. Applying Proposition 21, we infer that $\mathsf{GNUnravel}_k(M)$ satisfies φ.

The proof of Proposition 21 is straightforward:

Proof. Let κ be the maximal cardinality of M and M' (which may be infinite). For structures of bounded cardinality like this, the closure ordinals for fixpoints is also bounded. Hence, we can translate from GNFP each fixpoint subformula into an infinite disjunction of formulas that describe the possible fixpoint approximants.

Formally, this translation is from GNFP to *κ-infinitary* GNF, which is defined the same as GNF except that we also allow the infinitary connectives:

$$\bigwedge_{i \in S} R(\vec{x}) \wedge \varphi_i(\vec{x}), \bigvee_{i \in S} R(\vec{x}) \wedge \varphi_i(\vec{x})$$

for any index set S of size at most κ, provided each $\varphi_i(\vec{x})$ is in κ-infinitary GNF.

It thus suffices to prove the result for κ-infinitary GNF. The induction is no more difficult than for GNF, because if each $\varphi_i(\vec{x})$ is preserved by the game so is their conjunction, while if some φ_i is preserved then so is the disjunction. □

Now we can observe that GNFP is a subset of GSO:

Proposition 22. *There is a linear translation of* GNFP *formulas to* GSO *formulas.*

Proof. The translation is inductive. For the fixpoint case:

$$[\mu_{X,\vec{x}}.R(\vec{x}) \wedge \varphi(\vec{x}, X)](\vec{t})$$

can be translated to

$$\begin{aligned} &\exists X \subseteq R \\ &X(\vec{t}) \wedge \\ &[\forall \vec{x}\ \varphi'(\vec{x}, X) \rightarrow X(\vec{x})] \wedge \\ &\forall Y \subseteq R\ ([\forall \vec{x}\ \varphi'(\vec{x}, Y) \rightarrow Y(\vec{x})] \rightarrow \forall \vec{z}\ X(\vec{z}) \rightarrow Y(\vec{z})) \end{aligned}$$

where φ' is the inductive translation of $R(\vec{x}) \wedge \varphi(\vec{x}, X)$.

The second conjunct expresses that X is a "superset of a fixpoint": closed under the operator associated to $R(\vec{x}) \wedge \varphi(\vec{x}, X)$; the third states states that X is contained in any other superset of a fixpoint. It is easy to check that this guarantees X is the least fixpoint. □

Thus the forward mapping theorem, Theorem 10, applies to GNFP, reducing the existence of a tree-like model to the existence of a tree model. As before, we can apply Rabin's theorem to decide whether a tree-like model of the formula exists. Thus we have shown:

Theorem 21. *Satisfiability of a* GNFP *sentence is decidable.*

9.3 Optimized decidability of GNFP via translation to an automaton

Recall that for GNF we discussed several ways of converting a formula to an automaton. One approach was the inductive construction of Subsection 7.4 in which we convert each subformula to an automaton, with inductive steps for each construct of the grammar for a formula in normal form. Note that it is not clear how to get a useful "Scott Normal Form" in the presence of fixpoints, since we need to deal with the presence of free second order variables. Thus we will proceed by extending the approach from Subsection 7.4 that does not rely on this further normalization.

Recall that in the "direct inductive translation" for GNF, the inductive step for UCQ-shaped formula was handled in a special way, by converting the automata for nested formulas to localized automata and then "plugging" them into an automaton for the outer UCQ. We can extend this construction to GNFP.

In order to do this, we need an inductive step for the fixpoint operator, and we will perform this directly on localized automata. For this step, it is helpful to view testing whether a tuple is in the fixpoint as a game.

Testing whether some tuple $\vec{t}$ is in the least fixpoint $[\mu_{Y,\vec{y}} \,.\, \varphi]$ in some structure M can be viewed as a game. Positions in this game consist of the current tuple $\vec{y}$ being tested in the fixpoint, with the initial position being $\vec{t}$. In general, in position $\vec{y}$, one round of the game consists of the following:

- Eve chooses some valuation for Y such that $\varphi(\vec{y}, Y)$ holds (if it is not possible, she loses), then
- Adam chooses tuple $\vec{y}' \in Y$ (if it is not possible, he loses), and the game proceeds to the next round in position $\vec{y}'$.

Adam wins if the game continues forever.

The idea is that if $\vec{t}$ really is in the least fixpoint, then it must be added in some fixpoint approximant. This gives Eve a strategy for choosing Y at each stage in the game, in such a way that after finitely many challenges by Adam, she should be able to guess the empty valuation.

When the fixpoint can consist of only guarded tuples, like in GNFP, there is a version of this game on a tree code t of a structure, that can be implemented using tree automata. We start with an automaton $\mathcal{A}_\varphi$ for the body φ of the fixpoint. In fact, we start with a localized automaton for φ because we need to launch different versions based on Adam's challenges. Initially, Eve navigates to a node in t carrying $\vec{t}$, and launches the appropriate localized $\mathcal{A}_\varphi$ from there. In general, the game proceeds as follows:

- Eve and Adam simulate some localized version of $\mathcal{A}_\varphi$. During the simulation Eve can guess a valuation for Y (recall that $\mathcal{A}_\varphi$ runs on trees with an annotation describing the valuation for the second-order variable Y, and that information is missing from t). Because Y can only contain guarded tuples, this amounts to guessing an annotation of the tree with this valuation.
- When Eve guesses some $\vec{y}' \in Y$, Adam can either continue the simulation, or challenge her on this assertion. A challenge corresponds to launching a new localized copy of $\mathcal{A}_\varphi$ from the node carrying $\vec{y}'$ (again, we know that $\vec{y}'$ must be present locally in a node, since any tuple in the fixpoint must be guarded).

After each challenge, the game continues as before (with the new copy of $\mathcal{A}_\varphi$ being simulated, Eve guessing a new valuation for Y, etc.). Adam wins if he challenges infinitely often, or if the game stabilizes in some simulation of $\mathcal{A}_\varphi$ where he wins.

Assuming we have a localized automaton for φ, we can implement this game using a 2-way alternating parity automaton. We assign a large odd priority (larger than the priorities in $\mathcal{A}_\varphi$) to the states where Adam challenges, so that he wins if he is able to challenge infinitely many times; the other priorities are just inherited from $\mathcal{A}_\varphi$.

This construction is summarized in the following lemma (see page 56 for the definition of monotonicity for an automaton).

Lemma 8. *Let $\eta(\vec{y})$ be in* GNF *over $\sigma \cup \{Y\}$ that is positive in Y and where the number of free variables of η matches the arity of Y. Let $\mathcal{A}_\eta$ be a localized* 2APT$^\omega$ *for η over $\Sigma^{code}_{\sigma\cup\{Y\},k}$-trees that is monotonic with respect to Y.*

Then we can construct a localized 2APT$^\omega$ *$\mathcal{A}_\psi$ over $\Sigma^{code}_{\sigma,k}$ for*

$$\psi := [\mu_{Y,\vec{y}}\,.\, R(\vec{y}) \wedge \eta(\vec{y})](\vec{z})$$

in linear time such that the number of states is the same as in $\mathcal{A}_\eta$, and the number of priorities is increased by 1.

The monotonicity property of $\mathcal{A}_\eta$ is important for the correctness of Lemma 8. Normally, it would be problematic to guess the missing information about Y while simulating an alternating automaton like $\mathcal{A}_\eta$, since the automaton might not consistently declare that a certain tuple is in the fixpoint approximant. However, for an automaton that is monotonic in Y, this makes no difference.

We can define the *CQ-rank* of a GNFP formula exactly as we did for GNF: the maximum number of atoms in a CQ-shaped formula when normalized. As with GNF, we will end up with an automaton having states representing that track the CQ-shaped subformulas that are satisfied, and thus the CQ-rank will impact the number of states in an automaton.

With Lemma 8 in place, we can now adapt Lemma 4 to inductively define an automaton for any GNFP sentence φ. The induction is based on the nesting depth of both UCQ-shaped formulas and fixpoints.

Lemma 9. *Given a normal form formula $\psi(\vec{x})$ in* GNFP *of width at most k over signature σ, we can construct a* 2APT$^\omega$ *$\mathcal{A}_\psi$ such that for all valid k tree codes t, for all local assignments $\vec{a}/\vec{x}$, and for all nodes v in t with $\vec{a} \subseteq \mathrm{names}(v)$,*

$$\mathcal{A}_\psi^{\vec{a}/\vec{x}} \text{ accepts } t \text{ starting from } v \quad \textit{iff} \quad \mathrm{decode}(T) \text{ satisfies } \psi([v,\vec{a}]).$$

Further, there is a polynomial function f independent of ψ such that the number of states of $\mathcal{A}_\psi$ is at most $N_\psi := f(m_\psi) \cdot 2^{f(kr_\psi)}$ and the number of priorities is at most $f(km_\psi)$ where $m_\psi = |\psi|$, r_ψ is the CQ-rank of ψ. The overall size of the automaton and the running time of the construction is at most exponential in $|\sigma| \cdot N_\psi$.

This means that despite the increased expressivity of GNFP over GNF, satisfiability testing can still be done in 2EXPTIME.

9.4 Fixpoint logics and open world query answering

Recall from Subsection 9.1 that Datalog programs represent a special kind of LFP formula, one in which there is no negation. Datalog programs are written in a special syntax, consisting of rules relating intensional relations to each other and to extensional relations. Rules are a kind of short-hand for fixpoints, where the intensional relations represent fixpoint variables while the extensional relations are ordinary relations that are part of the input signature. We have also mentioned one subset of Datalog that is contained in GNFP: Monadic Datalog,

in which the intensional relations are all unary, as well as Guarded Datalog, in which each rule is of the form:

$$R(\vec{x}) := A(\vec{x}, \vec{y}) \wedge \varphi(\vec{x}, \vec{y})$$

where $A(\vec{x}, \vec{y})$ is an atom using an extensional relation A, containing all variables of the rule. Finally we defined Frontier-guarded Datalog, in which rules are of the form

$$R(\vec{x}) := A(\vec{x}) \wedge \varphi(\vec{x}, \vec{y})$$

where $A(\vec{x}, \vec{y})$ is an atom using an extensional relation, which contains all variables in the head of the rule.

We showed that all of these are expressible in GNFP.

Now let us return to the Open World Query Answering problem, OWQ. Recall from Section 8 that the input to this problem is a sentence φ, a finite structure M_0, and a background theory Σ. The output is yes exactly when

$$M_0 \wedge \Sigma \models \varphi$$

We say φ *is entailed by* Σ *on* M_0 when this holds. Typically φ is a conjunctive query or union of conjunctive queries. For simplicity, we will restrict to the case where φ is a conjunctive query below.

Recall also that a *frontier-guarded TGD* is a sentence of the form:

$$\forall x_1 \dots x_j \bigwedge_{m \leq q} A_m \rightarrow$$
$$\exists y_1 \dots y_k \bigwedge_{n \leq r} B_n$$

where $A_1 \dots A_q$ and $B_1 \dots B_r$ are relational atoms and there is one atom A_i that contains all the variables in $\vec{x}$ that occur in some B_n (a *guard* for the *exported variables*). A *Guarded TGD* is a sentence as above, in which there is a guard A_i for all variables in $\vec{x}$. We showed that frontier-guarded TGDs are expressible in GNF, and hence the same is true for guarded TGDs.

In Section 8 we showed that for Σ consisting of frontier-guarded TGDs, the Open World Query Answering problem is decidable: this followed via a reduction to GNF satisfiability. This gives a 2EXPTIME algorithm, and one can show that this is optimal. We also showed that the "data complexity" is in CoNP. That is, when we fix Σ and φ, there is an algorithm that runs in CoNP in the size of M_0. However, one could hope for an algorithm that has even lower data complexity. One way to achieve this aim is to reduce Open World Query Answering to standard evaluation of a formula. Given a conjunctive query Q and a theory Σ, a *rewriting* of Q with respect to Σ is a formula Q_Σ such that:

For any finite structure M_0, Q_Σ holds in M_0 exactly when $M_0 \wedge \Sigma \models \varphi$

Thus if we can rewrite Q to Q_Σ, we have reduced the open world query answering problem for Q and Σ to evaluation of Q_Σ on M_0.

We will show that when Σ consists of Guarded TGDs, for every CQ Q we can get a Q_Σ that consists of Frontier-Guarded Datalog rules.

Example 29. *Consider a signature with binary relations $R(x, y)$ and $S(x, y)$ as well as unary relations $U(x)$ and $UReach(x)$.*

Consider the Guarded TGDs:

$$R(x, y) \wedge UReach(y) \rightarrow UReach(x)$$
$$U(x) \rightarrow UReach(x)$$
$$UReach(x) \rightarrow \exists z\ S(x, z)$$

and the query $Q(x) = \exists z\ S(x, z)$.

One can check that the certain answers of Q under Σ are given by Datalog program with the following rules:

$$UReach'(x) := UReach(x)$$
$$UReach'(x) := U(x)$$
$$UReach'(x) := R(x, y) \wedge UReach(y)$$

with $UReach'(x)$ being the goal relation.

Notice that this is a Guarded Datalog program, since the body of each rule is guarded.

We start with a result that gives the intuition for how this rewriting works:

Theorem 22. *For every set Σ of Guarded TGDs, and for every atomic conjunctive query $Q(\boldsymbol{x})$, one can effectively find a Guarded Datalog program P such that the result of P on any structure M is the same as the certain answers to Q on M.*

Note that entailment here, and throughout the section, can be interpreted either in the classical sense or in the finite sense, since we have the finite model property. Indeed, in our proofs, we use constructions that make use of infinite structures, but the conclusion holds in the finite.

A *Full Guarded TGD* is a TGD with no existentials in the head. The idea behind the proof the theorem will be that we take all full guarded TGDs that are consequences of Σ, and turn them into Datalog rules. We will show that the full guarded TGDs are sufficient to capture the certain answers.

We say that a structure M is *fact-saturated* (with respect to Σ) if no new fact over the active domain of M plus the elements named by constant symbols is entailed by the facts of M together with Σ.

Lemma 10. *For Σ a set of Frontier-Guarded TGDs, if a structure M is not fact-saturated with respect to Σ, then there is a guarded subset X of the domain of M such that the induced substructure M_X is not fact-saturated with respect to Σ.*

Proof. We prove the contrapositive. Assume that every induced substructure M_X, for X a guarded subset, is fact-saturated with respect to Σ. Let N be constructed from M by chasing each M_X with Σ independently and taking the union of the results: that is $N = \bigcup_{X \text{ guarded}} \mathsf{Chase}_\Sigma(M_X)$. Recalling that the chase of M_X only satisfies facts over M_X that are entailed, we see that N does not satisfy any new facts over the domain of M.

We claim that N satisfies every sentence in Σ. Given a σ in Σ of the form

$$\forall \vec{x}\ (\varphi(\vec{x}) \rightarrow \exists \vec{y}\, \rho(\vec{x}, \vec{y}))$$

and a binding of variables $\vec{x}$ into $\vec{b} \in N$ such that the corresponding facts $\varphi(\vec{b})$ hold in N. Note that $\vec{b}$ is guarded. If $\vec{b}$ contains only constants and elements of $\vec{a}$, then each fact in $\varphi(\vec{b})$ must be in M. Hence $\varphi(\vec{b})$ is in M_X and we are done, since M_X satisfies Σ. Consider any non-constant element b_i outside of M. If any such element exists, then the guard fact for $\vec{b}$ must have been generated in the chase process for some M_{X_0}, hence every non-constant element a_i was generated in M_{X_0}, and every fact in $\varphi(\vec{b})$ involving such an element must be in M_{X_0}. Since each other fact is in M, hence in M_{X_0}, we have $\varphi(\vec{b})$ is contained in M_{X_0} as before, and so we are done because Σ holds in M_{X_0}.

Thus we have a structure satisfying Σ, containing M, and containing no new facts over the elements of M and the constants. Therefore M must be fact-saturated. □

We are now ready to give the proof of Theorem 22:

Proof of Theorem 22. A *derived Full Guarded TGD* for Σ is a full guarded TGD with a single atom in the head which are entailed by Σ. We let $\Sigma_{FullGuarded}$ be all the derived full guarded TGDs. Note that there are only finitely many of these.

Lemma 10 implies that:

For every finite structure M_0 and atomic query $Q = \text{goal}(\vec{x})$, the certain answers of Q over M_0 with respect to Σ are the same as the goal-facts using elements of M_0 entailed by M_0 and $\Sigma_{FullGuarded}$.

The Full TGDs of $\Sigma_{FullGuarded}$ are not quite Guarded Datalog: Guarded Datalog requires us to distinguish extensional and intensional relations, and requires that atoms over extensional relations do not occur as consequences within rules. We turn $\Sigma_{FullGuarded}$ into a Guarded Datalog program by replacing each relation R in $\Sigma_{FullGuarded}$ by a copy R'; thus a Full TGD:

$$\forall \vec{x}\ (R(\vec{x}, \vec{y}) \ldots \rightarrow S(\vec{x}))$$

is transformed to the Datalog rule:

$$S'(\vec{x}) := R'(\vec{x}, \vec{y}) \ldots$$

In addition we add rules:

$$R'(\vec{x}) := R'(\vec{x})$$

Finally, we let goal$'$ be the goal predicate. It is easy to see that this Datalog program computes a fact goal$'(\vec{a})$ over M exactly when goal$(\vec{a})$ is entailed by $\Sigma_{FullGuarded}$ over M. □

9.5 Bibliographic remarks and further reading

Applying tree automata techniques to decidable fixpoint logics goes back to work on the fixpoint extension of basic modal logic, the μ-calculus. The definition of the μ-calculus as well as automata-based decision procedures can be found in many places, including [Vardi, 1998].

The first definition of a fixpoint extension of a guarded relational logic was *Guarded fixpoint logic*, which is formed as GNFP is but extending GF rather than GNF. It was introduced by Grädel and Walukiewicz in [Grädel and Walukiewicz, 1999], and the complexity of satisfiability for the logic was established in the same paper. In [Grädel and Walukiewicz, 1999], first a decidability procedure was given using the tree model property, then a tight complexity bound is shown via alternating automata. A later presentation [Grädel, 1999a] works by reduction to the μ-calculus with backward and forward modalities ("two-way μ-calculus), relying on a prior decidability result for this logic [Vardi, 1998]. The Guarded Negation Fixpoint logic was introduced in [Bárány et al., 2011, Bárány et al., 2015], and the decidability results of [Grädel and Walukiewicz, 1999] were extended to it. As with GNF, the approach to satisfiability of GNFP in [Bárány et al., 2011, Bárány et al., 2015] was not through a direct automaton translation, but via reduction to Guarded fixpoint logic and the results of [Grädel and Walukiewicz, 1999].

For GFP and GNFP, satisfiability over finite structures is not the same as satisfiability over all structures. However satisfiability over finite structures was shown decidable for both GFP [Bárány and Bojańczyk, 2012] and GNFP [Bárány et al., 2015].

Our description of the game for deciding whether a tuple is in a fixpoint is adapted from [Benedikt et al., 2016b], but this approach to fixpoints has been used before, e.g., in [Blumensath et al., 2014].

The application of fixpoints to open world query answering derives from [Bárány et al., 2013]. Datalog rewriting for Guarded TGDs was first announced in [Marnette, 2011, Baget et al., 2011a].

10 Tree-like models and interpolation

We now discuss how the tree-like model property can be used to assist with computational problems on restricted logics other than satisfiability.

Interpolation relates to implication of formulas. Given formulas φ_1 and φ_2, we say φ_1 *entails* φ_2, written $\varphi_1 \models \varphi_2$, if:

> for every structure M and binding σ such that $M, \sigma \models \varphi_1$, it is the case that $M, \sigma \models \varphi_2$.

Entailment between formulas of a logic is closely related to the validity problem for the logic, since φ_1 entails φ_2 is the same thing as saying that $\forall \vec{x}\ \varphi_1 \to \varphi_2$ is valid.

Interpolation is based on the intuition that if φ_1 entails φ_2, then the "cause" of the entailment should be another formula that uses only relations that are used in both φ_1 and φ_2. Formally, let φ_{L} and φ_{R} be sentences over signatures σ_{L} and σ_{R} such that $\varphi_{\mathrm{L}} \models \varphi_{\mathrm{R}}$ (that is, φ_{L} entails φ_{R}). An *interpolant* for such an entailment is a formula θ for which $\varphi_{\mathrm{L}} \models \theta$ and $\theta \models \varphi_{\mathrm{R}}$, and θ mentions only relations appearing in both φ_{L} and φ_{R}.

A *uniform interpolant* for φ_{L}, subsignature σ' of σ_{L}, and logic $\mathcal{L}$ is a formula θ over signature σ' such that:

- $\varphi_{\mathrm{L}} \models \theta$ and
- for every entailment $\varphi_{\mathrm{L}} \models \varphi_{\mathrm{R}}$ with φ_{R} in $\mathcal{L}$ such that the signature of φ_{R} intersected with σ_{L} is contained in σ', θ is an interpolant for the entailment.

One can think of a uniform interpolant of φ_{L} as the best over-approximation of φ_{L} in signature σ', with respect to entailments of σ' formulas.

It is difficult to approximate a formula in a logic $\mathcal{L}$ while staying in the same logic. In particular, it is difficult to get uniform interpolants while staying within first-order logic:

Example 30. *Consider a signature σ_{L} consisting of a binary relation $R(x, y)$ and a unary relation $U(x)$.*

Let φ_{L} be:

$$\exists x\ U(x) \wedge (\forall x\ U(x) \to (\exists y\ R(x, y) \wedge U(y)))$$

That is, there is a U element, and every U element links to a U element via an R edge.

Consider σ' that has only $R(x, y)$ in it. For every number n, if we let φ_n state that there is some path with n edges, then φ_n is expressible in GF*. Indeed, it is the existential quantification of a formula in modal logic. Further φ_{L} entails each φ_n.*

Thus a uniform interpolant of φ_{L} over σ' with respect to GF *entailments would need to be a sentence using only R that is implied by φ_{L}, but which implies each φ_n. One can show that there is no such sentence in first-order logic.*

Note that we can find a uniform interpolant for φ_{L} over σ' in GNFP*. With greatest fixpoints we can write a sentence stating that there are arbitrarily long paths:*

$$\exists z\ [\nu_{longpaths,x}\exists y\ R(x,y) \wedge longpaths(y)](z)$$

Rewritten with only least fixed points it could be expressed as:

$$\exists z\ \neg[\mu_{shortpaths,x}.(\forall y\ R(x,y) \rightarrow shortpaths(y))](z)$$

The guarded universal quantification within can be converted to GF *in the usual way. The fixpoint is permitted in* GNFP *since it is unary, and similarly the outermost negation is permissible since it is unary. We can see that the sentence is implied by φ_{L} and implies each φ_n. Indeed, since the φ_n are essentially the only statements about R that are implied by φ_{L}, one can show that the sentence above is a uniform interpolant.*

Example 30 suggests that we can get uniform interpolants for guarded logic sentences by moving to a fixpoint logic. Our main result is that this is the case, and that such interpolants are effectively computable:

Theorem 23. *Given σ_{L} a relational signature, $\sigma' \subseteq \sigma_{\mathrm{L}}$, and $k \in \mathbb{N}$, and φ_{L} over σ_{L} in* GNFP^k*, we can compute an* LFP *sentence that is a uniform interpolant for* GNFP^k *entailments.*

Theorem 23 implies that entailments between GNFP formulas have LFP interpolants:

Corollary 3. *For $\varphi_{\mathrm{L}}, \varphi_{\mathrm{R}} \in$ GNFP for which the entailment $\varphi_{\mathrm{L}} \models \varphi_{\mathrm{R}}$ holds, there is an interpolant for the entailment in* LFP*.*

We now explain the main components of the proof of Theorem 23. It involves mapping back and forth between logics on relational structures and logics on trees. In the process we will develop a finer understanding of the relationship between relational structures and their codes.

Recall that in Theorem 10 we have given a *forward mapping*, translating an input GSO formula φ_0 to an MSO formula φ_0' over tree-codes, holding on the codes that correspond to tree-like models of φ_0. For a set of unary relations σ, we let $\mathsf{MSO}[\sigma]$ denote Monadic Second Order Logic over a binary child relation and the unary relations in σ.

If we look at the translation in Theorem 10, we can see that if we start with a first-order sentence, that we get a very restricted kind of $\mathsf{MSO}[\sigma]$ sentence over trees: to check a sentence, we need to just check for the existence of certain up-and-down paths connecting elements in the tree. This requires only the two-way μ-calculus (see Subsection 9.1). The same holds in the presence of fixed points:

Proposition 23. *For every* LFP *sentence φ and number k, we can construct in polynomial time a two-way μ-calculus sentence φ' in the language of k-codes such that for any model M that has a k-tree code t with root* Root

$$M \models \varphi \leftrightarrow t, \mathsf{Root} \models \varphi'$$

For the purposes of our interpolation argument, the important advantage of the two-way μ-calculus over MSO is that two-way μ-calculus formulas are preserved under bisimulation, provided that the inputs are trees. Recall that a bisimulation relates distinguished elements in two Kripke structures. We can extend the definition to talk about a bisimulation between two trees, taking the distinguished elements to be the roots. Formally, let us say that a formula $\varphi'(x)$ is *bisimulation-invariant over trees* if whenever t_1, Root_1 and t_2, Root_2 are bisimilar, then t_1, Root_1 satisfies φ' if and only if t_2, Root_2 does. Similarly if φ' is a sentence, we say that φ' is bisimulation-invariant over trees if whenever t_1, Root_1 and t_2, Root_2 are bisimilar, then t_1 satisfies φ' if and only if t_2 does.

Proposition 24. *If φ' is a two-way μ-calculus sentence over some vocabulary σ', then φ' is bisimulation-invariant over trees.*

Proof. We can define a bi-directional variant of the bisimulation game in which Spoiler can move backward. A position of the game consists of an element x in M and x' in M' as with the bisimulation game. A round of the game proceeds by Spoiler choosing one structure, say M. Spoiler can pick $x_1 \in M$ with $M \models R(x, x_1)$, and Duplicator must respond in the other structure with x'_1 such that $M' \models R(x', x'_1)$. But Spoiler now can also choose to pick $x_1 \in M$ with $M \models R(x_1, x)$, and Duplicator must respond in the other structure with x'_1 such that $M' \models R(x'_1, x')$. We can now extend Proposition 5 to show that if Duplicator has a winning strategy from this game, starting at position M, x and M', x', then M, x and M', x satisfy the same two-way μ-calculus formulas.

Thus it suffices to show that Duplicator has a winning strategy in the game starting from t_1, r_1 and t_2, r_2. Since t_1, r_1 and t_2, r_2 are bisimilar, Duplicator has a winning strategy in the one-way bisimulation game. Duplicator will play to ensure that each position is a position reachable from this winning strategy. Suppose the position of the game consists of t_1, c_1 and t_2, c_2. If Spoiler moves forward to a child d_1 of c_1, Duplicator just follows her winning strategy to choose a child d_2 of c_2. But if Spoiler moves backward from c_1, he is moving to the the parent of c_1 in the tree, p_1. Letting p_2 be the parent of c_2 in t_2, we know that the play at some previous step must have consisted of t_1, p_1 and t_2, p_2. Thus Duplicator can respond by moving from c_2 to p_2. □

We will not need to make use of any further properties of the two-way μ-calculus, since Propositions 23 and 24 tell us something important about *any* forward mapping. Let us consider any $\mathsf{MSO}[\sigma]$ sentence φ'_1 that represents a forward mapping. We know that there is a two-way μ-calculus sentence $\varphi'_2(x)$ such that φ'_1 is equivalent to φ'_2 evaluated at the root of a tree, and that two-way μ-calculus sentences are preserved under bisimulation at the root. Thus, using the definition of bisimulation-invariance over trees for a sentences, we see that *the result φ'_1 of the forward mapping is bisimulation-invariant over trees.*

For an arbitrary signature σ, we let $\mathsf{LFP}[\sigma]$ denote least fixpoint logic over signature σ.

For a φ'_0 over tree codes (with some given k and signature σ) a GNF^k *backward mapping for φ'_0* is a sentence φ_1 such that:

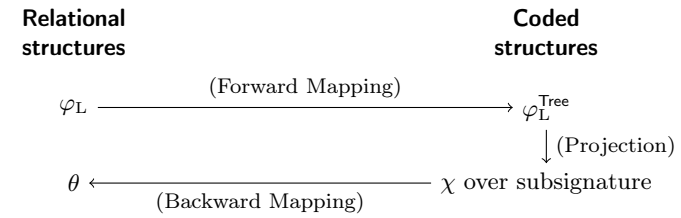

Figure 12: Back-and-forth for Interpolation

$$\text{for all } \sigma\text{-structures } M,\ M \models \varphi_1 \text{ iff } \mathsf{code}(\mathcal{U}_{\mathsf{GNUnravel}_k}(M)) \models \varphi_0',$$

where $\mathcal{U}_{\mathsf{GNUnravel}_k}(M)$ is the GNF^k-unravelling of M. That is, the backward mapping is the inverse of the forward mapping, but it does not talk about arbitrary tree codes, but only about unravellings.

Given a bisimulation-invariant MSO sentences, we can find a backward mapping for it in fixed point logic:

Theorem 24. *Given φ in $\mathsf{MSO}[\Sigma^{code}_{\sigma,k}]$ that is bisimulation-invariant, we can construct a GNF^k backward mapping $\varphi_1 \in \mathsf{LFP}[\sigma]$.*

The theorem is proven by first converting φ to a two-way μ-calculus formula, and then proceeding by induction on its structure.

Our overall approach for getting a uniform interpolant for a GNFP sentence φ_{L} over signature σ_{L} with respect to subsignature σ' will revolve going *back and forth between relational structures and their tree codes.* It works as follows:

- *use forward mapping* to go to an MSO sentence $\varphi_{\mathrm{L}}^{\mathsf{Tree}}$ representing tree codes of tree-like structures satisfying φ;
- *project* $\varphi_{\mathrm{L}}^{\mathsf{Tree}}$ to get an MSO sentence χ over tree codes in the target signature σ'; roughly, χ will represent all tree codes that can be expanded to a tree code satisfying φ;
- *use backward mapping* on χ to get a sentence θ over σ'.

Figure 12 shows the approach diagrammatically.

We have discussed the forward mapping and backward mapping already. We now discuss the component used in the projection step. It is easy to see that MSO sentences over trees are closed under projection – existentially quantifying over some subset of the node relations. However, to apply Lemma 24 we will need to come up with a sentence that is bisimulation invariant. We will thus want an MSO sentence that represents the bisimulation closure of the projection. The following lemma gives us this:

Lemma 11. *Let σ_1 be a collection of unary relations for nodes and $\sigma_0 \subseteq \sigma_1$. Given φ' in $\mathsf{MSO}[\sigma_1]$ that is bisimulation-invariant over trees we can construct $\theta' \in \mathsf{MSO}[\sigma_0]$ such that*

$t \models \theta'$ if and only there is t'_0 bisimilar to t over σ_0, and there exists t'_1 over σ_1 such that $t'_1 \models \varphi'$ and t'_0 is obtained by restricting t'_1 to the relations in σ_0.

We do not prove the lemma; it is proven using the idea of just projecting the automaton for φ'; the proof is due to D'Augostino and Hollenberg (see Theorem 3.3. of [D'Agostino and Hollenberg, 2000]).

We can show that the formula formed above is a uniform interpolant with respect to entailment over trees:

Theorem 25. *Let φ' in $\mathsf{MSO}[\sigma_1]$ be bisimulation-invariant over trees, $\sigma_0 \subseteq \sigma_1$, and let χ be the sentence formed from φ' in Lemma 11. Then χ is a uniform interpolant for φ' over σ_0, with respect to entailments in any logic L that is contained in MSO and is bisimulation-invariant.*

If M and M' are two Kripke structures with different vocabularies for the unary relations, and σ is a subset of the set of common unary relations, we say that M and M' are *bisimilar over σ* if there is a relation that satisfies the properties of a bisimulation but only for the relations in σ. In general, for any of our simulation relations, we can parameterize it by a subsignature in the obvious way.

Towards proving this claim we state an intermediate lemma about how to "amalgamate" two structures in different signatures that are bisimilar in their common signature.

Lemma 12. *If M_{LEFT} is a structure for signature σ_{L}, M_{RIGHT} is a structure for signature σ_{R}, and M_{LEFT} is bisimilar to M_{RIGHT} over signature $\sigma' = \sigma_{\text{L}} \cap \sigma_{\text{R}}$, then there is M' in $\sigma_{\text{L}} \cup \sigma_{\text{R}}$ that is bisimilar to M_{RIGHT} over signature σ_{R} and bisimilar to M_{LEFT} over signature σ_{L}.*

The short proof of this result can be found in Lemma 3.5 of [D'Agostino and Hollenberg, 2000].

We now prove Theorem 25. Suppose $\varphi' \models \rho$ where the common signature of ρ and φ' is contained in σ', and ρ is bisimulation invariant. Suppose that there is $t \models \chi \wedge \neg\rho$. By the definition of χ, there is t' over σ_{L} that is bisimilar to t over σ' which satisfies φ'. By Lemma 12 there is some t'' that is bisimilar to t over σ_{R} and bisimilar to t' over σ_{L}. t'' satisfies $\neg\rho$ since t satisfies $\neg\rho$ and t and t'' are bisimilar over σ_{R}, while t'' satisfies φ' since t' does and t'' and t' are bisimilar over σ_{L}. But this contradicts $\varphi' \models \rho$.

We can now make the back-and-forth approach outlined above more precise. The algorithm consists of the following steps:

- Apply the forward mapping to φ_{L} to get $\varphi_{\text{L}}^{\mathsf{Tree}}$.
- Let $\text{consistent}_{\sigma_{\text{L}},k}$ be the $\mathsf{MSO}[\Sigma^{\text{code}}_{\sigma_{\text{L}},k}]$-formula that expresses that a tree is consistent with respect to $\Sigma^{\text{code}}_{\sigma_{\text{L}},k}$. It is easy to see that such a sentence is bisimulation-invariant, since it involves only some properties of the label of a node with respect to its parent.

Since $\varphi_{\mathrm{L}}^{\mathsf{Tree}}$ and $\mathrm{consistent}_{\sigma_{\mathrm{L}},k}$ are both bisimulation-invariant, so is their conjunction $\varphi_{\mathrm{L}}^{\mathsf{Tree}} \wedge \mathrm{consistent}_{\sigma_{\mathrm{L}},k}$. We can thus apply Lemma 11 to $\varphi_{\mathrm{L}}^{\mathsf{Tree}} \wedge \mathrm{consistent}_{\sigma_{\mathrm{L}},k}$ to get χ over the subsignature $\Sigma^{\mathrm{code}}_{\sigma',k}$.

- We apply the $\mathsf{GNF}^k[\sigma']$-backward mapping to χ to get θ in LFP over σ'.

We claim that θ is the required uniform interpolant. By definition of the backward mapping, θ is in LFP over σ'.

For θ to be a uniform interpolant for φ_{L} means two things:

- θ is an overapproximation of φ_{L}: that is φ_{L} entails θ
- θ is the best approximation: that is for all φ_{R} in GNFP^k whose signature overlaps with that of φ_{L} in the subsignature σ', if φ_{L} entails φ_{R}, then θ entails φ_{R}.

We will prove both of these by mapping to the corresponding statements on trees.

Original sentence entails interpolant. First, we prove $\varphi_{\mathrm{L}} \models \theta$. Let M be a σ_{L}-structure and assume $M \models \varphi_{\mathrm{L}}$. Then $\mathcal{U}_{\mathsf{GNUnravel}_k,\sigma_{\mathrm{L}}}(M) \models \varphi_{\mathrm{L}}$ since $\mathcal{U}_{\mathsf{GNUnravel}_k,\sigma_{\mathrm{L}}}(M)$ and M agree on all $\mathsf{GNFP}^k[\sigma_{\mathrm{L}}]$ sentences. Hence, by the property of $\varphi_{\mathrm{L}}^{\mathsf{Tree}}$ given by the forward mapping, we have $\mathsf{code}(\mathcal{U}_{\mathsf{GNUnravel}_k,\sigma_{\mathrm{L}}}(M)) \models \varphi_{\mathrm{L}}^{\mathsf{Tree}}$. Since $\mathsf{code}(\mathcal{U}_{\mathsf{GNUnravel}_k,\sigma_{\mathrm{L}}}(M))$ is a consistent $\Sigma^{\mathrm{code}}_{\sigma_{\mathrm{L}},k}$-tree, this means that $\mathsf{code}(\mathcal{U}_{\mathsf{GNUnravel}_k,\sigma_{\mathrm{L}}}(M)) \models \varphi_{\mathrm{L}}^{\mathsf{Tree}} \wedge \mathrm{consistent}_{\sigma_{\mathrm{L}},k}$. Since χ is a uniform interpolant for $\varphi_{\mathrm{L}}^{\mathsf{Tree}} \wedge \mathrm{consistent}_{\sigma_{\mathrm{L}},k}$, we have $\mathsf{code}(\mathcal{U}_{\mathsf{GNUnravel}_k,\sigma_{\mathrm{L}}}(M)) \models \chi$. But χ is in $\mathsf{MSO}[\Sigma^{\mathrm{code}}_{\sigma',k}]$, so the restriction of $\mathsf{code}(\mathcal{U}_{\mathsf{GNUnravel}_k}(M))$ to the subsignature $\Sigma^{\mathrm{code}}_{\sigma',k}$ also satisfies χ. Moreover, the restriction of $\mathsf{code}(\mathcal{U}_{\mathsf{GNUnravel}_k,\sigma_{\mathrm{L}}}(M))$ to the subsignature is the same as just taking the unravelling with respect to this subsignature. Hence, letting $\mathcal{U}_{\mathsf{GNUnravel}_k,\sigma'}(M)$ be the unravelling of M with respect to σ' we have that $\mathsf{code}(\mathcal{U}_{\mathsf{GNUnravel}_k,\sigma'}(M)) \models \chi$. So since θ is the backward mapping of χ, we have that $M \models \theta$.

Interpolant entails appropriate sentences in subsignature. Assume that $\varphi_{\mathrm{L}} \models \varphi_{\mathrm{R}}$ for some $\varphi_{\mathrm{R}} \in \mathsf{GNFP}^k[\sigma_{\mathrm{R}}]$ with $\sigma_{\mathrm{R}} \cap \sigma_{\mathrm{L}} \subseteq \sigma'$. We need to show that $\theta \models \varphi_{\mathrm{R}}$. Let $\varphi_{\mathrm{R}}^{\mathsf{Tree}}$ be the result of applying the forward mapping to φ_{R}.

We want to claim that the entailment between φ_{L} and φ_{R} transfers to the tree world, once we add on the consistency sentence. Let $\sigma'' = \sigma_{\mathrm{L}} \cup \sigma_{\mathrm{R}}$.

We claim that over trees $\mathrm{consistent}_{\sigma_{\mathrm{L}},k} \wedge \varphi_{\mathrm{L}}^{\mathsf{Tree}} \models \mathrm{consistent}_{\sigma_{\mathrm{R}},k} \rightarrow \varphi_{\mathrm{R}}^{\mathsf{Tree}}$ over all $\Sigma^{\mathrm{code}}_{\sigma'',k}$-trees. Suppose that t is a $\Sigma^{\mathrm{code}}_{\sigma'',k}$-tree and $t \models \mathrm{consistent}_{\sigma_{\mathrm{L}},k} \wedge \varphi_{\mathrm{L}}^{\mathsf{Tree}}$. Then t must be consistent with respect to the subsignature $\Sigma^{\mathrm{code}}_{\sigma_{\mathrm{L}},k}$. If t is not consistent with respect to $\Sigma^{\mathrm{code}}_{\sigma_{\mathrm{R}},k}$, then t trivially satisfies $\mathrm{consistent}_{\sigma_{\mathrm{R}},k} \rightarrow \varphi_{\mathrm{R}}^{\mathsf{Tree}}$ and we are done. Otherwise, t is consistent with respect to both $\Sigma^{\mathrm{code}}_{\sigma_{\mathrm{L}},k}$ and $\Sigma^{\mathrm{code}}_{\sigma_{\mathrm{R}},k}$, which is enough to conclude that it is a consistent $\Sigma^{\mathrm{code}}_{\sigma'',k}$-tree. Hence, by the definition of $\varphi_{\mathrm{L}}^{\mathsf{Tree}}$ we have $\mathrm{decode}(t) \models \varphi_{\mathrm{L}}$. Since $\varphi_{\mathrm{L}} \models \varphi_{\mathrm{R}}$, this means that

$\mathrm{decode}(t) \models \varphi_{\mathrm{R}}$. Another application of the forward mapping theorem allows us to conclude that $t \models \varphi_{\mathrm{R}}^{\mathsf{Tree}}$, and hence $t \models \mathrm{consistent}_{\sigma_{\mathrm{R}},k} \to \varphi_{\mathrm{R}}^{\mathsf{Tree}}$ as desired.

Now recall that χ is a uniform interpolant for $\mathrm{consistent}_{\sigma_{\mathrm{L}},k} \wedge \varphi_{\mathrm{L}}^{\mathsf{Tree}}$ over the subsignature $\Sigma^{\mathrm{code}}_{\sigma',k}$ for entailments of bisimulation-invariant sentences. Applying this to $\mathrm{consistent}_{\sigma_{\mathrm{L}},k} \wedge \varphi_{\mathrm{L}}^{\mathsf{Tree}} \models \mathrm{consistent}_{\sigma_{\mathrm{R}},k} \to \varphi_{\mathrm{R}}^{\mathsf{Tree}}$ we know that $\chi \models \mathrm{consistent}_{\sigma_{\mathrm{R}},k} \to \varphi_{\mathrm{R}}^{\mathsf{Tree}}$ over all $\Sigma^{\mathrm{code}}_{\sigma'',k}$-trees. We will now use the properties of the backward mapping to show that $\theta \models \varphi_{\mathrm{R}}$.

Let M be a σ''-structure such that $M \models \theta$. Hence,

$$\mathsf{code}(\mathcal{U}_{\mathsf{GNUnravel}_k,\sigma'}(M)) \models \chi$$

by the properties of the backward mapping θ of χ. But

$$\mathsf{code}(\mathcal{U}_{\mathsf{GNUnravel}_k,\sigma''}(M)) \models \chi$$

as well, since $\mathsf{code}(\mathcal{U}_{\mathsf{GNUnravel}_k,\sigma''}(M))$ is exactly $\mathsf{code}(\mathcal{U}_{\mathsf{GNUnravel}_k,\sigma'}(M))$ when restricted to $\Sigma^{\mathrm{code}}_{\sigma',k}$. By the previous paragraph this implies that

$$\mathsf{code}(\mathcal{U}_{\mathsf{GNUnravel}_k,\sigma''}(M)) \models \mathrm{consistent}_{\sigma_{\mathrm{R}},k} \to \varphi_{\mathrm{R}}^{\mathsf{Tree}}$$

Since $\mathsf{code}(\mathcal{U}_{\mathsf{GNUnravel}_k,\sigma''}(M))$ is a consistent $\Sigma^{\mathrm{code}}_{\sigma'',k}$-tree, it is also $\Sigma^{\mathrm{code}}_{\sigma_{\mathrm{R}},k}$-consistent. Hence,

$$\mathsf{code}(\mathcal{U}_{\mathsf{GNUnravel}_k,\sigma''}(M)) \models \varphi_{\mathrm{R}}^{\mathsf{Tree}}$$

By the forward mapping, this means that $\mathcal{U}_{\mathsf{GNUnravel}_k,\sigma''}(M) \models \varphi_{\mathrm{R}}$. Since M and $\mathcal{U}_{\mathsf{GNUnravel}_k,\sigma''}(M)$ agree on all GNFP^k sentences and $\varphi_{\mathrm{R}} \in \mathsf{GNFP}^k[\sigma_{\mathrm{R}}]$ with $\sigma_{\mathrm{R}} \subseteq \sigma''$, this means that $M \models \varphi_{\mathrm{R}}$ as desired.

This completes the proof that θ entails φ_{R}, and hence completes the proof that θ is a uniform interpolant (Theorem 23).

10.1 Bibliographic Remarks

Interpolation results were first proven for first-order logic by Craig [Craig, 1957]. An important logic that has uniform interpolation is the μ-calculus, the natural extension of basic modal logic with a fixpoint operator. The uniform interpolation result was proven by D'Agostino and Hollenberg [D'Agostino and Hollenberg, 2000]. The interpolation result we present for GNFP^k is based on arguments in [Benedikt et al., 2015, 2017], incorporating parts of the proof for μ-calculus given in [D'Agostino and Hollenberg, 2000]. The underlying idea of a backward mapping from trees to relational logics originates in Grädel, Hirsch and Otto's [Grädel et al., 2002]. [Grädel et al., 2002] gives applications of this technique to give several insights into the expressiveness of guarded logics.

11 Larger decidable fixpoint logics

In Section 9 we presented the logic GNFP and used the technique of tree-like models to prove that it is a decidable subset of least fixpoint logic LFP. We also showed, using a translation to automata, that the complexity of satisfiability has the same worst-case behavior as GNF. In particular, the complexity is much better than the complexity of MSO on trees, with the latter being non-elementary.

It is natural to ask whether there are larger fixpoint logics that share the attractive decidability and complexity properties of GNF. Here we introduce an extension of GNFP, denoted GNFP-UP. Decidability for this logic will be almost immediate from the argument for GNFP. But the automata-based analysis will be more problematic. We can construct automata, but we really require a blow-up. The analysis will allow us to de-couple the features of a fixpoint logic that lead to decidability, against those that lead to elementary complexity.

11.1 GNFP-UP basics

Recall again LFP, which extends first-order logic with second-order variables, which can be bound via a fixpoint constructor.

We now look at another restriction of LFP, denoted *Guarded negation fixpoint logic with unguarded parameters* (GNFP-UP). This is a fragment of LFP that restricts negation as in GNF, but is more liberal than GNFP in its restrictions of fixpoints. Informally GNFP-UP allows two types of variables in the body of a fixpoint: "fixpoint variables" – those that are changing in a fixpoint calculation – and "parameter variables" – additional values that are kept fixed during a fixpoint. It allows the parameter variables to be unguarded in fixpoint definitions, but requires fixpoint variables and negation to be guarded.

Formally, a GNFP-UP[σ] *formula* φ is generated recursively from the following grammar:

$$\begin{aligned}\varphi ::= {} & R\vec{t} \mid Y\vec{t} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists y.\varphi \mid \\ & \alpha \wedge \neg\varphi \text{ where } \mathsf{Free}(\varphi) \subseteq \mathsf{Free}(\alpha) \mid \\ & [\mu^{\vec{z}}_{Y,\vec{y}}.\, \mathrm{gdd}(\vec{y}) \wedge \varphi(\vec{y},\vec{z},Y,\vec{Z})](\vec{t}) \text{ for } \varphi \text{ positive in } Y\end{aligned}$$

Above $\vec{t}$ is a tuple of variables or constants, $R\vec{t}$ and α are atoms using a relation in σ or $=$, In the fixpoint construction, the variables $\vec{z}$ are the *parameter variables* of the fixpoint, while $\vec{y}$ are the *fixpoint variables*. The last thing we have to explain in the grammar above is the *guardedness predicate* $\mathrm{gdd}(\vec{y})$. This asserts that $\vec{y}$ is guarded by an atom in σ or $=$. It can be understood as an abbreviation for the disjunction of existentially quantified atoms that use a relation from σ or $=$ and involve all of the variables in $\vec{y}$. Because of this, only guarded relations can be defined using fixpoints in GNFP-UP: i.e. any tuple of elements in the relation defined by the fixpoint formula must already be guarded by an atom in the base signature σ. Note that the relations defined using a fixpoint operator cannot be used as guards.

The parameters $\vec{z}$ are not required to be guarded in the fixpoint definition. However, for the purposes of negation, parameters are treated like other variables and must be guarded. For example, if $\alpha(\vec{x})$ is an atomic formula over σ, and Y identifies a fixpoint with parameters $\vec{z}$, then $\alpha(\vec{x}) \wedge \neg Y\,\vec{x}$ is not permitted since Y implicitly uses parameters $\vec{z}$ and these parameters are not guarded by α (since the free variables in $Y\,\vec{x}$ are really $\vec{x}$ and $\vec{z}$).

A formula φ that includes free first-order variables $\vec{x}$ is *$\vec{x}$-guarded* if it is logically equivalent to $\mathrm{gdd}(\vec{x}) \wedge \varphi$. If $\mathsf{Free}(\varphi) = \vec{x}$ and φ is $\vec{x}$-guarded, then we say it is *answer-guarded*. Sentences or formulas with one free variable are always answer-guarded since we can use a trivial guard like $x = x$. For readability purposes, we often omit such trivial guards.

GNFP-UP vs. GNFP. A good example to keep in mind is that GNFP-UP can express the transitive closure of a binary relation R.

Example 31. *Suppose R is a binary relation in σ. Consider the following* GNFP-UP$[\sigma]$*-formula:*

$$\varphi(x,z) := [\mu^{z}_{Y,y}\ .\ R\,yz \vee \exists y'\ (Ryy' \wedge Y\,y')](x)\ .$$

Observe that φ has two free variables, the variable x being tested in the fixpoint and the parameter variable z. The formula $\varphi(x,z)$ expresses that there is some R-path from element x to z, i.e. (x,z) is in the transitive closure of R.

We can express that x participates in an R-cycle by using the formula $\varphi(x,x)$.

This gives us new formulas. But what new sentences can we get out of this? We cannot express that the structure is strongly R-connected, since this would require unguarded negation. But we can say every pair of guarded elements is R-connected: $\neg\exists\ x\ z\ (\mathrm{gdd}(x,z) \wedge \neg\varphi(x,z)) \in$ GNFP-UP*.*

Note that the sentence $\neg\exists x\ z\ (\varphi(x,z) \wedge \neg Rxz) \vee (Rxz \wedge \neg\varphi(x,z))$ that says R is transitively closed is not in GNFP-UP*, since we cannot use the fixpoint relation defined by φ as a guard for $\neg Rxz$.*

We now build on the transitive closure example to show how to use GNFP-UP to express *navigational queries*. These are formulas over a signature σ containing only binary and unary relations, generalizing the case of a Kripke Structure, which has only a single binary relation. One can think of an instance of such a schema as a "graph database". For these languages, a regular expression E over symbols R, R^{-} coming from binary relations $R \in \sigma$ can be seen as defining a *navigation relation* that holds for (x,y) exactly when there is some path between x and y matching E. A *conjunctive 2-way regular path query* (C2RPQ) [Calvanese et al., 2000] is just a CQ over such expressions.

We now claim:

Proposition 25. *Every boolean* C2RPQ *can be expressed in* GNFP-UP*.*

Example 32. *Consider a signature consisting of a binary relation $R(x,y)$ and two unary relations $U(x)$ and $V(x)$.*

The property that there is an alternating U and V path from x to y can be expressed as a C2RPQ

$$\varphi_1(x,y) = x(UV)^*y$$

The property that there is a node labeled U havng both even- and and odd-length paths from x and having an even-length paths to y can be expressed as a C2RPQ

$$\begin{aligned}\varphi_2(x,y) = \\ \exists u\ U(u) \wedge [x((U \mid V)(U \mid V))^*y] \\ \wedge [x((U \mid V)(U \mid V))^*(U \mid V)u] \\ \wedge [u((U \mid V)(U \mid V))^*y]\end{aligned}$$

Proof. Consider some C2RPQ over signature σ. Let $\Sigma := \{R, R^- : R \text{ is a binary relation in } \sigma\}$.

Given a regular expression E over Σ, we can capture the navigation relation defined by it using a GNFP-UP formula E'. We start with a finite state automaton $\mathcal{A} = \langle \Sigma, Q, q_0, \Delta, F\rangle$ for E and write a GNFP-UP$[\sigma]$ formula with simultaneous fixpoints $E'(x,y) := [\mu^y_{X_0,x}\ .\ S](x)$ which has a second-order variable X_i for each state $q_i \in Q$, and the equation for the i-th component X_i, x_i in S captures the possible transitions from state q_i:

$$\bigvee_{(q_i,T,q_j)\in\Delta} \exists z\ (\chi_T(x_i,z) \wedge X_j\, z)\ \vee \begin{cases} x_i = y & \text{if } i \in F \\ \bot & \text{if } i \notin F \end{cases}$$

where $\chi_T(x_i,z)$ is Rx_iz if $T = R$ and Rzx_i if $T = R^-$.

Once we have E' in GNFP-UP for each regular expression E appearing in the C2RPQ, it is easy to translate into GNFP-UP by replacing each $E(x,y)$ in the C2RPQ by $E'(x,y)$. □

A similar argument shows that GNFP-UP can express disjunctions of C2RPQs.

In LFP, it is always possible to eliminate the use of parameters by increasing the arity of the defined fixpoint predicates and passing the parameters explicitly in the fixpoint. This is not usually possible in our context, because the fixpoint variables are required to be guarded. Indeed, it can be shown that the transitive closure of a binary relation R cannot be expressed in GNFP [Benedikt et al., 2016b].

Proposition 26. GNFP-UP *is strictly more expressive than* GNFP.

Parameters of a formula. If Y identifies a fixpoint with parameters $\vec{z}$, then params$(Y) := \vec{z}$, the *parameters associated with the second-order variable* Y.

We use Free(φ) to denote the *free first-order variables* in φ. It is defined recursively. For atoms $R\vec{t}$ with $R \in \sigma$ and $\vec{t}$ a tuple consisting of constants and variables, the free first-order variables are just the variables in $\vec{t}$. For $Y\vec{t}$ with Y a second-order variable, Free$(Y\,\vec{t})$ is the union of the variables in $\vec{t}$ *and* params(Y).

For boolean connectives, $\mathsf{Free}(\varphi_1 \wedge \varphi_2) = \mathsf{Free}(\varphi_1 \vee \varphi_2) = \mathsf{Free}(\varphi_1) \cup \mathsf{Free}(\varphi_2)$, and $\mathsf{Free}(\neg\varphi) = \mathsf{Free}(\varphi)$. For quantification, $\mathsf{Free}(\exists x.\varphi) = \mathsf{Free}(\varphi)\setminus\{x\}$. Finally, for $[\mu^{\vec{z}}_{Y,\vec{y}}\,.\,\varphi](\vec{t})$, the free first-order variables consist of the parameter variables $\vec{z}$ together with the variables in $\vec{t}$.

The parameters in φ consists of the union of params(Y) for all second-order variables Y occurring in φ; we let params(φ) denote the *subset of these parameters that occur free in* φ.

Normal form. We give a normal form for GNFP-UP. Analogous to the normal forms for GNF and GNFP, it highlights the fact that GNFP-UP formulas can be built up from UCQ-shaped formulas using guarded negation and guarded fixpoints with parameters.

Formally, a *normal form* GNFP-UP$[\sigma]$ *formula* φ or ψ is generated recursively from the following grammars:

$$\begin{aligned}\varphi &::= \textstyle\bigvee_i \exists\vec{y}_i.\, \bigwedge_j \psi_{ij}\\ \psi &::= R\,\vec{t} \;\mid\; Y\,\vec{t} \;\mid\; \alpha \wedge \neg\varphi \text{ where } \mathsf{Free}(\varphi) \subseteq \mathsf{Free}(\alpha) \;\mid\\ &\qquad [\mu^{\vec{z}}_{Y_m,\vec{y}_m}.S](\vec{t})\end{aligned}$$

where $\vec{t}$ is a tuple of variables or constants, $R\vec{t}$ and α are atoms using a relation in σ or $=$, and S is a system with equations of the form $Y, \vec{y} := \mathrm{gdd}(\vec{y}) \wedge \varphi(\vec{y}, \vec{z}, \vec{Y}, \vec{Z})$, as described earlier.

Any GNFP-UP formula can be converted into normal form in a canonical way. The *width* of a GNFP-UP formula is the maximum number of free variables used in any subformula after the formula is converted into normal form. We write $(\text{GNFP-UP})^k[\sigma]$ for GNFP-UP formulas of width k.

In order to help study the power of GNFP-UP in more detail, we first define a way to measure how the parameters are used. Roughly speaking, the *parameter depth* is the maximum number of nested parameter changes. We define $\mathrm{pdepth}_{\vec{z}}(\eta)$ inductively as follows:

$$\begin{aligned}&\mathrm{pdepth}_{\vec{z}}(R\,\vec{t}) = \mathrm{pdepth}_{\vec{z}}(Y\,\vec{t}) := 0\\ &\mathrm{pdepth}_{\vec{z}}(\alpha \wedge \neg\varphi) := \mathrm{pdepth}_{\vec{z}\cap\mathsf{Free}(\varphi)}(\varphi)\\ &\mathrm{pdepth}_{\vec{z}}(\textstyle\bigvee_i \exists\vec{x}_i \;\bigwedge_j \psi_{ij}) := \max_i p_i \text{ s.t.}\\ &\qquad p_i := \begin{cases} 1 + \max_j \mathrm{pdepth}_{\mathrm{params}(\psi_{ij})}(\psi_{ij}) & \text{if } \exists j \;\; \mathrm{params}(\psi_{ij}) \not\subseteq \vec{z}\\ \max_j \mathrm{pdepth}_{\vec{z}}(\psi_{ij}) & \text{otherwise}\end{cases}\\ &\mathrm{pdepth}_{\vec{z}}([\mu^{\vec{z}'}_{X_m,\vec{x}_m}.S](\vec{t})) :=\\ &\qquad \begin{cases} 1 + \max_{\varphi_j\in S} \mathrm{pdepth}_{\mathrm{params}(\varphi_j)}(\varphi_j) & \text{if } \exists j \;\; \mathrm{params}(\varphi_j) \not\subseteq \vec{z}\\ \max_{\varphi_j\in S} \mathrm{pdepth}_{\vec{z}}(\varphi_j) & \text{otherwise}\end{cases}.\end{aligned}$$

The parameter depth $\mathrm{pdepth}\varphi$ for normal form $\varphi \in$ GNFP-UP is just $\mathrm{pdepth}_{\mathsf{Free}(\varphi)}(\varphi)$. For φ not necessarily in normal form, we define it to be the pdepth after converting to normal form.

Observe that a formula that does not use any parameters has pdepth 0. Even a formula that does use parameters can have pdepth 0 if all of its parameters actually come from free variables of the formula. This is because parameters like this can be viewed as constants, since they have a fixed interpretation in any structure. Because of this, if $\varphi \in \text{GNFP-UP}[\sigma]$ with $\text{pdepth}(\varphi) = 0$ and $\text{params}(\varphi) = \vec{z}$, then we can view φ as a GNFP formula without parameters, over the signature σ extended with extra constants $\vec{z}$.

In general, the pdepth increases when we pass through a subformula that introduces more parameters. This can happen when passing through existential quantification that introduces a variable that is later used as a parameter (see the third case in the pdepth definition), or it can happen when passing through a fixpoint definition that introduces a fixpoint variable that is later used as a parameter (see the fourth case).

Later, we will see that the parameter depth is the major factor impacting the complexity of satisfiability testing.

We have mentioned that GNFP-UP subsumes GNFP. We can now say more precisely that GNFP formulas, and thus GNF formulas, can be expressed as GNFP-UP formulas of pdepth 0.

11.2 Decidability via the tree model property

Our aim will be to show that satisfiable $(\text{GNFP-UP})^k$ sentences have tree-like models. We will focus on sentences without constant symbols, where the statement is identical to the ones for previous logics:

Proposition 27. *Every satisfiable* $(\text{GNFP-UP})^k$ *sentence has a model of tree-width at most* k*.*

The proof of this proposition re-uses the infrastructure built for ordinary GNFP. We will use the same notion of bisimulation and the same notion of unravelling. The only thing we need to show is that the notion of bisimulation used for GNFP^k also preserves $(\text{GNFP-UP})^k$. For this it suffices to show that if we restrict to models of size κ, then GNFP-UP is contained in infinitary logic (with κ-sized disjunctions and conjunctions) over GNF:

Proposition 28. *For all* $\varphi \in (\text{GNFP-UP})^k[\sigma]$ *and for all cardinals* κ*, there is* $\varphi' \in \mathsf{GNF}^k_\infty[\sigma]$ *such that for all structures* $\mathfrak{A}$ *of cardinality at most* κ*,* $\mathfrak{A} \models \varphi$ *iff* $\mathfrak{A} \models \varphi'$*.*

Proof. Consider $\psi := [\mu^{\vec{z}}_{Y,\vec{y}}.\, \text{gdd}(\vec{y}) \wedge \psi'](\vec{x})$ for $\psi' \in \mathsf{GNF}^k_\infty[\sigma]$. Then for each ordinal β, the β-approximant to the fixpoint defined by ψ can be expressed in GNF_∞, while preserving the width. This is easily shown by transfinite induction on β.

Now given $\varphi \in (\text{GNFP-UP})^k[\sigma]$ and κ, we work from the inside out, replacing each fixpoint definition with its $\kappa + 1$ approximant, an upper bound on the closure ordinal where the fixpoint is reached in structures of cardinality at most κ. □

GNFP-UP is contained in GSO just as GNFP is, and thus we can use the forward mapping theorem, Theorem 10, to get a translation to a tree coding. Thus combining the tree-like model property of Proposition 27, the forward mapping, and Rabin's theorem, we get:

Theorem 26. *Satisfiability for* GNFP-UP *is decidable.*

11.3 Decidability of GNFP-UP using automata

In this section, we show how to convert a formula φ in GNFP-UP into an automaton. Our notion of automaton will be a 2-way alternating parity automaton on binary trees, as was used for GNFP.

For GNFP, we could directly define the localized versions of the automata using a state set of size at most singly exponential in the size of the input formula. However, by adding parameters in GNFP-UP, this direct definition of a localized version becomes more challenging. We are forced to construct non-localized automata at some points — namely, for subformulas that introduce new parameters — and then apply Theorems 14 and 15, resulting in an exponential blow-up. The parameter depth is a measure of how many of these blow-ups occur.

Construction. We will assume our sentence is in the normal form for GNFP-UP defined earlier. We now describe more details of the construction of an automaton for normal form $\varphi \in$ GNFP-UP. Recall that, it is straightforward to construct an automaton that checks consistency. Hence, we can concentrate on defining an automaton for φ that runs on consistent trees and accepts iff the decoding of the consistent input tree actually satisfies φ.

The main theorem states that the size of the automaton for φ is a tower of exponentials whose height depends on the pdepth. Given a function f, we write $\exp_f^n(m)$ for a tower of exponentials of height n based on f, i.e. $\exp_f^0(m) = m$ and $\exp_f^n(m) = 2^{f(\exp_f^{n-1}(m))}$.

Theorem 27. *For normal form* $\varphi \in$ (GNFP-UP)$^k[\sigma]$ *with* $pdepth(\varphi) \geq 1$*, we can construct a 2-way alternating parity tree automaton* $\mathcal{A}_\varphi$ *such that for all consistent code-trees* t*,* $\mathrm{decode}(t) \models \varphi$ *iff* $t \in L(\mathcal{A}_\varphi)$*, and the size of* $\mathcal{A}_\varphi$ *is at most* $(pdepth(\varphi)+1)$*-exponential in* $|\varphi|$*.*

More precisely, there is a polynomial function f *independent of* φ *such that the size is at most* $exp_f^{pdepth(\varphi)}(f(m) \cdot 2^{f(klr)})$ *where* $m = |\varphi|$*,* $l = |\mathsf{Const}\sigma|$*, and* $r = \mathrm{rank}_{\mathrm{CQ}}(\sigma)$ *(see definitions below).*

The main factor affecting the output size is the pdepth, since this determines the height of the tower of exponentials. However, for more precise bounds, the other factors affecting the size are the *size of the formula* φ, the *width* k, the number of *constants* in σ, and the *CQ-rank*. As with GNF and GNFP, the CQ-rank is the maximum number of conjuncts ψ_i in any CQ-shaped subformula $\exists \vec{x}. \bigwedge_i \psi_i$ for non-empty $\vec{x}$.

The proof of Theorem 27 is by induction on $|\varphi|$, and constructs localized 2-way automata for subformulas of φ. For GNFP subformulas, we already know how to do this (see Lemma 9).

We use these automata for GNFP as building blocks for our GNFP-UP construction. Recall that formulas of pdepth 0 can always be viewed as GNFP formulas. We can also transform parts of the formula into GNFP formulas over a slightly different signature.

For this purpose, given $\psi \in (\text{GNFP-UP})^k[\sigma]$ with $\text{params}(\psi) \subseteq \vec{z}$, define the *augmented signature* $\sigma_{\vec{z},\psi}$ to be the signature σ together with additional constants $z \in \vec{z}$ and subformula predicates F_η for subformulas η with $\text{params}(\eta) \subseteq \vec{z}$. For such η, the arity of F_η is usually $|\mathsf{Free}(\eta) \setminus \text{params}(\eta)|$; in the special case that η is a fixpoint formula, then the arity of F_η is the arity of this fixpoint predicate. Then we can transform the outer part of a GNFP-UP formula to a GNFP formula over this augmented signature. This transformation is analogous to the trick we used for UCQ-shaped formula in GNFP. There we transformed the outer part of the formula, which we called the "UCQ skeleton", to an automaton, and plugged in local automata for the subformulas.

The difference is that we can only perform this transformation on the outer part of the formula that uses the same set of parameters. Consider $\eta \in (\text{GNFP-UP})^k[\sigma]$ with $\mathsf{Free}(\eta) \subseteq \vec{y}\vec{z}$ and $\text{params}(\eta) \subseteq \vec{z}$. We define $\text{transform}_{\vec{z}}(\eta) \in \mathsf{GNFP}^k[\sigma_{\vec{z},\eta}]$ inductively as follows:

$$\text{transform}_{\vec{z}}(R\,\vec{t}) := R(\vec{t}) \qquad \text{transform}_{\vec{z}}(Y\,\vec{t}) := Y(\vec{t})$$

$$\text{transform}_{\vec{z}}(\alpha \wedge \neg\varphi) := \alpha \wedge \neg\text{transform}_{\vec{z}\cap\mathsf{Free}(\varphi)}(\varphi)$$

$$\text{transform}_{\vec{z}}([\mu^{\vec{z}'}_{X,\vec{x}}.S](\vec{t})) :=$$

$$\begin{cases} F_{[\mu^{\vec{z}'}_{X,\vec{x}}.S](\vec{t})}\,(vect) \text{ if there is } \varphi_j \in S \text{ with } \text{params}(\varphi_j) \not\subseteq \vec{z} \\ [\mu_{X,\vec{x}}.S'](\vec{t}) \text{ otherwise} \end{cases}$$

where S' is the result of applying $\text{transform}_{\vec{z}}$ to each $\varphi_j \in S$

$$\text{transform}_{\vec{z}}(\textstyle\bigvee_i \exists\vec{x}_i.\, \bigwedge_j \psi_{ij}) :=$$

$$\begin{cases} F_{\bigvee_i \exists\vec{x}_i.\, \bigwedge_j \psi_{ij}}\,\vec{y} \text{ if there is } i,j \text{ such that } \text{params}(\psi_{ij}) \not\subseteq \vec{z} \\ \bigvee_i \exists\vec{x}_i.\, \bigwedge_j \text{transform}_{\vec{z}\cap\mathsf{Free}(\psi_{ij})}(\psi_{ij}) \text{ o.w.} \end{cases}$$

This transformation does not increase the width, CQ-rank, or the size of the formula.

The GNFP formula φ' obtained using this transformation is "equivalent" to the GNFP-UP formula φ, under the assumption that the additional predicates in the augmented signature are interpreted in the expected way: F_η holds for a particular binding of the free variables and its parameter variables exactly when η holds of these. To make the output φ' truly equivalent to φ, we would have to add "defining formulas" for each new symbol F_η. We could do this for GNF formulas, since the resulting definitions would be in GNF and the formula resulting from conjoining φ' and the definitions would be a normal form for GNF in terms of UCQs and GFP formulas. This "Scott Normal Form" approach

matches the technique for GNF satisfiability in [Bárány et al., 2015]. However for GNFP-UP we will stick with φ' and then "bake in" the semantics of the formulas F_η in the translation to automata. This will make it easier to track how the blow-up occurs.

If the transformation applied to η only introduces $F_{\eta'}$ for strict subformulas η' of η, then we say the transformation is *helpful for* η. We can see from the inductive definition of the transformation that it is unhelpful exactly when we reach a fixpoint or UCQ-shaped formula where the parameters of an immediate subformula differ from the parameters of the formula itself. If a transformation is helpful for η, all occurrences of these new predicates $F_{\eta'}$ appear under a guard of $\mathsf{Free}(\eta') \setminus \mathrm{params}(\eta')$. Another way to understand the parameter depth is to say that the parameter depth measures the number of unhelpful breakpoints we reach as we try to transform the entire formula using this operation.

The main idea in the construction, described in Lemma 13 below, is to transform the outer part of the formula into a GNFP formula. If the transformation is helpful, we can then use the GNFP automaton for the outer part of the formula, and plug in inductively defined automata checking the subformulas. When this is not possible, we must use different techniques, which result in an exponential blow-up at these stages.

Lemma 13. *Let $\varphi(\vec{y}, \vec{z}, \vec{Z})$ be a subformula of $\varphi \in (\text{GNFP-UP})^k[\sigma]$ with $\mathrm{params}(\varphi) \subseteq \vec{z}$. For each local assignment $\vec{b}/\vec{y}$, we can construct a 2-way alternating parity tree automaton $\mathcal{B}_\varphi^{\vec{b}/\vec{y}}$ such that for all consistent code-trees $(t, \vec{z}^{\rightarrow}, \vec{Z}^{\rightarrow})$ and for all nodes $w \in \mathsf{domain}(t)$ with $\vec{b} \subseteq \mathit{names}(w)$,*

$$\mathfrak{D}(t), [w, \vec{b}], \vec{z}, \vec{Z} \models \varphi \quad \textit{iff} \quad \mathcal{B}_\varphi^{\vec{b}/\vec{y}} \textit{ accepts } (t, \vec{z}^{\rightarrow}, \vec{Z}^{\rightarrow}) \textit{ from } w.$$

For $\mathrm{pdepth}_{\vec{z}}(\varphi) \geq 1$, there is a polynomial function f independent of φ such that the size of all such localized automata is at most $\mathit{exp}_f^{\mathrm{pdepth}_{\vec{z}}(\varphi)}(f(mn) \cdot 2^{f(klr)})$ where $m = |\varphi|$, $n = |\sigma|$, $l = |\mathsf{Const}\sigma|$, and $r = \mathrm{rank}_{\mathrm{CQ}}(\varphi)$. The number of priorities is linear in $|\varphi|$. For $\mathrm{pdepth}_{\vec{z}}(\varphi) = 0$, the bounds match those for GNFP (see Lemma 9).

Proof sketch. The proof is by induction on $|\varphi|$.

Assume $\varphi' := \mathrm{transform}_{\vec{z}}(\varphi)$ and the transformation is helpful at φ. This always holds for the smallest (atomic) formulas, so this covers the base of the induction. We construct $\mathcal{B}_\varphi^{\vec{b}/\vec{y}}$ to simulate the automaton $\mathcal{A}_{\varphi'}^{\vec{b}/\vec{y}}$ formed from the GNFP formula φ, using Lemma 9, while allowing Eve to guess valuations for the F_η relations from φ'. Since the transformation is helpful at φ, we know that every F_η relation in φ' is for some formula η that is strictly smaller than φ, and hence the inductive hypothesis ensures there is a corresponding automaton for every suitable local assignment. During the simulation of $\mathcal{A}_{\varphi'}^{\vec{b}/\vec{y}}$, if Eve asserts $F_{\eta(\vec{x})}(\vec{a})$ at w for some $\vec{a} \subseteq \mathrm{names}(w)$, then Adam can challenge this by launching the localized automaton for the intersection of $\mathcal{B}_{\eta(\vec{x})}^{\vec{a}/\vec{x}}$ and $\mathcal{B}_{\mathrm{gdd}(\vec{x})}^{\vec{a}/\vec{x}}$ from w; likewise, if Eve does not assert $F_{\eta(\vec{x})}(\vec{a})$ at w for some $\vec{a} \subseteq \mathrm{names}(w)$, then

Adam can challenge this by launching the localized automaton for the dual of the intersection of $\mathcal{B}^{\vec{a}/\vec{x}}_{\eta(\vec{x})}$ and $\mathcal{B}^{\vec{a}/\vec{x}}_{\mathrm{gdd}(\vec{x})}$ from w. Correctness follows from Lemma 9, and the fact that the inductive hypothesis ensures the subautomata for F_η in φ' are correct. There is no exponential blow-up in this case.

Next, assume that $\varphi' := \mathrm{transform}_{\vec{z}}(\varphi)$ and the transformation is unhelpful at φ. There are two possible cases, depending on whether we are dealing with a UCQ-shaped formula or a fixpoint. But each of these will result in an exponential blow up.

The first case is when φ is a UCQ-shaped formula $\bigvee_i \exists \vec{x}_i. \bigwedge_j \psi_{ij}$ where variables from some $\vec{x}_i$ are used as parameters in $\bigwedge_j \psi_{ij}$. To start, we consider each CQ-shaped formula separately, so fix some $\exists \vec{x}_i. \bigwedge_j \psi_{ij}$. We use the inductive hypothesis to obtain 2-way alternating automata for each conjunct ψ_{ij}, using the empty local assignment. This is possible since the size of these conjuncts must be strictly less than the size of φ. Because we are using the empty local assignment, these automata operate on trees with markers for all of the free variables: $\vec{x}_i \cup \vec{y} \cup \vec{z}$. Take the intersection of these automata using the closure properties of automata mentioned in Proposition 16. Then take the intersection with the automaton checking consistency. This yields a 2-way alternating automaton corresponding to $\bigwedge_j \psi_{ij}$. We then convert this automaton to a nondeterministic version using Theorem 14 and project away the information about $\vec{x}_i$ using the projection operator mentioned in Proposition 18. This yields an equivalent (1-way) nondeterministic parity automaton for the CQ. Next, we localize this automaton to the desired variables $\vec{y}$ using Theorem 15. Finally, to construct the automaton $\mathcal{B}^{\vec{b}/\vec{y}}_{\varphi}$, we take the union of the individual CQ automata, using union closure as in Proposition 16. The conversion from a 2-way alternating automaton to a nondeterministic automaton is the costly step in this process, resulting in an exponential blow-up. This matches the claimed size bound since $\mathrm{pdepth}\psi_{ij} < \mathrm{pdepth}_{\vec{z}}(\varphi)$.

The second case is when φ is a fixpoint formula where fixpoint variables are used as parameters in the body of the fixpoint. Suppose it is of the form $[\mu^{\vec{z}'}_{X,\vec{x}}. \mathrm{gdd}(\vec{x}) \wedge \chi(\vec{x}\vec{z}', X\vec{Z})](\vec{t})$ where $\mathrm{params}(\chi) \cap \vec{x} \neq \emptyset$, so $\mathrm{params}(\chi) \not\subseteq \vec{z}$; the construction is similar for a simultaneous fixpoint. The formula χ in the body of the fixpoint is strictly smaller, so we can apply the inductive hypothesis to get an automaton for this part. We want this automaton to be localized to $\vec{x}$ so we can test for tuples in the fixpoint by launching copies of the automaton for some local assignment. However, the inductive hypothesis does not directly yield this, since some variables from $\vec{x}$ are used as parameters. Hence, we must use the inductive hypothesis to get a 2-way automaton for $\mathcal{B}^{\emptyset/\emptyset}_{\chi'}$, apply Theorem 14 to get an equivalent 1-way nondeterministic automaton (resulting in an exponential blow-up), and then localize to some $\vec{a}/\vec{x}$ using Theorem 15. Once we have these localized automata $\mathcal{B}^{\vec{a}/\vec{x}}_{\chi}$ for the body of the fixpoint, we can construct an automaton that captures the fixpoint game described in Section 9. □

Theorem 27 easily follows from this lemma.

The conversion to normal form from an arbitrary GNFP-UP sentence can be done blowing up the size at most exponentially while keeping the width and CQ-rank linear. Hence, we have the following corollary.

Corollary 4. *For $\varphi' \in$ GNFP-UP$[\sigma]$ (not necessarily in normal form), we can construct an automaton $\mathcal{A}_{\varphi'}$ of $(pdepth\varphi' + 1)$-exponential size.*

Because of the tree-like model property for GNFP-UP, we can use the automaton construction and EXPTIME emptiness testing of two-way alternating parity automata (Proposition 11) to get a finer analysis of the satisfiability of GNFP-UP:

Theorem 28. *Satisfiability for $\varphi \in$ GNFP-UP is decidable in $(pdepth\varphi + 2)$-*EXPTIME.

11.4 Bibliographical remarks and further reading

The language GNFP-UP is defined in [Benedikt et al., 2016b], and this section closely follows the development there. The language extends earlier Datalog-based languages [Rudolph and Krötzsch, 2013, Bourhis et al., 2015].

12 Conclusion and further reading

We have given an overview of the use of the tree-like model property for deciding satisfiability and other related computational problems on restricted logics.

Some natural questions are:

- What are the limitations of the method?
- Are there logics that are decidable where the method is not useful? If so, what other methods are available?
- The method allows us to conclude decidability, and to get bounds on the complexity of decision procedures. But can it be utilized to get useful decision procedures in practice?

For each of these questions, the answer is necessarily complicated. In this section we can only give some brief comments and give pointers to some relevant work.

Limitations. One of the chief limitations of the method of tree-like models is its reliance on infinite structures. The "witness example" produced by the method is a structure coded by an infinite tree. Thus, by itself, the method does not allow one to conclude anything about satisfiability, validity, or entailment over finite models. For GF and GNF, it turns out that the finite variant of these problems agrees with the unrestricted version. But to show this requires techniques very different from those given here. For GNFP, the finite and infinite variation of the problems disagree; but the finite variant of the problem is also decidable [Bárány et al., 2015]. Again, the arguments involved are quite distinct from those given here. For the logic GNFP-UP, it is open whether finite satisfiability is decidable.

Other methods. The tree-like model property is only one paradigm for decidability. Another set of logics that are decidable are *two-variable logics*. The most well-known of these is the *two-variable fragment of first-order logic*, FO^2. Most commonly, FO^2 is built up from a signature consisting of unary and binary relations only. Every formula has variables from x, y, and formulas are built up from atomic formulas by the boolean operations and the quantifications

$$\varphi(x) := \exists y\ \varphi_1(x, y)$$
$$\varphi(y) := \exists x\ \varphi_1(x, y)$$

Universal quantification can be expressed in a similar way.

Decidability of FO^2 is shown via the finite model property [Mortimer, 1975, Grädel et al., 1997a]. The key idea is that to understand FO^2, it suffices to understand objects called atomic 1-types and atomic 2-types. A 1-type is just a complete description of the atomic formulas true of a single element, while 2-type is similarly a complete description of the atomic formulas true of a pair

of elements. An FO^2 formula can only express which 1-types exist, and how 1-types connect with one another via 2-types. The idea behind the finite model property of FO^2 is that if we have any model of a formula φ, we can look at the abstract connection of its 1-types via 2-types, and re-arranged them in a way to use only finitely many elements. The same kind of analysis, but much more complicated, is done for extensions of FO^2: FO^2 has been shown decidable when certain symbols are restricted to be equivalence relations [Kieronski and Otto, 2005]. A more powerful extension is *two-variable logic with counting*, C^2 where existential quantification is of one of the forms:

$$\varphi(x) := \exists^{\geq n} y \ \varphi_1(x, y)$$
$$\varphi(y) := \exists^{\geq n} x \ \varphi_1(x, y)$$

where n is a number. The first form says that there are at least n elements y such that $\varphi_1(x, y)$ holds. Note that the ordinary existential quantification of FO^2 is a special case where $n = 1$, and ordinary universal quantification of FO^2 can be expressed using existential quantification and negation. For example, $\forall x \ \exists^{\geq 1} y R(x, y)$ is a sentence of C^2 stating that for every x, there is at most 1 y such that $R(x, y)$ holds.

Satisfiability of C^2 is shown decidable using the method of analyzing 1- and 2-types [Grädel et al., 1997b, Pacholski et al., 1997, Pratt-Hartmann, 2005] but the analysis is much more difficult than for FO^2.

There are other logics that are decidable using methods that are quite specific to the logic: for example, the Fluted fragment [Purdy, 1996, Pratt-Hartmann et al., 2016] or the Gödel-Kalmar-Shutte fragment [Gödel, 1933]. A good summary of many of these fragments can be found in the classic textbook of Börger, Grädel, and Gurevich [Börger et al., 1997].

Practical aspects. There are a number of implementations of satisfiability of modal logic; however, these generally do not go through the method of automata. We recommend the overview of implementation given in [Hustadt and Schmidt, 1999] for further details.

In the case of arity 2 schemas, guarded logics extend many extensions of modal logic known as *description logics*. Many implementations of reasoning algorithms for description logics exist, and they are based, either directly or indirectly, on the tree-like model property. A good starting point for learning about description logics is the handbook [Baader et al., 2003].

To our knowledge, there is not a single implementation of the automata-theoretic decidability algorithms for the remaining logics in this text. The chief issue with the automata-theoretic approach for guarded logics is the exponential dependence on the width for the formula. In the automata-theoretic approach, this is not only a worst-case bound: the label alphabet of the automaton is exponential in the width, and thus even forming the automaton is infeasible once the arity of relations is large.

An alternative approach to decidability for certain guarded logics is based on resolution, originating in [Ganzinger and de Nivelle, 1999] and extended in [de Nivelle, 1998, Kazakov and de Nivelle, 2004]. Resolution is a standard decision procedure for first-order logic, based on converting formulas into *clauses*, and applying resolution steps. A clause is formula $\forall \vec{x}(\bigvee L_i)$, where each L_i is an atomic formula or its negation. An arbitrary first-order formula can be converted into a conjunction of clauses while preserving satisfiability. A resolution step takes two clauses where there is a literal of the form $R(\vec{u})$ in one clause and a "matching" literal of the form $\neg R(\vec{v})$ in the other. The resolutioin step produces a new clause in which the matching literals are removed and the remaining literals appear. Resolution can be paramaterized by a rule for deciding which two formulas to resolve. The idea behind works such as [Ganzinger and de Nivelle, 1999] is that if the selection rule is chosen carefully, then resolution will terminate, giving a decision procedure for GF.

References

Serbe Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

Hajnal Andréka, Johan van Benthem, and István Németi. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27: 217–274, 1998.

Franz Baader and Ulrike Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69(1):5–40, 2001.

Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

Jean-François Baget, Marie-Laure Mugnier, Sebastian Rudolph, and Michaël Thomazo. Complexity boundaries for generalized guarded existential rules, 2011a. Research Report LIRMM 11006.

Jean-François Baget, Marie-Laure Mugnier, Sebastian Rudolph, and Michaël Thomazo. Walking the complexity lines for generalized guarded existential rules. In *IJCAI*, 2011b.

Vince Bárány and Mikołaj Bojańczyk. Finite satisfiability for guarded fixpoint logic. *Information Processing Letters*, 112(10):371–375, 2012.

Vince Bárány, Georg Gottlob, and Martin Otto. Querying the guarded fragment. In *LICS*, 2010.

Vince Bárány, Balder ten Cate, and Luc Segoufin. Guarded negation. In *ICALP*, 2011.

Vince Bárány, Balder ten Cate, and Martin Otto. Queries with guarded negation. *Proceedings of VLDB*, 5(11):1328–1339, 2012.

Vince Bárány, Michael Benedikt, and Balder ten Cate. Rewriting guarded negation queries. In *MFCS*, 2013.

Vince Bárány, Balder ten Cate, and Luc Segoufin. Guarded negation. *Journal of the ACM*, 62(3), 2015.

Michael Benedikt, Balder ten Cate, and Michael Vanden Boom. Interpolation with decidable fixpoint logics. In *LICS*, 2015.

Michael Benedikt, Pierre Bourhis, Balder ten Cate, and Gabriele Puppis. Querying visible and invisible information. In *LICS*, 2016a.

Michael Benedikt, Pierre Bourhis, and Michael Vanden Boom. A step up in expressiveness of decidable fixpoint logics. In *LICS*, 2016b.

Michael Benedikt, Pierre Bourhis, and Michael Vanden Boom. Characterizing definability in decidable fixpoint logics. In *ICALP*, 2017.

Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*. Cambridge University Press, 2001.

Achim Blumensath, Thomas Colcombet, Denis Kuperberg, Pawel Parys, and Michael Vanden Boom. Two-way cost automata and cost logics over infinite trees. In *CSL-LICS 2014*, pages 16:1–16:9, 2014.

Egon Börger, Eric Grädel, and Yuri Gurevich. *The Classical Decision Problem*. Springer, 1997.

Pierre Bourhis, Markus Krötzsch, and Sebastian Rudolph. Reasonable highly expressive query languages. In *IJCAI*, 2015.

Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. In *KR*, 2008.

Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. *Journal of Web Semantics*, 14(0):57–83, 2012.

Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Containment of conjunctive regular path queries with inverse. In *KR*, 2000.

C. C. Chang and H. J. Keisler. *Model Theory*. North-Holland, 1990.

Bruno Courcelle. The monadic second order theory of graphs i: Recognisable sets of finite graphs. *Information and Computation*, 85:12—75, 1990.

William Craig. Linear reasoning. a new form of the Herbrand-Gentzen theorem. *The Journal of Symbolic Logic*, 22(03):250–268, 1957.

Giovanna D'Agostino and Marco Hollenberg. Logical Questions Concerning The mu-Calculus: Interpolation, Lyndon and Los-Tarski. *The Journal of Symbolic Logic*, 65(1):310–332, 2000.

Marcello D'Agostino, Dov M. Gabbay, Reiner HÄHnle, and Joachim Posegg. *Handbook of Tableau Methods*. Kluwer Academic Publishers, Hingham, MA, USA, 2001.

Hans de Nivelle. A resolution decision procedure for the guarded fragment. In *CADE*, 1998.

Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory*. Springer-Verlag, 1999. Second edition.

Ronald Fagin, Phokion G. Kolaitis, Renee J. Miller, and Lucian Popa. Data exchange: Semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005.

Harald Ganzinger and Hans de Nivelle. A superposition decision procedure for the guarded fragment with equality. In *LICS*, 1999.

M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

Kurt Gödel. Zum entscheidungsproblem des logischen funktionkalküls. *Monatschefte für Mathematik und Physik*, 40:433–443, 1933.

Erich Grädel. Decision procedures for guarded logics. In *CADE*, 1999a.

Erich Grädel. On the restraining power of guards. *Journal of Symbolic Logic*, 64(4):1719–1742, 1999b.

Erich Grädel. Why is modal logic so robustly decidable?, 1999c.

Erich Grädel. Guarded fixed point logics and the monadic theory of countable trees. *Theoretical Computer Science*, 288(1):129 – 152, 2002.

Erich Grädel and Martin Otto. The freedoms of (guarded) bisimulation. In *Johan van Benthem on Logic and Information Dynamics*, pages 3–31. 2014.

Erich Grädel and Igor Walukiewicz. Guarded fixed point logic. In *LICS*, 1999.

Erich Grädel, Phokion G. Kolaitis, and Moshe Y. Vardi. On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic*, 3(1):53–69, 1997a.

Erich Grädel, Martin Otto, and Eric Rosen. Two-variable logic with counting is decidable. In *LICS*, 1997b.

Erich Grädel, Colin Hirsch, and Martin Otto. Back and forth between guarded and modal logics. *ACM Transactions on Computational Logic*, 3(3):418–463, 2002.

Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata Logics, and Infinite Games: A Guide to Current Research*. Springer-Verlag, 2002.

Edith Hemaspaandra and Henning Schnoor. On the complexity of elementary modal logics. In *STACS 2008, 25th Annual Symposium on Theoretical Aspects of Computer Science, Bordeaux, France, February 21-23, 2008, Proceedings*, pages 349–360, 2008.

Ullrich Hustadt and Renate A. Schmidt. An Empirical Analysis of Modal Theorem Provers. *Journal of Applied Non-Classical Logics*, 9(4):479–522, 1999.

Neil Immerman. *Descriptive Complexity*. Springer Graduate Texts in Computer Science, 1999.

Yevgeny Kazakov and Hans de Nivelle. A resolution decision procedure for the guarded fragment with transitive guards. In *IJCAR*, 2004.

Emanuel Kieronski and Martin Otto. Small substructures and decidability issues for first-order logic with two variables. In *LICS*, 2005.

Orna Kupferman and Moshe Y. Vardi. Weak alternating automata and tree automata emptiness. In *STOC*, 1998.

Richard E. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM Journal of Computing*, 6(3):467–480, 1977.

Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004.

Christof Löding. Automata on infinite trees, December 2011. Available at `http://old.automata.rwth-aachen.de/users/loeding/inf-tree-automata.pdf`.

Bruno Marnette. Resolution and datalog rewriting under value invention and equality constraints. Technical report, 2011. `http://arxiv.org/abs/1212.0254`.

M. Mortimer. On language with two variables. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 21:135–140, 1975.

Adrian Onet. *The Chase Procedure and its Applications in Data Exchange*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

Leszek Pacholski, Wieslaw Szwast, and Lidia Tendera. Complexity of two-variable logic with counting. In *LICS*, pages 318–327, 1997.

Christos M. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

Ian Pratt-Hartmann. Complexity of the two-variable fragment with counting quantifiers. *Journal of Logic, Language and Information*, 14(3):369–395, 2005.

Ian Pratt-Hartmann, Wieslaw Szwast, and Lidia Tendera. Quine's fluted fragment is non-elementary. In *CSL*, 2016.

William C. Purdy. Fluted formulas and the limits of decidability. *Journal of Symbolic Logic*, 61(2):608–620, 1996.

M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the AMS*, 141:1–35, 1969.

Alexander Rabinovich. Composition theorems for generalized sum and recursively defined types. *Electronic Notes in Theoretical Computer Science*, 123, 2005.

Sebastian Rudolph and Markus Krötzsch. Flag & check: data access with monadically defined queries. In *PODS*, 2013.

Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computing and Systems Science*, 4(2):177–192, 1970.

Dana Scott. A decision method for validity of sentences in two variables. *The Journal of Symbolic Logic*, page 477, 1962.

Oded Shmueli. Equivalence of datalog queries is undecidable. *Journal of Logic Programming*, 15(3):231–241, 1993.

Michael Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 1st edition, 1996.

L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time (preliminary report). In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, pages 1–9, 1973.

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

Wolfgang Thomas. Languages, automata and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3. Springer-Verlag, 1997a.

Wolfgang Thomas. Ehrenfeucht games, the composition method, and the monadic theory of ordinal words. In *Structures in Logic and Computer Science, A Selection of Essays in Honor of Andrzej Ehrenfeucht*, 1997b.

Moshe Y. Vardi. Why is modal logic so robustly decidable? In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 31, pages 149–184. American Mathematical Society, 1997.

Moshe Y. Vardi. Reasoning about the past with two-way automata. In *ICALP*, 1998.

Undecidability of general first order logic

In this appendix we give a sketch of the proof of the undecidability of satisfiability for general first order logic, as mentioned in the body.

Theorem 29. *Satisfiability of first-order logic is undecidable.*

We make use of the well-known *halting problem* for Turing Machines. This problem takes as input a deterministic Turing Machine M with binary alphabet, and an input string x_0. The output is true if M halts on x_0. A diagonal argument shows that the halting problem is undecidable.

We will show that first-order logic satisfiability is undecidable by reducing the complement of the halting problem to first-order satisfiability. That is, given M and x_0 we will effectively construct a sentence φ_{M,x_0} that is satisfiable exactly when M does *not* halt on x_0. Using this reduction, we see that if we could decide first-order logic satisfiability, we could decide whether M does not halt on x_0 (and thus, by complementing, whether M does halt).

Let us recall some of the basic definitions concerning Turing Machines. A Turing Machine M over binary alphabet consists of:

- a finite set of states Q,
- an initial state $q_0 \in Q$
- a set of accepting states $A \subseteq Q$
- a transition function $\delta : 2 \times Q \to (Q \times \mathsf{Dir} \times 2)$ where $\mathsf{Dir} = \{\mathsf{Left}, \mathsf{Right}\}$.

A *configuration* of a Turing Machine consists of a finite binary sequence $s_1 \dots s_n$ representing the tape content, a state $q \in Q$, and a head position $h_i \leq n$. A *run* of a Turing Machine is a sequence of configurations $c_1 \dots$ with several properties described below.

- The state associated with configuration c_1 is q_0, the head position is 1 and the tape content of c_1 is x_0
- For each index i of the sequence c_i, c_{i+1} satisfies the transition function, in the following sense:

  Let h_i be the head position of c_i, $\mathsf{HeadVal}_i$ the value at index h_i in the tape content, and q_i the state in c_i. Let h_{i+1}, q_{i+1} be defined similarly for c_{i+1}, and let $\mathsf{HeadVal}_{i+1}$ be the value underneath h_i in the tape of c_{i+1}. The tape values of c_{i+1} are the same as those of c_i for cell positions other than h_i. For the cell position h_i, if $\delta(\mathsf{HeadVal}_i, q_i) = (q_{i+1}, \mathsf{Dir}, \mathsf{HeadVal}_{i+1})$, then the cell position at h_i is $\mathsf{HeadVal}_{i+1}$.

  The head position h_{i+1} must satisfy the following:

  If $\delta(\mathsf{HeadVal}_i, q_i) = (q_{i+1}, \mathsf{Right}, \mathsf{HeadVal}_{i+1})$ then $h_{i+1} = hi + 1$.
  If $\delta(\mathsf{HeadVal}_i, q_i) = (q_{i+1}, \mathsf{Left}, \mathsf{HeadVal}_{i+1})$ and $h_{i+1} = hi - 1$

A *non-terminating run* is a run that is infinite, where $h(i) > 0$ for all i, and where each state is in $Q - A$. A Turing Machine M does not halt on x_0 exactly when it has a non-terminating run. Note that we have defined a run to terminate if $h(i) = 0$. That is, if the tape head goes past the initial part of the tape. This is for convenience only.

Intuitively we want to build a sentence φ_{M,x_0} describing a non-terminating run of M on x_0. How can we describe such a run?

We can represent a run as a structure with relations: $<$, $\mathsf{IsZero}(x, y)$, $\mathsf{IsOne}(x, y)$, $\mathsf{HasHead}(x, y)$, $\mathsf{HasState}_q(x)$.

A *discrete linear order with initial point* is a binary relation $<$ that is a linear order with an initial element, in which every element has an immediate successor and every element other than the initial one has an immediate predecessor. This is captured by the following axioms:

$$\begin{aligned}
\forall xy\ x < y \vee y > x \vee x = y \\
\forall xyz\ x < y \wedge y < z \rightarrow x < z \\
\forall x\ \neg(x < x) \\
\exists x_0\ \neg\exists y\ y < x \wedge \\
\forall x\ \exists y\ x < y \wedge [\neg\exists z\ x < z < y] \wedge \\
\forall x\ x \neq x_0 \rightarrow \exists y\ y < x \wedge [\neg\exists z\ x < z < y]
\end{aligned}$$

Let $\succ (x, y)$ abbreviate the formula $x < y \wedge [\neg\exists z\ x < z < y]$. For any positive integer i, let $\mathsf{IsNum}_i(x)$ be the formulas defined inductively:

$$\mathsf{IsNum}_1(x) = \neg\exists y\ y < x \mathsf{IsNum}_{i+1}(x) = \exists x_i\ \succ (x_i, x) \wedge \mathsf{IsNum}_i(x)$$

We now consider the sentence φ_{M,x_0} that is a conjunction of the following:

- $<$ is a discrete linear order with initial element
- Initial configuration axioms:

$$\exists z_1 \ldots z_{m_0} \bigwedge_{1 \leq i \leq m_0} \mathsf{IsNum}_i(z_i) \wedge \bigwedge_{i|x_0(i)=1} \mathsf{IsOne}(z_0, i) \wedge \bigwedge_{i|x_0(i)=0} \mathsf{IsZero}(z_0, z_i) \wedge$$
$$\forall x\ (x > z_{m_0} \rightarrow \neg\mathsf{IsOne}(z_0, z_i) \vee \mathsf{IsZero}(z_0, z_i)$$

- Initial state axioms:

$$\exists z_0\ \mathsf{IsNum}_1(z_0) \wedge \mathsf{HasState}_{q_0}(z_0)$$

- No accepting state axiom:

$$\forall x \bigwedge_{q \in A} \neg\mathsf{HasState}_q(x)$$

- Unique state axiom:

$$\forall x \bigwedge_{q \neq q' \in Q} \neg(\mathsf{HasState}_q(x) \wedge \mathsf{HasState}_{q'}(x)$$

- Unique head axiom:

$$\forall x \forall y \forall y' \; y < y' \rightarrow \neg(\mathsf{HasHead}(x, y) \wedge \mathsf{HasHead}(x, y')$$

- Unique bit axiom:

$$\forall x \forall y \; \mathsf{IsZero}(x, y) \rightarrow \neg \mathsf{IsOne}(x, y)$$

- Transition away from the head axiom:

$$\forall x y y' \; \mathsf{HasHead}(x, y') \wedge (y < y' \vee y' > y) \rightarrow \exists x' \; \succ (x, x') \wedge (\mathsf{IsOne}(x, y) \leftrightarrow \mathsf{IsOne}(x', y)$$

- Transition right on 1 and write 0 axiom:

$$\forall x y \; \mathsf{IsOne}(x, y) \wedge \mathsf{HasHead}(x, y) \wedge \mathsf{HasState}_q(x) \rightarrow$$
$$\exists x' y' \; \succ (x, x') \wedge \mathsf{HasState}_{q'}(x') \wedge \succ (y, y') \wedge \mathsf{HasHead}(x', y') \wedge \mathsf{IsZero}(x', y')$$

  Where $\delta(q, 1) = (q', \mathsf{Right}, 0)$

- Transition right on 1 and write 1 axiom:

$$\forall x y \; \mathsf{IsOne}(x, y) \wedge \mathsf{HasHead}(x, y) \wedge \mathsf{HasState}_q(x) \rightarrow$$
$$\exists x' y' \; \succ (x, x') \wedge \mathsf{HasState}_{q'}(x') \wedge \succ (y, y') \wedge \mathsf{HasHead}(x', y') \wedge \mathsf{IsOne}(x', y')$$

  Where $\delta(q, 1) = (q', \mathsf{Right}, 1)$

- Analogous axioms for transitioning right on 0

- Transition left on 1 and write 0 axiom:

$$\forall x y y' \; \mathsf{IsOne}(x, y) \wedge \mathsf{HasHead}(x, y) \wedge \mathsf{HasState}_q(x) \wedge \succ (y', y) \rightarrow$$
$$\exists x' \; \succ (x, x') \wedge \mathsf{HasHead}(x', y') \wedge \mathsf{IsZero}(x', y')$$

  Where $\delta(q, 1) = (q', \mathsf{Left}, 0)$

- Transition left on 1 and write 1 axiom:

$$\forall x y z \; \mathsf{IsOne}(x, y) \wedge \mathsf{HasHead}(x, y) \wedge \mathsf{HasState}_q(x) \wedge \succ (y', y) \rightarrow$$
$$\exists x' \; \succ (x, x') \wedge \mathsf{HasHead}(x', y') \wedge \mathsf{IsOne}(x', y')$$

  Where $\delta(q, 1) = (q', \mathsf{Left}, 1)$

- Analogous axioms for transitioning left on 0

We claim that M has a non-terminating run if and only if φ_{M,x_0} is satisfiable.

First, suppose M has a non-terminating run $\mathsf{Tape}_i : i \in \mathbb{N}$ on input x_0. We create a structure S_M as follows:

- $<$ is the usual linear order on the non-negative integers
- $\mathsf{IsOne}(i, j)$ holds if and only if the j^{th} cell of Tape_i is one, and similarly for $\mathsf{IsZero}(i, j)$
- $\mathsf{HasState}_q(i)$ holds if and only if Tape_i is in state q
- $\mathsf{HasHead}(i, j)$ holds if and only if in Tape_i, the head is on cell j

It is easy to verify that all the axioms of φ_{M,x_0} are satisfied in S_M.

In the other direction, we claim that if φ_{M,x_0} is satisfied, then M does not terminate. Let S be a model. For any $x, e \in S$ the configuration $t_{x,e}$ is defined as follows

- the cell domain is $\{y \in S | y < e\}$,
- the cell value on an element y is one if $\mathsf{IsOne}(x, y)$ holds and zero if $\mathsf{IsZero}(x, y)$ holds, assuming these are defined. Note that by the Unique bit axiom, which must be satisfied in M, at most one of IsOne, IsZero holds.
- the state is q if and only if $\mathsf{HasState}_q(x)$ holds. This is unique by the Unique state axiom.
- the head is on y if and only if $\mathsf{HasHead}(x, y)$ holds. This is unique by the Unique head axiom.
- the cell domain is ordered by $<$

We construct a non-terminating run of M inductively, in which each configuration is of the form $t_{x,e}$ for some $x, e \in S$. For the base case, we take Tape_1 to be t_{z_0,m_0} where m_0 is the length of x_0 and z_0 is the element satisfying $\mathsf{IsNum}_1(z_0)$. Note that the axioms guarantee that such a z_0 is unique.

For the induction step, suppose we already have Tape_i, with corresponding x_i, e_i, state q_i, and head h_i, and $\mathsf{HeadVal}_i$ be the value under the head. We define Tape_{i+1} by taking the unique x_{i+1} such that $\succ (x_i, x_{i+1})$ holds, and taking e_{i+1} to be defined by:

- the unique e such that $\succ (e_i, e_{i+1})$ holds, if h_i is the final element of Tape_i, and $\delta(q_i, \mathsf{HeadVal}_i)$ is of the form (q', Right, v).
- e_i otherwise

We first need to show that each Tape_i is a well-defined Turing Machine configuration, and that for each i the sequence $\mathsf{Tape}_1 \ldots \mathsf{Tape}_i$ is a valid run. We prove the two statements by induction.

The fact that Tape_1 is well-defined configuration agreeing with the tape of M on x_0 follows from the Initial state axioms.

Now consider the induction step, where we already know that $\mathsf{Tape}_1 \ldots \mathsf{Tape}_i$ represent a valid run. Let q_i be the state of Tape_i, h_i the head position, and $\mathsf{HeadVal}_i$ the bit value under the head. δ is a total function so must be defined on $(q_i, \mathsf{HeadVal}_i)$. We do a case analysis based on the direction and bit components of the output. For example, suppose $\delta(q_i, \mathsf{HeadVal}_i) = (q', \mathsf{Right}, 0)$, and suppose also that h_i is not the final element of Tape_i. Thus the element e_{i+1} defined above is the same as e_i.

Let x_{i+1} be the unique unique such that $\succ (x_i, x_{i+1})$. By the Transition Right axioms, we know

$$\exists y' \ \wedge \mathsf{HasState}_{q'}(x_{i+1}) \wedge \succ (h_i, y') \wedge \mathsf{HasHead}(x', y') \wedge \mathsf{IsZero}(x_{i+1}, y')$$

By the unique head axiom, the only position with $\mathsf{HasHead}$ holding is the y' above, which agrees with the place that the head should be on such a transition.

By the Transition away from the head axioms, we know

$$\forall y \ (y < h_i \vee h_i > y) \rightarrow (\mathsf{IsOne}(x_i, y) \leftrightarrow \mathsf{IsOne}(x_{i+1}, y)$$

Combining this with the above, we see that the tape bits are consistent with what they should be on such a transition as well. The other cases are similar.

Finally, to show that the sequence as a whole gives a non-terminating run, we use the No accepting state axiom.

This completes the proof of Theorem 29.