Existential Fixed-Point Logic as a Fragment of Second-Order Logic

Andreas Blass

Mathematics Department, University of Michigan,
Ann Arbor, MI 48109–1043, USA
ablass@umich.edu

To Yuri Gurevich, on the occasion of his 75*th birthday.*

Abstract. The standard translation of existential fixed-point formulas into second-order logic produces strict universal formulas, that is, formulas consisting of universal quantifiers on relations (not functions) followed by an existential first-order formula. This form implies many of the pleasant properties of existential fixed-point logic, but not all. In particular, strict universal sentences can express some co-NP-complete properties of structures, whereas properties expressible by existential fixed-point formulas are always in P. We therefore investigate what additional syntactic properties, beyond strict universality, are enjoyed by the second-order translations of existential fixed-point formulas. In particular, do such syntactic properties account for polynomial-time model-checking?

1 Introduction

In [3], Yuri Gurevich and I pointed out numerous pleasant properties of existential fixed-point logic ($\exists$FPL), the logic roughly described as first-order logic, minus universal quantification, plus the least-fixed-point operator for positive inductive definitions. (This and other concepts used in this introduction are explained in more detail in Sect. 2.) In that paper, we also showed that formulas of existential fixed-point logic can be translated into equivalent formulas in a fragment of second-order logic called "strict $\forall^1_1$". Many, but not all of the pleasant properties of $\exists$FPL formulas are consequences of this translation; that is, they are enjoyed not only by $\exists$FPL formulas but by all strict $\forall^1_1$ formulas. The "not all" here refers particularly to PTime model-checking for all $\exists$FPL formulas; strict $\forall^1_1$ formulas do not all enjoy this property unless P=NP.

This situation suggests that perhaps the second-order translations of $\exists$FPL formulas actually lie in a smaller fragment of second-order logic, a subset of the strict $\forall^1_1$ fragment, such that the subset enjoys PTime model-checking. Of course, one could trivially define such a subset, namely the set of formulas that result from the standard translation procedure applied to $\exists$FPL formulas.

Partially supported by NSF grant DMS-0653696

The purpose of this paper[1] is to give a more detailed description of a subset with the desired properties.

We first show, in Sect. 3, that the model-checking problem for any ∃FPL sentence reduces to the propositional satisfiability problem for instances of a corresponding quantifier-free first-order formula. We describe the structure of these quantifier-free formulas and exploit that structure to transform these formulas, in Sects. 4 and 6, in a way that, on the one hand, does not alter the satisfiability of their instances but, on the other hand, ultimately leads to Horn formulas, so that satisfiability can be decided in polynomial time.

Along the way, the material in Sect. 5 presents an apparently new satisfiability-preserving transformation of propositional formulas in conjunctive normal form.

2 Preliminaries

2.1 Existential Fixed-Point Logic

In this subsection, we review the syntax and semantics of existential fixed-point logic.

A *vocabulary* for existential fixed-point logic (∃FPL) consists of a vocabulary in the usual sense for first-order logic (predicate symbols and function symbols with specified natural numbers as arities) plus a specification, for each predicate symbol, whether it is *positive* or *negatable*. Terms and atomic formulas are defined as in first-order logic (without equality, for simplicity). Then ∃FPL formulas of a vocabulary L are defined by the following recursion, in which we omit some parentheses to improve readability.

- Atomic formulas of L are L-formulas.
- If φ is an atomic L-formula whose predicate symbol is negatable, then $\neg\varphi$ is an L-formula.
- If φ and ψ are L-formulas, then so are $\varphi \wedge \psi$ and $\varphi \vee \psi$.
- If φ is an L-formula and x is a variable, then $\exists x\, \varphi$ is an L-formula.
- Let $L' = L \cup \{P_1, \dots, P_k\}$ be a language obtained by adding to L some k new (i.e., not already in L) positive predicate symbols P_i, say of arities r_i. Let $\mathbf{x}_1, \dots, \mathbf{x}_k$ be lists of distinct variables of lengths r_i, respectively. Let $\delta_1, \dots, \delta_k$ and φ be L'-formulas. Then

  $$\texttt{Let } P_1(\mathbf{x}_1) \leftarrow \delta_1, \dots, P_k(\mathbf{x}_k) \leftarrow \delta_k \texttt{ then } \varphi$$

  is an L-formula. Formulas of this form are called *fixed-point formulas*, the predicate symbols P_i are called the *recursion variables*, the δ_i's are called their *defining formulas*, and φ is called the *conclusion*.

[1] My talk at Yuri Gurevich's 70th birthday conference in Brno contained much of the present paper's material, but I had overlooked what I now call the conjunction problem in Sect. 3. The solution of that problem given here in Sect. 4 is new. This paper is, except for preliminary material, disjoint from my written contribution [2] to Yuri's 70th birthday celebration.

Free variables of a formula are defined as in first-order logic with the additional clause that a variable is free in the fixed-point formula

$$\texttt{Let } P_1(\mathbf{x}_1) \leftarrow \delta_1, \dots, P_k(\mathbf{x}_k) \leftarrow \delta_k \texttt{ then } \varphi$$

if either it is free in some δ_i and is not in the list $\mathbf{x}_i$ or it is free in φ.

The semantics of ∃FPL is defined like that of first-order logic, with the following additional clause for fixed-point formulas. Let θ be the fixed-point formula displayed above. Let $\mathfrak{A}$ be an L-structure with underlying set A, and let values in A for the free variables of θ be given. Consider any k-tuple $(R_1, \dots, R_k)$ of relations on A, where each R_i is r_i-ary. Let $(\mathfrak{A}, R_1, \dots, R_k)$ be the L'-structure that agrees with $\mathfrak{A}$ as an L-structure and interprets the additional predicate symbols P_i as the corresponding R_i. Each of the L'-formulas δ_j defines, in $(\mathfrak{A}, R_1, \dots, R_k)$, an r_j-ary relation S_j on A. In detail, an r_j-tuple $\mathbf{a}$ of elements of A is in S_j if δ_j is true in $(\mathfrak{A}, R_1, \dots, R_k)$ when the variables $\mathbf{x}_j$ are interpreted as $\mathbf{a}$ and the other free variables have their originally given interpretations. This construction $\mathbf{\Delta}$ sending k-tuples $(R_1, \dots, R_k)$ to k-tuples $(S_1, \dots, S_k)$ is a monotone operator on k-tuples of relations of arities r_i on A. (Monotonicity is with respect to componentwise set-thoretic inclusion; it follows from the requirement that the P_i's are positive in the δ_j's.) Let $\mathbf{\Delta}^\infty$ be the least fixed-point of this monotone operator. Then the interpretation of θ in $\mathfrak{A}$ is defined to be the interpretation of the conclusion φ in $(\mathfrak{A}, \mathbf{\Delta}^\infty)$.

Less formally, the "Let … then …" construction produces the least fixed-point of any definable positive operator on (tuples of) relations, and then uses that fixed-point in a further formula φ.

The absence of the universal quantifier would be meaningless if we allowed negation of arbitrary formulas, as one can simulate $\forall x$ with $\neg\exists x\,\neg$. This is why negation is allowed only on atomic formulas. The distinction between positive and negatable predicate symbols and the prohibition of negation on positive atomic formulas serve to ensure that the δ_i in the fixed-point formula θ above contain only positive occurrences of the recursion variables P_j and thus define a monotone operator $\mathbf{\Delta}$. They also serve to ensure that the conclusion φ in θ contains only positive occurrences of the predicates P_j; without such a restriction, we could surreptitiously introduce the negation of a positive predicate Q by writing $\texttt{Let } P(x) \leftarrow Q(x) \texttt{ then } \neg P(x)$, which would be equivalent to $\neg Q(x)$.

The definition of ∃FPL formulas is a recursion involving all vocabularies simultaneously, because fixed-point formulas of one vocabulary L can have subformulas, like the δ_i's and φ above, from a larger vocabulary L'. In effect, the additional symbols P_i of L' play the role of bound second-order variables. This connection with second-order logic will be clarified in the next subsection.

2.2 Translation to Second-Order Logic

In this subsection, we review the standard translation from ∃FPL formulas to strict $\forall^1_1$ formulas of second-order logic. As mentioned in the introduction, these are formulas obtained from existential formulas of first-oder logic by prefixing them with a string (possibly an empty string) of universal second-order quantifiers over predicate symbols.[2]

Note that there is a symbiosis between the two requirements (1) that the second-order quantifiers apply to predicate symbols and not function symbols and (2) that the first-order part of the formula be purely existential. Each of these requirements alone would be meaningless. Specifically, if we imposed only requirement (1) but allowed arbitrary first-order parts, then we could use the first-order part to say that the universally quantified predicates are the graphs of functions, thereby making (1) pointless. If, on the other hand, we imposed only requirement (2) but allowed universal quantification of functions rather than predicates, then arbitrary first-order parts could be simulated by converting them to Herbrand normal form (the dual of Skolem normal form).

We now check, by induction on ∃FPL formulas φ, that they are equivalent to strict $\forall^1_1$ formulas. This is obvious in the case of atomic or negated atomic formulas.

In the case of conjunctions and disjunctions, we write the conjuncts or disjuncts in strict $\forall^1_1$ form using different bound second-order variables, combine them with $\wedge$ or $\vee$, and pull the second-order quantifiers out as a prefix using the usual prenexing rules.

In the case of existential quantification, we pull the second-order universal quantifiers out of the scope of the new existential first-order quantifier using the logical equivalence

$$(\exists x)(\forall P)\,\varphi(P(\dots)) \iff (\forall P')(\exists x)\,\varphi(P'(x,\dots)).$$

Here the arity of P' exceeds that of P by one, and every occurrence of P in the body of the formula is changed to an occurrence of P' with the additional argument x.

Finally, in the case of fixed-point formulas, we use the fact that

$$\texttt{Let } P_1(\mathbf{x}_1) \leftarrow \delta_1, \dots, P_k(\mathbf{x}_k) \leftarrow \delta_k \texttt{ then } \varphi$$

is equivalent to

$$(\forall P_1)\dots(\forall P_k)\left[\left(\bigwedge_{i=1}^{k}(\forall \mathbf{x}_i)\,(\delta_i \implies P_i(\mathbf{x}))\right) \implies \varphi\right].$$

To see the equivalence, note that, since the P_i's occur only positively in φ, if φ holds for the intended interpretation $\mathbf{\Delta}^\infty$ of the P_i's, then it also holds for all larger relations, and, in particular, for all relations closed under the operator $\mathbf{\Delta}$. And this is precisely what the second-order formula above says: φ holds whenever the (interpretations of the) P_i are closed under the operator $\mathbf{\Delta}$ given by the defining formulas δ_i.

[2] The terminology "strict $\forall^1_1$" was chosen in analogy with "strict Π^1_1" in [1, Sect. 8.2]. The difference is that "strict Π^1_1" is used in a set-theoretic context and allows not only existential quantifiers but also bounded universal quantifiers $(\forall x \in y)$ in the first-order part of the formula.

Now if we insert into this equivalent formula some strict $\forall^1_1$ forms of the δ_i's and φ and then apply standard prenex operations, the result is in strict $\forall^1_1$ form, as desired. Note, in particular, that the universal first-order quantifiers $\forall \mathbf{x}_i$ are in the antecedent of an implication so the first-order part is existential.

The preceding proof, showing that ∃FPL formulas can be translated to equivalent strict $\forall^1_1$ formulas, would become an algorithm for carrying out the translation if we added some unimportant details, such as the choice of bound variables and the order in which similar quantifiers are pulled out during prenex operations. We assume henceforth that these details have been supplied, and we refer to the resulting algorithm as the *standard translation* from ∃FPL to strict $\forall^1_1$.

In [3] some semantical properties of ∃FPL formulas were established by showing that they actually hold for all strict $\forall^1_1$ formulas. These properties include the facts that

– The set of valid ∃FPL sentences is a complete computably enumerable set.
– The set of satisfiable ∃FPL sentences is a complete computably enumerable set.[3]
– If a formula is satisfied by some elements in a structure, then this fact depends only on a finite part of the structure.

But at least one important property of ∃FPL formulas, namely PTime model-checking, does not (unless P=NP) hold for arbitrary strict $\forall^1_1$ formulas. Specifically, on undirected graphs, regarded as structures with a single binary relation E of adjacency, the strict $\forall^1_1$ formula

$$(\forall P_1)(\forall P_2)(\forall P_3)(\exists x)(\exists y)\left[\bigwedge_{i=1}^{3} \neg P_i(x) \vee \bigvee_{i=1}^{3} (E(x,y) \wedge P_i(x) \wedge P_i(y))\right]$$

expresses that the graph is not 3-colorable, a co-NP-complete property.

This situation suggests that perhaps the second-order translations of ∃FPL formulas are not merely strict $\forall^1_1$ but have some additional syntactic property that ensures their PTime decidability. The main purpose of this paper is to establish such an additional property.

3 Model-Checking

In this section, we discuss model-checking for strict $\forall^1_1$ sentences. That is, we consider, for any fixed strict $\forall^1_1$ sentence φ in vocabulary L, the following decision problem:[4] An instance is a finite L-structure $\mathfrak{A}$ and the question is whether $\mathfrak{A} \models \varphi$.

[3] The expected duality between validity and satisfiability is not available for logics, like ∃FPL, that are not closed under negation.

[4] We are dealing here with what is often called data complexity of the model-checking problem. That is, we regard the "data" $\mathfrak{A}$ as the input, and we measure resource usage relative to the size of $\mathfrak{A}$, while the "query" φ is held fixed.

It is convenient to address this problem by considering the negation of φ instead. It has a standard translation to a strict $\exists^1_1$ sentence; that is, $\neg\varphi$ can be put into the form

$$(\exists R_1)\dots(\exists R_m)\,\psi$$

where the R_i are predicate symbols and where ψ is a universal first-order sentence of the vocabulary $L' = L \cup \{R_1, \dots, R_m\}$. This strict $\exists^1_1$ sentence is true in $\mathfrak{A}$ if and only if there are relations[5] R_i on the underlying set A such that all instances of ψ are true for this interpretation of the existentially quantified predicate variables in $\neg\varphi$. Here "instances of ψ" refers to all the formulas obtained by replacing the (universally) quantified first-order variables in ψ by arbitrary elements of A.

This criterion for $\mathfrak{A} \models \neg\varphi$ is essentially a question of propositional satisfiability. Indeed, consider the set Σ of all instances of ψ. These are quantifier-free $L'(A)$-sentences, where $L'(A)$ is the language obtained from $L' = L \cup \{R_1, \dots, R_m\}$ by adding (names for) all the elements of A as constant symbols. In these sentences, replace each atomic sentence that uses an L-predicate symbol (i.e., any predicate symbol other than the R_i's) by its truth value in $\mathfrak{A}$. What remains is a set Σ' of sentences that are Boolean combinations of instances of the R_i's. Regard all these instances of R_i's as propositional variables. Any truth assignment to these propositional variables amounts to a choice of relations R_i; the truth assignment satisfies Σ' if and only if the R_i relations satisfy ψ. Therefore $\neg\varphi$ is true in $\mathfrak{A}$ if and only if Σ' is (truth-functionally) satisfiable.

The process leading from φ and $\mathfrak{A}$ to Σ' can be summarized as follows.

1. Perform the standard translation of φ to strict $\forall^1_1$ form.
2. Negate the result and push the negation in past quantifiers and connectives, until only atomic formulas are negated; the result is the standard strict $\exists^1_1$ form of $\neg\varphi$.
3. Delete all quantifiers, but remember which predicate symbols were bound second-order variables.
4. Form all instances of the resulting formula, replacing the first-order variables by (names of) elements of $\mathfrak{A}$ in all possible ways.
5. In the resulting formulas, replace the atomic subformulas whose predicate symbols are in the vocabulary of $\mathfrak{A}$ (as opposed to the predicate symbols that were quantified second-order variables before step 3) by their truth values in $\mathfrak{A}$.

The resulting set of formulas is the propositional translation Σ' of $\neg\varphi$. Its propositional variables are of the form $R(\mathbf{a})$, where R was a bound second-order variable before step 3, and $\mathbf{a}$ is a tuple of elements of $\mathfrak{A}$.

Note that we have arranged the steps so that the input $\mathfrak{A}$ of our model-checking problem enters the process only at step 4.

The propositional translation has two key properties. First, $\mathfrak{A} \models \varphi$ if and only if the propositional translation of $\neg\varphi$ is not satisfiable. Second, the propositional

[5] To avoid excessive notation, we use the same symbols for these relations as for the corresponding symbols in our strict $\exists^1_1$ sentence.

translation is, for fixed φ, computable in polynomial time from $\mathfrak{A}$. In particular, the size of the propositional translation of $\neg\varphi$ is bounded by the product of

- $|A|^d$, where d is the number of universally quantified, first-order variables in the strict $\forall^1_1$ translation of φ, because these are the variables that must be replaced, in all possible ways, by elements of A,
- $\log|A|$ to account for the length in bits of the names of the elements of A, and
- a constant, namely the length of the formula obtained in step 3 above, before $\mathfrak{A}$ entered the process.

Thus, we have a PTime reduction of the model-checking problem for a (fixed) $\exists$FPL sentence φ to a propositional satisfiability problem. Our goal is to detect the special properties of the propositional translations of $\exists$FPL sentences that make their satisfiability decidable in PTime. For example, if the propositional translations always consisted of Horn formulas, then that would provide a PTime solution of the model-checking problem. We therefore turn our attention to the structure of the formulas that arise in the propositional translations of $\exists$FPL sentences.

Let us begin by disposing of a tempting error. When we translated $\exists$FPL formulas θ into second-order logic, the second-order variables originated from the recursion variables P_i in the $P_i(\mathbf{x}) \leftarrow \delta_i(\mathbf{x})$ parts of fixed-point formulas. Those P_i's are positive predicate symbols. So they occur only positively in the strict $\forall^1_1$ form of θ, and therefore the resulting propositional variables occur only negatively in the propositional translation of $\neg\varphi$. It is, of course, trivial to decide satisfiability of propositional formulas in which all the variables occur only negatively; just give them all the value "false" and see whether the formulas become true.

The error in the preceding paragraph is that, although the P_i are positive predicate symbols and therefore occur only positively in the defining formulas δ_i and in the conclusion φ, they nevertheless acquire negative occurrences in the strict $\forall^1_1$ translation. Specifically, the underlined occurrences in the translation

$$(\forall P_1)\dots(\forall P_k)\left[\left(\bigwedge_{i=1}^{k}(\forall \mathbf{x}_i)\,(\delta_i \implies \underline{P_i(\mathbf{x})})\right) \implies \varphi\right]$$

of a fixed-point formula are negative. So the situation is not so trivial as the preceding paragraph would suggest.

A better, but still incorrect approach involves rewriting the translation of a fixed-point formula exhibited above in the logically equivalent form

$$(\forall P_1)\dots(\forall P_k)\left[\varphi \vee \bigvee_{i=1}^{k} \exists \mathbf{x}_i(\neg P_i(\mathbf{x}_i) \wedge \delta_i)\right].$$

The negation, in strict $\exists^1_1$ form, then looks like

$$(\exists P_1)\dots(\exists P_k)\left[\neg\varphi \wedge \bigwedge_{i=1}^{k}(\forall \mathbf{x}_i)\,(P_i(\mathbf{x}_i) \vee \neg\delta_i)\right].$$

The first-order body of this formula is a conjunction of $k+1$ subformulas, each of which contains at most one positive occurrence of a P_i. When we form instances of this body, we get at most one positive literal in each conjunct. That is, we get only Horn clauses, and it is well-known that satisfiability of sets of Horn clauses is decidable in PTime.

There are two errors in this approach. The first is that ∃FPL formulas are not simply fixed-point formulas like the one under consideration here. In particular, we might have the conjunction of two (or more) such formulas. Then the propositional translation of the negation will be a disjunction of formulas like those here, and, when put into conjunctive normal form, will have two (or more) positive literals in some of its clauses. We shall address this *conjunction problem* in Sect. 4.

The second error in the argument above is that a conjunct $P_i(\mathbf{x}) \vee \neg\delta_i$ can have more positive occurrences of literals than just the visible $P_i(\mathbf{x})$. If the formula δ_i contains some fixed-point formulas as subformulas, then the second-order variables arising from those subformulas will have negative occurrences (analogous to the underlined P_i's above) in δ_i and therefore positive occurrences in the conjunct $P_i(\mathbf{x}) \vee \neg\delta_i$ under consideration. So the conjunctive normal forms of our propositional translations need not be Horn formulas.

Notice that this error is relevant only when recursions are nested, that is, when the defining formula δ_i in a recursive clause $P_i(\mathbf{x}_i) \leftarrow \delta_i$ contains further fixed-point formulas. Accordingly, we call this the *nesting problem*; we shall address it in Sects. 5 and 6.

It is known that nesting of recursions is never really needed in ∃FPL. For example, the nested recursion

$$\texttt{Let } P(x) \leftarrow [\texttt{Let } Q(y) \leftarrow \delta(P,Q,x,y) \texttt{ then } \theta(P,Q,x)] \texttt{ then } \varphi(P)$$

(where we have indicated which predicate symbols and bound variables are available in the subformulas) is equivalent to

$$\texttt{Let } P(x) \leftarrow \theta(P,Q'_x,x),\ Q'(x,y) \leftarrow \delta(P,Q'_x,x,y) \texttt{ then } \varphi(P),$$

where Q'_x means the binary predicate symbol Q' with x inserted as its first argument. For the general proof that unnested recursions suffice, see, for example, [5, Sect. 1.C].

One can similarly circumvent the conjunction problem, because conjunctions in ∃FPL formulas can be pushed inward to apply only to atomic and negated atomic formulas. For example, the conjunction

$$(\texttt{Let } P(x) \leftarrow \delta(x) \texttt{ then } \varphi) \wedge (\texttt{Let } P'(y) \leftarrow \delta'(y) \texttt{ then } \varphi')$$

is equivalent to

$$\texttt{Let } P(x) \leftarrow \delta(x), P'(y) \leftarrow \delta'(y) \texttt{ then } \varphi \wedge \varphi'$$

(where we assume that bound variables have been renamed if necessary to avoid clashes).

In a sense, these observations explain, via the strict $\forall^1_1$ translation, why ∃FPL has PTime model-checking. Explicitly: Given an ∃FPL sentence, rewrite it to avoid nested recursions and to avoid conjunctions of compound formulas. Then produce the propositional translation of the negation of the new ∃FPL sentence, using a conjunctive normal form of its matrix. The result consists of Horn clauses, for which satisfiability is decidable in PTime. (The use of the conjunctive normal form can exponentially increase the size of the formula, but this doesn't matter as we are considering a single formula at a time and measuring complexity relative to the structure $\mathfrak{A}$.)

Unfortunately, this does not quite answer our original question, which concerned the direct translation of arbitrary ∃FPL formulas to strict $\forall^1_1$ form, without pre-processing to eliminate nested recursions and non-trivial conjunctions.

Fortunately, the satisfiability problem for the sets of formulas that actually arise can be transformed, on the level of propositional logic, to an equivalent satisfiability problem for Horn formulas. In the following sections, we shall carry out this transformation. In Sect. 4, we show how to convert the formulas that actually arise to formulas that avoid the conjunction problem; the conversion preserves satisfiability. In Sects. 5 and 6, we do the same for the nesting problem. Section 5 isolates the relevant construction in general, not just for the formulas obtained by translating ∃FPL formulas; this general, satisfiability-preserving transformation seems to be of independent interest. The application of the general transformation to the nesting problem for translated ∃FPL formulas is described in Sect. 6.

Remark 1. The general theme of this paper is that model-checking for a certain class of second-order formulas is in polynomial time because it can be reduced to the propositional satisfiability problem for Horn formulas. The same theme occurs in a paper [4] of Erich Grädel. The class of second-order formulas considered there, called SO-HORN, is, however, quite different from the class arising here from ∃FPL. The appropriate comparison would be between SO-HORN and the strict $\exists^1_1$ formulas arising from the negations of ∃FPL formulas. In both cases, the second-order quantifiers range only over relation variables, and in both cases the first-order matrix is required to be a universal formula, but the smiilarity ends there. SO-HORN allows both universal and existential second-order quantifiers, whereas strict $\exists^1_1$ requires the second-order quantifiers to be existential. On the other hand, the quantifier-free parts of SO-HORN formulas are required to already be in Horn form (at least with regard to the quantified predicate symbols), whereas, as we have seen above, we must deal with non-Horn formulas. Indeed, the following sections are primarily devoted to the problem of converting our formulas to Horn form without altering the satisfiability of their instances.

4 Conjunctions

To avoid annoying distractions during our manipulations of formulas, we assume from now on that there are no clashes of variables in our ∃FPL formulas. That is, no (first-order) variable has both free and bound occurrences, nor is any such variable bound twice (by $\exists$ or by the fixed-point construction); also no predicate symbol occurs more than once as a recursion variable. This simplification can, of course, be achieved by renaming bound variables and recursion variables as necessary. We shall refer to this convention as the "no clashes" assumption.

We point out, for future reference, a consequence of the no clashes assumption. Suppose that, in some $\exists$FPL sentence φ, a certain variable x occurs in two or more fixed-point subformulas, say with recursion variables P and Q. Then the scope of that x must include both of those fixed-point subformulas. As a result, when one converts φ to strict $\forall^1_1$ form, the second-order quantifiers $\forall P$ and $\forall Q$ will be pulled out of the scope of an $\exists x$, and so the predicate symbols P and Q will have their arities increased and will have x inserted as an additional argument. Thus, if x occurs in two or more fixed-point subformulas of φ, then the recursion variables of those subformulas will, in the strict $\forall^1_1$ translation of φ, have x among their arguments. We shall refer to this observation as "argument joining".

As a first step in the solution of the conjunction problem, we describe carefully the class of formulas that arise from steps 1 to 3 in the process described above for obtaining Σ' from φ and $\mathfrak{A}$. As remarked there, these three steps do not involve $\mathfrak{A}$, which enters only at steps 4 and 5. So we are working with just an $\exists$FPL formula φ. We first produce the standard strict $\exists^1_1$ form of $\neg\varphi$, and then we delete all the quantifiers, obtaining a quantifier-free first-order formula in the vocabulary consisting of the vocabulary of φ plus amplified recursion variables from φ. Here "amplified" refers to the extra argument places that recursion variables acquire when, in the production of the strict $\forall^1_1$ form, they are pulled out of the scope of first-order existential quantifiers. By inspection of the definitions of $\exists$FPL formulas and of their standard translations to strict $\forall^1_1$ form, we see that the quantifier-free formulas obtained by this process are among the primary formulas defined as follows.

Definition 2. The *primary* formulas of a vocabulary L form the smallest class such that

- atomic formulas whose predicate symbol is negatable are primary,
- negations of arbitrary atomic formulas are primary,
- conjunctions of primary formulas are primary,
- disjunctions of primary formulas are primary, and
- if α and $\delta_1, \dots, \delta_k$ are primary formulas for the vocabulary $L \cup \{P_1, \dots, P_k\}$, where the P_i are new positive predicate symbols, then

$$\alpha \wedge \bigwedge_{i=1}^{k} (P_i(\mathbf{x}_i) \vee \delta_i)$$

is primary.

We refer to the last item in this list of constructors as the *mix constructor*, because it mixes conjunction and disjunction. Each P_i occurring there will be called a *key* predicate, and $\mathbf{x}_i$ and δ_i will be called its *associated* variables and formula, respectively.

Note that the primary formulas include all negated atomic formulas, but they include the unnegated ones only when the predicate symbol is negatable. This strange-sounding situation — the predicate must be negatable in order to appear unnegated — arises from the fact that we are working with the translations not of ∃FPL formulas themselves but of their negations. Note further that a positive predicate can have at most one positive occurrence in a primary formula, namely an occurrence as the key predicate of a mix construction. Here the "at most one" claim follows from our no clashes assumption.

In this new context, argument joining becomes the fact that, if α is a primary formula and if a variable occurs in two or more subformulas of α obtained by the mix construction, then that variable is among the arguments of the key predicates of those mix subformulas.

The next definition describes a subclass of the primary formulas for which the conjunction problem does not arise. In fact, formulas in this subclass have an especially useful structure, which we describe, in terms of their parse trees, after the definition.

Definition 3. The *basic secondary* formulas of a vocabulary L form the smallest class such that

- atomic formulas whose predicate symbol is negatable are basic secondary,
- negations of arbitrary atomic formulas are basic secondary,
- conjunctions of basic secondary formulas are basic secondary, and
- disjunctions of basic secondary formulas are basic secondary.

The *secondary* formulas of a vocabulary L form the smallest class such that

- all basic secondary formulas are secondary, and
- if α and $\delta_1, \dots, \delta_k$ are secondary formulas for the vocabulary $L \cup \{P_1, \dots, P_k\}$, where the P_i are new positive predicate symbols, then the result of the mix construction,
$$\alpha \wedge \bigwedge_{i=1}^{k} (P_i(\mathbf{x}_i) \vee \delta_i),$$
is secondary.

Thus, secondary formulas are built by the same constructors as primary formulas but, in a secondary formula, the mix constructors must be applied after all the others, not intermingled with the others.

It is useful to consider parse trees showing how secondary formulas are built from basic secondary ones. The internal nodes of such a tree correspond to the mix construction $\alpha \wedge \bigwedge_{i=1}^{k} (P_i(\mathbf{x}_i) \vee \delta_i)$; such a node has $2k+1$ children, one for α, k for the key predicate subformulas $P_i(\mathbf{x}_i)$, and k corresponding to the associated δ_i's. Of these, the k corresponding to $P_i(\mathbf{x}_i)$ are leaves of the parse tree; the other $k+1$ might be leaves or internal nodes. All the leaves of the parse tree are either of the $P_i(\mathbf{x}_i)$ form just mentioned or basic secondary formulas. Notice that the leaves of the $P_i(\mathbf{x}_i)$ sort are the only place where positive predicate symbols have positive occurrences.

The main result in this section will say that every primary formula can be transformed into a secondary one while preserving the essential property relevant for ∃FPL. That essential property is, in view of the results of Sect. 3, instance-equisatifiability, defined as follows.

Definition 4. Two sets of quantifier-free formulas Σ_1 and Σ_2 (in a first-order language that extends L) are *equisatisfiable* if, whenever there exists a truth assignment satisfying one of them, there also exists a (possibly different) truth assignment satisfying the other. They are *instance-equisatisfiable* if, for every L-structure $\mathfrak{A}$, $\Sigma_1(\mathfrak{A})$ and $\Sigma_2(\mathfrak{A})$ are equisatisfiable, where $\Sigma_i(\mathfrak{A})$ is obtained from Σ_i by replacing the variables by elements of $\mathfrak{A}$ in all possible ways and then replacing all atomic subformulas whose predicate is in L by their truth values in $\mathfrak{A}$.

Notice that the construction of $\Sigma_i(\mathfrak{A})$ from Σ_i described in this definition is exactly the last two steps, 4 and 5, in the construction of Σ' in Sect. 3. Thus, for the purpose of model-checking ∃FPL formulas, the Σ' there, which consists of primary formulas, can safely be replaced by any instance-equisatisfiable set of formulas. That is how we shall use the following proposition and its corollary.

Proposition 5. *The conjunction and disjunction of two secondary formulas are each instance-equisatisfiable with a secondary formula.*

Proof. We proceed by induction on the two given secondary formulas, and we treat the most difficult case, namely where both of them arise from the mix construction. (If both of the given formulas are basic, then the result is trivial. If one arises from mix and the other is basic, then the proof is easier, or one can regard the basic formula as resulting from a mix in which the number of key predicates happens to be zero.) Suppose, therefore, that the given formulas are

$$\gamma \;=\; \alpha \wedge \bigwedge_{i=1}^{k} (P_i(\mathbf{x}_i) \vee \delta_i) \quad \text{and} \quad \gamma' \;=\; \alpha' \wedge \bigwedge_{i=1}^{k'} (P'_i(\mathbf{x}'_i) \vee \delta'_i).$$

Their conjunction $\gamma \wedge \gamma'$ is not merely instance-equisatisfiable but tautologically equivalent with

$$(\alpha \wedge \alpha') \wedge \bigwedge_{i=1}^{k} (P_i(\mathbf{x}_i) \vee \delta_i) \wedge \bigwedge_{i=1}^{k'} (P'_i(\mathbf{x}'_i) \vee \delta'_i),$$

which is a secondary formula with $k+k'$ key predicates, because, by the induction hypothesis, $\alpha \wedge \alpha'$ is a secondary formula.

For the disjunction, we use

$$\theta \;=\; (\alpha \vee \alpha') \wedge \bigwedge_{i=1}^{k} (P_i(\mathbf{x}_i) \vee \delta_i) \wedge \bigwedge_{i=1}^{k'} (P'_i(\mathbf{x}'_i) \vee \delta'_i),$$

which is a secondary formula, as above, because $\alpha \vee \alpha'$ is secondary by induction hypothesis. It is easy to see that θ tautologically implies $\gamma \vee \gamma'$. The converse, however, is not generally correct; we do not get equivalence but only instance-equisatisfiability.

To prove the non-trivial direction of instance-equisatisfiability, suppose we have an L-structure $\mathfrak{A}$ and truth assignment v satisfying all the instances of $\gamma \vee \gamma'$, where by "instance" we understand, as in the definition of instance-equisatisfiability, the result of substituting elements of $\mathfrak{A}$ for variables and then replacing all atomic formulas involving L-symbols by their truth values in $\mathfrak{A}$. Notice that the predicate symbols in γ and γ' are all either L-symbols or key predicates of mix constructions.

To emphasize the essential idea of the proof, we first consider the special case where γ and γ' have no (first-order) variables in common. In this case, we claim that v either satisfies all instances of γ or satisfies all instances of γ' (not merely some instances of the one and the remaining instances of the other). Suppose the claim were false, so some instance of γ, and some other instance of γ' were falsified by v. Then we could form a third instance, giving the variables in γ the same values as in the first instance (thus making γ false under v) and giving the variables in γ' the same values as in the second instance (thus making γ' false under v). But then this third instance would make $\gamma \vee \gamma'$ false under v, contrary to our choice of v.

Thus, we may suppose without loss of generality, that v satisfies all instances of γ. Now we can produce a truth assignment v^* satisfying θ as follows. Let v^* assign the value "true" to all those atomic formulas whose predicate symbol is one of the key predicates P_i' of γ', and let v^* agree with v on all other atomic formulas. The difference between v and v^*, affecting only the P_i', will not affect γ, because the P_i' don't occur in γ (thanks to the no clashes assumption). Thus v^* satisfies γ and therefore satisfies the part $(\alpha \vee \alpha') \wedge \bigwedge_{i=1}^{k}(P_i(\mathbf{x}_i) \vee \delta_i)$ of θ. But it also satisfies the remaining conjunct, $\bigwedge_{i=1}^{k'}(P_i'(\mathbf{x}_i') \vee \delta_i')$ of θ. So v^* satisfies θ as required.

The preceding argument used the assumption that γ and γ' have no common variables. We now indicate how to modify it to accommodate common variables, say the list $\mathbf{y}$ of variables. We no longer claim that v satisfies all instantiations of γ or all instantiations of γ'; instead, we claim that, for any fixed instantiation of $\mathbf{y}$, v satisfies all its extensions to instantiations of γ or all its extensions to instantiations of γ'. In other words, whether γ or γ' is satisfied (by v) may depend on the instantiation but only via the values assigned to the common variables $\mathbf{y}$. The proof of this modified claim is exactly as in the easier argument given above; once we fix the values of $\mathbf{y}$, the remaining variables, occurring in only one of γ and γ', can be treated as before.

In the easier argument, we obtained v^* by modifying the truth values assigned by v to the key predicates of γ', under the assumption that v satisfied all instances of γ. Of course, if v had satisfied all instances of γ', then we would have modified the truth values assigned to the key predicates of γ. Now in the present, more complicated situation, the decision as to which predicates should get new truth values may depend on the values assigned to $\mathbf{y}$. Thanks to argument joining, this is no problem. The variables $\mathbf{y}$ occur as arguments of all the key predicates in γ and in γ'. So we can modify the values assigned to instances of P_i with certain values for the $\mathbf{y}$ arguments and modify the values assigned to instances of P_i' with other values for the $\mathbf{y}$ arguments. What we do with one instantiation of $\mathbf{y}$ has no effect on what happens with other instantiations. (Another way to view this argument is that we treat the variables $\mathbf{y}$ as new constant symbols and consider separately all the expansions of $\mathfrak{A}$ giving values to these new constants. That reduces the problem to the easier case already treated.)

This completes the proof of the proposition.

Corollary 6. *Every primary formula is instance-equisatisfiable with a secondary formula.*

Proof. Induction on primary formulas, using Proposition 5 for the only nontrivial cases.

The proofs of the proposition and corollary provide an explicit algorithm for converting a primary formula to an instance-equisatisfiable secondary one.

5 Satisfiability and Trimming

This section is entirely about propositional logic, specifically about satisfiability of sets of clauses. Here "clause" means, as usual, a disjunction of literals, i.e., of propositional variables and negations of propositional variables. So a set of clauses is semantically equivalent to a conjunctive normal form, namely the conjunction of its clauses.

Theorem 7. *Let Γ be a set of clauses, let p be a propositional variable, and let Q be a set of propositional variables other than p. Suppose that, whenever a clause in Γ contains a negative occurrence of a variable from Q, it also contains a positive occurrence of p or of some variable from Q. Obtain Γ' from Γ by deleting positive occurrences of p from those clauses that also contain positive occurrences of at least one variable from Q. Then Γ is satisfiable if and only if Γ' is.*

Proof. One direction is trivial, because any truth assignment that satisfies all the clauses in Γ' will certainly satisfy the corresponding clauses in Γ, since the latter differ from the former at most by having additional disjuncts.

Suppose, therefore, that we have a truth assignment v that satisfies all the clauses in Γ. If it makes p false, then it also satisfies all the clauses in Γ', because the positive occurrences of p that were removed when we produced Γ' were not satisfied by v and so some other disjuncts in those clauses must have been satisfied.

So we may assume that v makes p true. In this situation, v need not satisfy Γ', but we can find another truth assignment v' that will satisfy Γ'. Let v' make all the variables in Q true, and let it agree with v on all the other variables. To show that v' satisfies every clause γ in Γ', we consider three cases.

First, suppose γ is one of the clauses that was altered, by removing the positive disjunct p, when we transformed Γ to Γ'. Recall that we undertook such a removal only when the clause in Γ contained, along with p, a positive occurrence of some variable from Q. Such variables are true under v', and therefore our clause γ is also true under v'.

It remains to consider those clauses γ that were not changed in the transition from Γ to Γ'. These were true under v, but we need that they are true under v'. That is very easy to check for those clauses γ in which variables from Q occur only positively. Since, in going from v to v', the only changes were that variables in Q, which might have been false under v, became true under v', any clause containing them only positively cannot change from true under v to false under v'.

There remain those clauses γ that are the same in Γ and in Γ' but have negative occurrences of some variable(s) from Q. By the hypothesis of the theorem, every such γ also has positive occurrences of p or of some variable from Q. Since p and all variables from Q are true under v', it follows that v' satisfies all such clauses γ.

Definition 8. The transformation from Γ to Γ' described in the theorem is called *trimming* Γ or, in more detail, trimming p using Q.

6 Trimming to Horn Form

In this section, we complete the reduction of the model-checking problem for any ∃FPL sentence to a decidable case of the propositional satisfiability problem, namely the satisfiability of sets of Horn clauses.

Given an ∃FPL sentence φ, we saw in Sect. 3 how to reduce the problem "Given $\mathfrak{A}$, decide whether φ is true in $\mathfrak{A}$" to the problem of satisfiability of the set of all $\mathfrak{A}$-instances of a certain formula constructed from φ. That formula is primary, in the sense defined in Sect. 4 and, as proved there, instance-equisatisfiable with a certain secondary formula. So the model-checking problem for φ is reduced to determining the satisfiability of the instances of this secondary formula.

The next step is to convert this secondary formula, say ψ, into conjunctive normal form. (As mentioned earlier, the possible exponential increase in the formula's size caused by this conversion is not a problem, because we are considering the model-checking problem for a fixed formula, with only the structure $\mathfrak{A}$ as input. The set of all $\mathfrak{A}$-instances still has size polynomial in the size of $\mathfrak{A}$.) We now look into the structure of this conjunctive normal form and its $\mathfrak{A}$-instances (for arbitrary $\mathfrak{A}$).

Recall that, when forming $\mathfrak{A}$-instances of ψ, we replace any atomic subformulas that use predicates from L by their truth values in $\mathfrak{A}$. The atomic subformulas of an instance therefore use only the predicates not in L, which are the recursion variables of φ and the key predicates of the primary and secondary formulas derived from φ. We shall use the letters P and Q to stand for such predicates in the following discussion.

Let us consider the conjunctive normal form of a secondary formula ψ, paying particular attention to the positive occurrences of atomic subformulas using the P, Q predicates. Basic secondary formulas are built using $\wedge$ and $\vee$ from atomic subformulas, subject to the condition that positive predicates — which include the P, Q predicates — occur only negatively. So there are not yet any positive occurrences of P's and Q's at this basic stage. That situation changes when we apply the mix construction to produce secondary formulas of the form

$$\alpha \wedge \bigwedge_{i=1}^{k} (P_i(\mathbf{x}_i) \vee \delta_i).$$

Now each P_i has a positive occurrence, and, in addition, other P's and Q's may have positive occurrences in α and in the δ_i's. To convert $\alpha \wedge \bigwedge_{i=1}^{k}(P_i(\mathbf{x}_i) \vee \delta_i)$ to conjunctive normal form, we would first convert α and the δ_i's to conjunctive normal form and then, for each i, distribute $P_i(\mathbf{x}_i) \vee -$ across the conjunctive normal form of δ_i. Thus, each conjunct of that conjunctive form acquires $P_i(\mathbf{x}_i)$ as a new literal. Note that such a conjunct may already have other positive occurrences of other P's and Q's, but only when these are the key predicates of subformulas of δ_i. Thus, those other P's and Q's come from mix subformulas that are descendants of the current mix formula $\alpha \wedge \bigwedge_{i=1}^{k}(P_i(\mathbf{x}_i) \vee \delta_i)$ in the parse tree of ψ. We emphasize that, in this conjunct, the new positive P and previously present Q are key predicates of comparable mix nodes of the parse tree.

Repeating this process for every application of the mix constructor in ψ, we arrive at a conjunctive normal form θ in which every individual conjunct has, because of the comparability noted at the end of the preceding paragraph, the following crucial property: All the positive occurrences of P's and Q's in it originated from mix nodes in a single branch of the parse tree of ψ.

From now on, to make contact with the terminology of Sect. 5, we shall identify the conjunction θ with the set of its conjuncts, which may thus be called clauses.

Consider a P and a Q such that the mix node with key P is an ancestor of the mix node with key Q in the parse tree of ψ. So P is some P_i in a mix formula $\alpha \wedge \bigwedge_{i=1}^{k}(P_i(\mathbf{x}_i) \vee \delta_i)$, while Q is the key of a mix subformula of δ_i. That mix subformula contains all the occurrences of Q in ψ, because of our no clash assumption. As a result, in the conjunctive normal form θ, every clause with an occurrence of Q also has a positive occurrence of $P(\mathbf{x}_i)$. (We need this information only for the negative occurrences of Q, but it is true for all occurrences of Q.)

Recall that, in this situation, when we formed the strict $\forall^1_1$ form of φ (as the first step toward ψ and θ), the variables $\mathbf{x}_i$ associated to P_i became additional arguments of Q. Because of that, when we now form instances of θ, all clauses containing negative occurrences of any particular instance of Q will also contain a positive occurrence of the corresponding instance of $P_i(\mathbf{x}_i)$. This means, by

Theorem 7, that we can delete positive occurrences of any instance of P from any clause that also contains a positive occurrence of a corresponding instance Q.

These deletions can be uniformly summarized as follows, in terms of θ itself rather than its instances: In any clause containing positive occurrences of P and Q, where Q originated in a descendant of the mix formula of P, one can delete the occurrence of P. But, in every conjunct of θ, the positively occurring P's and Q's originated along a branch, and so one can delete all the positive P's and Q's except the one farthest from the root of the parse tree of ψ.

That leaves at most one positive P in any clause; all instances then have at most one positive literal. That is, we have only Horn clauses in the trimmed conjunctive normal form.

7 Summary

The strict $\forall^1_1$ translation of any ∃FPL sentence has a special syntactic form. Its quantifier-free matrix is the negation of what we called a primary formula in Sect. 4, and the occurrences of variables are constrained by the no clash assumption and the argument joining property. This special form ensures polynomial-time model-checking, because this special syntactic form allows, first, reduction to a secondary formula (still subject to no clashes and argument joining) and, second, trimming to Horn form. Then the original ∃FPL sentence holds in $\mathfrak{A}$ if and only if the set of all $\mathfrak{A}$-instances of this Horn form is not satisfiable.

Acknowledgement. Because of the last-minute discovery of the conjunction problem, this paper was submitted after the official deadline, leaving less than the normal time for refereeing. Nevertheless, the referee provided a very useful report. I thank him or her for the report, in particular for informing me about the existence and relevance of [4].

References

1. Barwise, J.: Admissible Sets and Structures: An Approach to Definability Theory. Perspectives in Mathematical Logic. Springer-Verlag, Berlin (1975)
2. Blass, A.: Existential fixed-point logic, universal quantifiers, and topoi. In: Blass, A., Dershowitz, N., Reisig, W. (eds.) Fields of Logic and Computation. LNCS, vol. 6300, pp. 108–134. Springer, Heidelberg (2010)
3. Blass, A., Gurevich, Y.: Existential fixed-point logic. In: Börger, E. (ed.) Computation Theory and Logic. LNCS, vol. 270, pp. 20–36. Springer, Heidelberg (1987)
4. Grädel, E.: Capturing complexity classes by fragments of second-order logic. Theoret. Computer Sci. **101**, 35–57 (1992)
5. Moschovakis, Y.N.: Elementary Induction on Abstract Structures. Studies in Logic and the Foundations of Mathematics. North-Holland, New York (1974)