Stefan **Bold**, Benedikt **Löwe**,
Thoralf **Räsch**, Johan **van Benthem** (*eds.*)
Foundations of the Formal Sciences V
Infinite Games

The Complexity of Independence-Friendly Fixpoint Logic

Julian Bradfield and Stephan Kreutzer

University of Edinburgh
Laboratory for Foundations of Computer Science
King's Buildings
Mayfield Road
Edinburgh EH9 3JZ, United Kingdom
jcb@inf.ed.ac.uk

Oxford University
Computing Laboratory
Wolfson Building
Parks Road
Oxford OX1 3QD, United Kingdom
kreutzer@comlab.ox.ac.uk

ABSTRACT. We study the complexity of model-checking for the fixpoint extension of Hintikka and Sandu's independence-friendly logic. We show that this logic captures ExpTime; and by embedding PFP, we show that its combined complexity is ExpSpace-hard, and moreover the logic includes second order logic (on finite structures).

1 Introduction

In everyday life we often have to make choices in ignorance of the choices made by others that might have affected our choice. With the popularity of the agent paradigm, there is much theoretical and practical work on logics of knowledge and belief in which such factors can be explicitly expressed in designing multi-agent systems. However, ignorance is not the only reason for making independent choices: in mathematical writing, it is not uncommon to assert the existence of a value for some parameter uniformly in some earlier mentioned parameter.

Hintikka and Sandu [HinSan$_1$96] introduced a logic, called Independence-friendly (IF) logic, in which such independent choices can be formalized by

independent quantification. Some of the ideas go back some decades, for IF logic can also be viewed as an alternative account of branching quantifiers (Henkin quantifiers) in terms of games of imperfect information. Independent quantification is a subtle concept, with many pitfalls for the unwary. It is also quite powerful: it has long been known that it has existential second-order power. In previous work [BraFrö02], the first author and Fröschle applied the idea of independent quantification to modal logics, where it has natural links with the theory of true concurrency; this prompted some consideration of fixpoint versions of IF modal logics, since adding fixpoint operators is the easiest way to get a powerful temporal logic from a simple modal logic. This led the first author to an initial investigation [Bra03] of the fixpoint extension of first-order IF logic, which we call IF-LFP . It turned out that fixpoint IF logic is not trivial to define, and appears to be very expressive, with the interaction between fixpoints and independent quantification giving a dramatic increase in expressive power. In [Bra03], only some fairly simple complexity results were obtained; in this paper, we obtain much stronger results about the model-checking complexity of IF-LFP . For the data complexity, we show that not only is IF-LFP ExpTime-complete, but it captures ExpTime; and for the combined complexity, we obtain an ExpSpace hardness result. This latter result is obtained by an embedding of partial fixpoint logic into IF-LFP, which shows that on finite structures IF-LFP even includes second-order logic, a much stronger result than the first author previously conjectured.

2 IF-FOL and IF-LFP

2.1 Syntax

First of all, we state one important **notational convention**: to minimize the number of parentheses, we take the scope of all quantifiers and fixpoint operators to extend as far to the right as possible.

Now we define the syntax of first-order IF logic. Here we use the version of Hodges [$\text{Hod}_0$97], and we confine the 'independence-friendly' operators to the quantifiers; in the full logic, one can also specify conjunctions and disjunctions that are independent, but these are not necessary for our purposes — their addition changes none of our results.

Definition 1. As for FOL, IF-FOL has proposition (P, Q *etc.*), relation (R, S *etc.*), function (f, g *etc.*) and constant (a, b *etc.*) symbols, with given arities. It also has individual *variables* v, x *etc.* We write $\vec{x}, \vec{v}$ *etc.* for tuples of variables, and similarly for tuples of other objects; we use concatenation of symbols to denote concatenation of tuples with tuples or objects.

For formulae φ and terms t, the (meta-level) notations $\varphi[\vec{x}]$ and $t[\vec{x}]$ mean that the free variables of φ or t are included in the variables $\vec{x}$, without

repetition. The notions of 'term' and 'free variable' are as for FOL.

We assume equality $=$ is in the language, and atomic formulae are defined as usual by applying proposition or relation symbols to individual terms or tuples of terms. The free variables of the formula $R(\vec{t})$ are then those of $\vec{t}$. The compound formulae are given as follows:

- **Conjunction and disjunction.** If $\varphi[\vec{x}]$ and $\psi[\vec{y}]$ are formulae, then $(\varphi \vee \psi)[\vec{z}]$ and $(\varphi \wedge \psi)[\vec{z}]$ are also formulae, where $\vec{z}$ is the union of $\vec{x}$ and $\vec{y}$.
- **Quantifiers.** If $\varphi[\vec{y}, x]$ is a formula, x a variable, and W a finite set of variables, then $(\forall x/W.\, \varphi)[\vec{y}]$ and $(\exists x/W.\, \varphi)[\vec{y}]$ are formulae. If W is empty, we write just $\forall x.\, \varphi$ and $\exists x.\, \varphi$.
- **Game negation.** If $\varphi[\vec{x}]$ is a formula, so is $(\sim\varphi)[\vec{x}]$.
- **Flattening.** If $\varphi[\vec{x}]$ is a formula, so is $(\downarrow\varphi)[\vec{x}]$.
- **(Negation.** $\neg\varphi$ is an abbreviation for $\sim\downarrow\varphi$.**)**

Definition 2. IF-FOL$^+$ is the logic in which $\sim$, $\downarrow$ and $\neg$ are applied only to atomic formulae.

2.2 Traditional semantics

In the independent quantifiers the intention is that W is the set of independent variables, whose values the player is not allowed to know at this choice point: thus the classical Henkin quantifier $\begin{smallmatrix}\forall x & \exists y \\ \forall u & \exists v\end{smallmatrix}$, where x and y are independent of u and v, can be written as $\forall x/\varnothing.\, \exists y/\varnothing.\, \forall u/\{x, y\}.\, \exists v/\{x, y\}$. This notion of independence is the reason for saying that IF logic is natural in mathematical English: statements such as "For every x, and for all $\varepsilon > 0$, there exists δ, depending only on ε ..." can be transparently written as $\forall x, \varepsilon > 0.\, \exists \delta/x.\, \ldots$ in IF logic.

If one then plays the Hintikka evaluation game (otherwise known as the model-checking game) with this additional condition, which can be formalized by requiring strategies to be uniform in the 'unknown' variables, one gets a game semantics of imperfect information, and defines a formula to be true if and only if Eloise has a winning strategy.

These games are not determined, so it is *not* the case that Abelard has a winning strategy iff the formula is not true. For example, $\begin{smallmatrix}\forall x \\ \exists y\end{smallmatrix}.x = y$ (or $\forall x.\, \exists y/\{x\}.\, x = y$) is untrue in any structure with more than one element, but Abelard has no winning strategy.

An alternative interpretation of the logic, dating from the early work on branching quantifiers, and one that is easier to handle mathematically in straightforward cases, is via Skolem functions with limited arguments. In

FOL, the first order sentence $\forall x.\, \exists y.\, x = y$ is converted via Skolemization to the existential second-order sentence $\exists f : \mathbb{N} \to \mathbb{N}.\, \forall x.\, x = f(x)$. In this procedure, the Skolem function always takes as arguments all the universal variables currently in scope. By allowing Skolem functions to take only some of the arguments, we get a similar translation of IF-FOL: for example, $\forall x.\, \exists y/\{x\}.\, x = y$ becomes $\exists f : 1 \to \mathbb{N}.\, \forall x.\, x = f()$. It can be shown that these two semantics are equivalent, in that an IF-FOL sentence is true in the game semantics if and only if its Skolemization is true.

It is also well known that IF-FOL$^+$ is equivalent to existential second-order logic (in the cases where this matters, 'second-order' here means function quantification rather than set quantification). This is because the Skolemization process can be inverted: given an ESO sentence, it can be turned into an IF-FOL sentence (or equivalently, a sentence with Henkin quantifiers). We shall make use of this procedure later. Details can be found in [Wal70, End70], but here let us illustrate it by a standard example that demonstrates the power of IF logic. Consider the sentence 'there is an injective endofunction that is not surjective'. This is true only in infinite domains, and therefore not first-order expressible. It can be expressed directly in ESO as

$$\exists f.\, (\forall x_1, x_2.\, f(x_1) = f(x_2) \Rightarrow x_1 = x_2) \wedge (\exists c.\, \forall x.\, f(x) \neq c)$$

which for the sake of reducing complexity below we will simplify to

$$\exists f.\, \exists c.\, \forall x_1, x_2.\, (f(x_1) = f(x_2) \Rightarrow x_1 = x_2) \wedge f(x_1) \neq c.$$

The basic manoeuvre for talking about functions in IF-FOL is to replace $\exists f.\, \forall x$ by $\forall x.\, \exists y$, so that y plays the rôle of $f(x)$. In FOL, this works only if there is just one application of f; but in IF-FOL, we can do it for two (or more) applications of f: we write $\forall x_1.\, \exists y_1$, and then we write an independent $\forall x_2/\{x_1, y_1\}.\, \exists y_2/\{x_1, y_1\}$. Now in order to make sure that these two (x_i, y_i) pairings represent the same f, the body of the translated formula is given a clause $(x_1 = x_2) \Rightarrow (y_1 = y_2)$. Applying this procedure to the ESO sentence above and optimizing a bit, we get

$$\forall x_1, x_2.\, \exists y_1/x_2.\, \exists y_2/x_1.\, \exists c/\{x_1, x_2\}.\, (y_1 = y_2 \Leftrightarrow x_1 = x_2) \wedge y_1 \neq c.$$

The 'game negation' $\sim$ corresponds to swapping the roles of the two players in the game. In ordinary logic, with perfect information, this corresponds exactly to classical negation; but in IF-FOL it does not. The issue of negation is still controversial, particularly for open formulae; but one approach, taken by [Hod$_0$97], is to define the 'flattening' operator $\downarrow$, which smashes its argument down to a classical formula by removing all the uniformity constraints imposed by imperfect information. Then classical negation is defined to be game negation applied to a flattened formula.

2.3 Trump semantics

The game semantics is how Hintikka and Sandu originally interpreted IF logic. Later on, the trump semantics of Hodges [Hod$_0$97], with variants by others, gave a Tarski-style semantics, equivalent to the original. This semantics is as follows:

Definition 3. Let a structure A be given, with constants, propositions and relations interpreted in the usual way. A *deal* $\vec{a}$ for $\varphi[\vec{x}]$ or $\vec{t}[\vec{x}]$ is an assignment of an element of A to each variable in $\vec{x}$. Given a deal $\vec{a}$ for a tuple of terms $\vec{t}[\vec{x}]$, let $\vec{t}(\vec{a})$ denote the tuple of elements obtained by evaluating the terms under the deal $\vec{a}$.

If $\varphi[\vec{x}]$ is a formula and W is a subset of the variables in $\vec{x}$, two deals $\vec{a}$ and $\vec{b}$ for φ are $\simeq_W$*-equivalent* ($\vec{a} \simeq_W \vec{b}$) if and only if they agree on the variables not in W. A $\simeq_W$*-set* is a non-empty set of pairwise $\simeq_W$-equivalent deals.

The denotation $[\![\varphi]\!]$ of a formula is a pair (T, C) where T is the set of *trumps*, and C is the set of *cotrumps*. If φ has n free variables, then $T, C \subseteq \wp(\wp(A^n))$ – that is, a (co)trump is a set of deals.

- If $(R(\vec{t}))[\vec{x}]$ is atomic, then a non-empty set D of deals is a trump if and only if $\vec{t}(\vec{a}) \in R$ for every $\vec{a} \in D$; D is a cotrump if and only if it is non-empty and $\vec{t}(\vec{a}) \notin R$ for every $\vec{a} \in D$.
- D is a trump for $(\varphi \wedge \psi)[\vec{x}]$ if and only if D is a trump for $\varphi[\vec{x}]$ and D is a trump for $\psi[\vec{x}]$; D is a cotrump if and only if there are cotrumps E, F for φ, ψ such that every deal in D is an element of either E or F.
- D is a trump for $(\varphi \vee \psi)[\vec{x}]$ if and only if it is non-empty and there are trumps E of φ and F of ψ such that every deal in D belongs either to E or F; D is a cotrump if and only if it is a cotrump for both φ and ψ.
- D is a trump for $(\forall y/W.\, \psi)[\vec{x}]$ if and only if the set $\{\, \vec{a}b \mid \vec{a} \in D, b \in A \,\}$ is a trump for $\psi[\vec{x}, y]$. D is a cotrump if and only if it is non-empty and there is a cotrump E for $\psi[\vec{x}, y]$ such that for every $\simeq_W$-set $F \subseteq D$ there is a b such that $\{\, \vec{a}b \mid \vec{a} \in F \,\} \subseteq E$.
- D is a trump for $(\exists y/W.\, \psi)[\vec{x}]$ if and only if there is a trump E for $\psi[\vec{x}, y]$ such that for every $\simeq_W$-set $F \subseteq D$ there is a b such that $\{\, \vec{a}b \mid \vec{a} \in F \,\} \subseteq E$; D is a cotrump if and only if the set $\{\, \vec{a}b \mid \vec{a} \in D, b \in A \,\}$ is a cotrump for $\psi[\vec{x}, y]$.
- D is a trump for $\sim\varphi$ if and only if D is a cotrump for φ; D is a cotrump for $\sim\varphi$ if and only if it is a trump for φ.

- D is a trump (cotrump) for $\downarrow\varphi$ if and only if D is a non-empty set of members (non-members) of trumps of φ.

A trump for φ is essentially a set of winning positions for the model-checking game for φ, for a given *uniform* strategy, that is, a strategy where choices are uniform in the 'hidden' variables. The most intricate part of the above definition is the clause for $\exists y/W.\,\psi$: it says that a trump for $\exists y/W.\,\psi$ is got by adding a witness for y, uniform in the W-variables, to trumps for ψ.

In the absence of flattening, a cotrump is simply a trump for the game negation of a formula, in other words a set of winning positions for Abelard in the model-checking game. If one ignores flattening, it is not necessary to maintain cotrumps in the semantics, and so we shall often elide them.

It is easy to see that any subset of a trump is a trump. In the case of an ordinary first-order $\varphi(\vec{x})$, the set of trumps of φ is just the power set of the set of tuples satisfying φ. To see how a more complex set of trumps emerges, consider the following formula, which has x free: $\exists y/\{x\}.\,x = y$. Any singleton set of deals is a trump, but no other set of deals is a trump. Thus we obtain that $\forall x.\,\exists y/\{x\}.\,x = y$ has no trumps (unless the domain has only one element).

The strangeness of the trump definitions is partly to do with some more subtle features of IF logics, that we do not here have space to discuss, but which are considered in detail in Ahti-Veikko Pietarinen's thesis [Pie01]. However, to take one good example, raised by a referee, consider $\varphi = \exists x.\,\exists y/\{x\}.\,x = y$. What are its trumps? As above, the trumps of $\exists y/\{x\}.\,x = y$ are singleton sets of deals. The only potential trump for φ is the set containing the empty deal $D = \{\langle\rangle\}$. Applying the definition, D is a trump for φ if and only if there is a singleton deal set $\{a\}$ for x such that there is a b such that $\{b\} \subseteq \{a\}$. The right hand side is true – take $b = a$ – so D is a trump. How come, if there is more than one element in A? Surely we must choose y independently of x, and therefore φ can't be true? Not so: because the choices are both made by the same player (Eloise), she can, as it were, make a uniform choice of y that, by 'good luck' agrees with her previous choice of x. Since she is not in the business of making herself lose, she will always do so. In game-theoretic terms, this is the difference between requiring a strategy to make uniform moves, and requiring a player to choose a strategy uniformly. In fact Hintikka and Sandu tried to hide this issue by only allowing the syntax to express quantifications independent in the other player's variables, which is in practice all one wishes to use in any case; but they also incorporated it by asserting that a player's choices are always independent of their own earlier choices, which is intuitively bizarre.

Hodges removed the syntactic restriction to make his semantics cleaner, exposing the issue more obviously.

A sentence is said to be true if $\{\langle\rangle\} \in T$ (the empty deal is a trump set), and false if $\{\langle\rangle\} \in C$; this corresponds to Eloise or Abelard having a uniform winning strategy. Otherwise, it is undetermined. Note that 'false' is reserved for a strong sense of falsehood – undetermined sentences are also not true, and in the simple cases where negation and flattening are not employed, an undetermined sentence is as good as false. Note also that the game negation $\sim$ provides the usual de Morgan dualities, but that it cannot be pushed through flattening.

2.4 IF-LFP

We now describe the addition of fixpoint operators to IF-FOL. This is slightly intricate, although the normal intuitions for understanding fixpoint logics still apply.

Definition 4. IF-LFP extends the syntax of IF-FOL as follows:

- There is a set $\mathrm{Var} = \{X, Y, \ldots\}$ of fixpoint variables. Each variable X has an arity $\mathrm{ar}(X)$.
- If X is a fixpoint variable, and $\vec{t}$ an $\mathrm{ar}(X)$-vector of terms then $X(\vec{t})$ is a formula.
- Let φ be a formula with free fixpoint variable X. φ has free individual variables $\vec{x} = \langle x_1, \ldots, x_{\mathrm{ar}(X)}\rangle$ for the elements of X, together with other free individual variables $\vec{z}$; let $\mathrm{fv}_\varphi(X)$ be the length of $\vec{z}$. Now if $\vec{t}$ is a sequence of $\mathrm{ar}(X)$ terms with free variables $\vec{y}$, then $(\mu X(\vec{x}).\varphi)(\vec{t})[\vec{z}, \vec{y}]$ is a formula; **provided that** φ is IF-FOL$^+$. In this context, we write just $\mathrm{fv}(X)$ for $\mathrm{fv}_\varphi(X)$.
- Similarly for $\nu X(\vec{x}).\varphi$.

To give the semantics of IF-LFP, we first define valuations for free fixpoint variables, in the context of some IF-LFP formula.

Definition 5. A fixpoint valuation $\mathcal{V}$ maps each fixpoint variable X to a pair

$$(\mathcal{V}_T(X), \mathcal{V}_C(X)) \in (\wp(\wp(A^{\mathrm{ar}(X)+\mathrm{fv}(X)})))^2.$$

Let D be a non-empty set of deals for $X(\vec{t})[\vec{x}, \vec{z}, \vec{y}]$, where $\vec{y}$ are the free variables of $\vec{t}$ not already among $\vec{x}, \vec{z}$. A deal $d = \vec{a}\,\vec{c}\,\vec{b} \in D$, where $\vec{a}, \vec{c}, \vec{b}$ are the deals for $\vec{x}, \vec{z}, \vec{y}$ respectively, determines a deal $d' = \vec{t}(d)\vec{c}$ for $X[\vec{x}, \vec{z}]$. Let $D' = \{\, d' \mid d \in D \,\}$. The set D is a trump for $X(\vec{t})$ if and only if $D' \in \mathcal{V}_T(X)$; it is a cotrump if and only if $D' \in \mathcal{V}_C X$.

The intuition here is that a fixpoint variable needs to carry the trumps and cotrumps both for the elements of the fixpoint and for any free variables, as we shall see below. Then we define a suitable complete partial order on the range of valuations, which will also be the range of denotations for formulae; it is simply the inclusion order on trump sets and the reverse inclusion order on cotrump sets:

Definition 6. If (T_1, C_1) and (T_2, C_2) are elements of $(\wp(\wp(A^n)))^2$, define $(T_1, C_1) \preceq (T_2, C_2)$ if and only if $T_1 \subseteq T_2$ and $C_1 \supseteq C_2$.

This order gives the standard basic lemma for fixpoint logics:

Lemma 7. If $\varphi(X)[\vec{x}, \vec{z}]$ is an IF-FOL$^+$ formula and $\mathcal{V}$ is a fixpoint valuation, the map on $(\wp(\wp(A^{\mathrm{ar}(X)+\mathrm{fv}(X)}))^2$ given by

$$(T, C) \mapsto [\![\varphi]\!]_{\mathcal{V}[X:=(T,C)]}$$

is monotone with respect to $\preceq$; hence it has least and greatest fixpoints, constructible by iteration from the bottom and top elements of the set of denotations.

Thus we have the familiar definition of the μ operator:

Definition 8. $[\![\mu X(x).\varphi(X)[\vec{x}, \vec{z}]]\!]$ is the least fixpoint of the map on $(\wp(\wp(A^{\mathrm{ar}(X)+\mathrm{fv}(X)}))^2$ given by

$$(T, C) \mapsto [\![\varphi]\!]_{\mathcal{V}[X:=(T,C)]};$$

and $[\![\nu X(x).\varphi(x)[\vec{x}, \vec{z}]]\!]$ is the greatest fixpoint. $\mu^{\zeta}_{X,x}\varphi$ means the ζth approximant of $\mu X(x).\varphi$, defined recursively by $\mu^{\zeta}_{X,x}\varphi = \varphi(\bigcup_{\xi<\zeta} \mu^{\xi}_{X,x}\varphi)$.

A distinctive feature of the definition, compared to the normal LFP definition, is the way that free variables are explicitly mentioned. Normally, one can fix values for the free variables, and then compute the fixpoint, but because of independent quantification this is not possible in the IF setting. For example, consider the formula fragment

$$\forall z. \ldots \mu X(x). \ldots \vee \exists y/\{z\}. X(y)$$

The independent choice of y means that the trumps for the fixpoint depend on the possible deals for z, not just a single deal.

2.5 Examples of IF-LFP

In order to give some human-readable examples of IF-LFP, we here reproduce a section from [Bra03]. For convenience, we introduce the abbreviation $\varphi \Rightarrow \psi$ for $\psi \vee {\sim}\varphi$ provided that φ is atomic.

Let $G = (V, E)$ be a directed graph. The usual LFP formula $R(y, z) := (\mu(X, x).z = x \vee \exists w.\, E(x, w) \wedge X(w))(y)$ asserts that the vertex z is reachable from y. Hence the formula $\forall y.\, \forall z.\, R(y, z)$ asserts that G is strongly connected. Now consider the IF-LFP formula

$$\forall y.\, \forall z.\, (\mu(X, x).z = x \vee \exists w/\{y, z\}.\, E(x, w) \wedge X(w))(y).$$

At first sight, one might think this asserts not only that every z is reachable from every y, but that the path taken is independent of the choice of y and z. This is true exactly if G has a directed Hamiltonian cycle, a much harder property than being strongly connected.

Of course, the formula does not mean this, because the variable w is fresh each time the fixpoint is unfolded. In the trump semantics, the denotation of the fixpoint will include all the possible choice functions at each step, and hence all possible combinations of choice functions. Thus the formula reduces to strong connectivity.

It may be useful to look at the approximants of this formula in a little more detail, to get some intuitions about the trump semantics. Considering just

$$H := (\mu(X, x).z = x \vee \exists w/\{y, z\}.\, E(x, w) \wedge X(w))[x, y, z],$$

we see that in computing each approximant, the calculation of $[\![\exists w/\{y, z\}.\, \ldots]\!]$ involves generating a trump for every possible value of a choice function $f\colon x \mapsto w$. This is a feature of the original trump semantics, and can be understood by viewing it as a second-order semantics: just as the compositional Tarskian semantics of $\exists x.\, \varphi(x)$ involves computing all the witnesses for $\varphi(x)$, so computing the trumps of $\exists x/\{y\}.\, \varphi$ involves computing all the Skolem functions; and unlike the first-order case, it is necessary to work with functions (as IF can express existential second-order logic). Consequently, the nth approximant includes all states such that $x \to f_1(x) \to f_2 f_1(x) \to \ldots \to f_n \ldots f_1(x) = z$ for any sequence of successor-choosing functions f_i. Thus we see that the cumulative effect is the same as for a normal $\exists w$, and the independent choice has indeed not bought us anything.

It is, however, possible to produce a slightly more involved formula expressing the Hamiltonian cycle property in this inductively defined way, by using the standard trick for expressing functions in Henkin quantifier logics. We replace the formula H by

$$\begin{aligned}&\forall s.\, \exists t/\{y, z\}.\, E(s, t) \wedge (\mu X(x).x = z \vee \\ &\forall u.\, \exists v/\{x, y, z, s, t\}.\, (s = u \Rightarrow t = v) \wedge (x = u \Rightarrow X(v)))(y).\end{aligned}$$

This works because the actual function f selecting a successor for every node is made outside the fixpoint by $\forall s.\, \exists t/\{y, z\}.\, E(s, t) \wedge \ldots$; then inside

the fixpoint, a new choice function g is made so that $X(g(x))$, and g is constrained to be the same as f by the clause $(s = u \Rightarrow t = v)$. (The reader who is not familiar with the IF/Henkin to existential second-order translation might wish to ponder why $\forall s.\, \exists t/\{y, z\}.\, E(s,t) \wedge \mu(X,x).x = z \vee (x = s \Rightarrow X(t))$ does not work.)

2.6 The issue of negation

As we remarked above, negation is a somewhat problematic concept in IF logics. In the game-theoretic presentation, IF sentences may be true, false, or undetermined. When we say that an IF logic is equivalent to a classical logic, such as IF-FOL $= \Sigma^1_1$, we mean that the IF formula is true if and only if its classical equivalent is true. Hence if an IF sentence φ is undetermined, its translation $\hat{\varphi}$ is false; and so $\sim\varphi$ is certainly not equivalent to $\neg\hat{\varphi}$. Hodges' introduction of the $\downarrow$ operator provides a way to get classical negation into IF sentences: if φ is an IF sentence that is undetermined, it has neither trumps nor cotrumps; hence $\downarrow\varphi$ has the empty deal as a co-trump, since $\langle\rangle$ is a non-member of the co-trumps of φ; and so $\sim\downarrow\varphi$ is true. The intuitive interpretation of $\downarrow$ on formulae with free variables is unclear, and its combination with game negation more so. Indeed, there is not a simple game account of the flattening operator.

The arguments we apply in the following results rely largely on the ability to simulate operations on functions by operations on trumps. It is unclear to us how to combine boolean negation on the functional side with game/flattening negation on the trump side. The model-checking upper bound holds also for the full IF logic with negation and flattening; and the complexity lower bounds hold *a fortiori* for the full logic. However, our lower bounds, obtaining by translating from classical logic to IF logic, use various techniques to avoid having to translate classical negation to IF negation, and it would be useful to know whether additional power is obtained from IF negation.

3 Second-order inductions and independence-friendly logics

It has been known from the early studies of Henkin quantifiers [Wal70, End70] that existential second-order sentences can be transformed into sentences with the Henkin quantifier, and thus into IF-FOL. A technique frequently used in our results is the translation of existential second order inductions into IF-LFP. For this we show that the translation of existential second-order logic into independence-friendly logic can be extended to a translation of positive existential second-order inductions into independence-friendly fixpoint logic. Throughout this paper we only consider finite struc-

tures. Therefore we only give the translation for finite structures here. We first give a formal definition of positive Σ^1_1-inductive formulae.

Definition 9. An (n,k)-ary third-order variable $\mathcal{R}$ is a variable interpreted by a set whose members are n-tuples of k-ary functions. Let, for some $k, n < \omega$, $\mathcal{R}$ be a (n,k)-ary third-order variable. A formula $\varphi(\mathcal{R}, f_1, \ldots, f_n)$ is Σ^1_1-*inductive* if it is built up by the usual formula building rules for Σ^1_1 augmented by a rule that allows the use of atoms $\mathcal{R}f_1 \ldots f_n$, where the f_i are k-ary function symbols, provided that the variable $\mathcal{R}$ is only used positively in φ.

Σ^1_1-inductive formulae φ can be used to define least fixpoint inductions in the same way as first-order formulae with a free relation variable in which they are positive are used to define fixpoint inductions. So we can define the stages $\mathcal{R}^\alpha$, $\alpha < \omega$, of the fixpoint induction in φ which ultimately lead to the least fixpoint of the operator defined by the formula φ. We call a relation that is obtained as the least fixpoint of a Σ^1_1-inductive formula Σ^1_1-*inductive*. Note, that the Σ^1_1-inductive relations are third-order objects, *i.e.*, sets of functions.

We show next that any Σ^1_1-inductive third-order relation $\mathcal{R}$ can be defined by an IF-LFP-formula in the sense that there is a formula $\varphi(R, \vec{x}, y)$, positive in the second-order variable R, such that the maximal trumps in the least fixpoint of the operator defined by φ are precisely the graphs of the functions in $\mathcal{R}$. For the sake of simplicity, we only consider the case of $(1,k)$-ary inductions, *i.e.*, where the fixpoint is a set of functions.

An important concept used in the following proofs is the notion of *functional trumps*; and a technically useful concept is that of *maximal trumps*.

Definition 10. Let $\varphi(\vec{x}, y)$ be a formula. A trump T for φ is *functional in* $\vec{x}$ *and* y, if for all pairs $(\vec{a}, b), (\vec{a}', b')$ of deals in T we have $b = b'$ whenever $\vec{a} = \vec{a}'$. T is *maximal* if there is no $T' \supsetneq T$ that is a trump for φ.

Note that because any subset of a trump is a trump, the trumps of a formula are determined by its maximal trumps. Of course, any subset of a functional trump is functional.

Notation. In the following proofs we will frequently use a construction like

$$\forall \vec{x} / \{\vec{x}_1, y_1, \ldots, \vec{x}_n, y_n\} \exists y / \{\vec{x}_1, y_1, \ldots, \vec{x}_n, y_n\} \big((\vec{x} = \vec{x}_1 \to y = y_1) \wedge \varphi \big)$$

for some formula φ. We will abbreviate this by

$$\forall \vec{x} \exists y \, \mathrm{clone}(\vec{x}_1, y_1; \vec{x}_2, y_2 \ldots, \vec{x}_n, y_n) \varphi.$$

and we will usually omit the list $(\vec{x}_2, y_2 \ldots, \vec{x}_n, y_n)$ of other variables which appear in the independence sets of the quantifiers, assuming that all other

variables than the clones and originals are in that list. Essentially, this formula says that the Skolem functions f_y and f_{y_1} chosen for y and y_1, respectively, are the same. The next lemma makes this precise and establishes some useful properties of the clone construction.

Lemma 11. Let $\mathbf{A}$ be a structure and let $\vec{x}$ be a k-tuple of variables.

(i) Let ψ be a formula defined as $\psi(\vec{x}, y) := \forall\vec{x}'\exists y'\mathrm{clone}(\vec{x}, y)\psi'$. Then the trumps for ψ are precisely the sets of deals $(\ldots, \vec{a}, f(\vec{a}))$ for $\vec{a} \in A^{|\vec{x}|}$ functional in $\vec{x}$ and y with some Skolem function f, such that the deals $(\ldots, \vec{a}, f(\vec{a}), \vec{a}', f(\vec{a}'))$ for $\vec{a} \in A^{|\vec{x}|}$ form a trump for ψ'. In particular, if ψ' is the formula **true**, then the trumps of ψ are exactly the sets of deals functional in $\vec{x}$ and y.

(ii) Let $\varphi(\vec{x}', y')$ be a formula with only functional trumps and let ψ be defined as $\psi(\vec{x}, y) := \forall\vec{x}'\exists y'\mathrm{clone}(\vec{x}, y)\ \varphi$. Then the trumps for ψ and the trumps for φ are the same, in the sense that for every trump $T' \subseteq A^{k+1}$ of φ there is a trump $T \subseteq A^{k+1}$ of ψ such that an assignment of elements $\vec{a}$ to the variables $\vec{x}'$ and b to y' is a deal in T' if, and only if, the corresponding assignment of $\vec{a}$ to $\vec{x}$ and b to y is a deal in T and, conversely, for every trump T of ψ there is a corresponding trump T' for φ.

Proof. We first prove Part (i) of the lemma. Following our notation, the formula ψ is an abbreviation for

$$\forall\vec{x}'/\{\vec{x}, y\}\exists y'/\{\vec{x}, y\}(\vec{x} = \vec{x}' \to y = y') \wedge \psi'.$$

Towards a contradiction, suppose there was a non-functional trump T for ψ, *i.e.*, T contains deals $(\vec{a}, b)$ and $(\vec{a}, b')$ for some $\vec{a}$ and $b \neq b'$. By the semantics of universal quantifiers, this implies that there must be a trump for $\exists y_1/\{\vec{x}, y\}(\vec{x} = \vec{x}' \to y = y') \wedge \psi'$ containing $(\vec{a}, b, \vec{a})$ and $(\vec{a}, b', \vec{a})$. But then, the set $\{(\vec{a}, b, \vec{a}), (\vec{a}, b', \vec{a})\}$ is a $\{\vec{x}, y\}$-set (recall Definition 3). Hence there must be a trump T' for $(\vec{x} = \vec{x}' \to y = y') \wedge \psi'$ and an element c so that T' contains the deals $(\vec{a}, b, \vec{a}, c)$ and $(\vec{a}, b', \vec{a}, c)$. But this is impossible as not both $b = c$ and $b' = c$ can be true but obviously every deal $(\vec{d}, e, \vec{d}', e')$ in a trump for $(\vec{x} = \vec{x}' \to y = y')$ satisfies the condition that if $\vec{d} = \vec{d}'$ then also $e = e'$. Finally, if T is a functional trump, then the corresponding T' must be a trump for ψ', and so the deals $(\vec{a}, b, \vec{a}, b)$ must be a trump for ψ'. Part (ii) of the lemma follows analogously. q.e.d.

The next lemma shows that every formula in Σ^1_1 is equivalent to a formula in IF-LFP. The proof of the lemma follows easily from the work on Henkin-

quantifiers. However, as we need the lemma also for formulae with free function variables we give an explicit translation of Σ^1_1-formulae into IF-FOL.

Lemma 12. Let $\varphi(f_1, \ldots, f_n)$ be a Σ^1_1-formula with free function variables $f_1, \ldots, f_k$. Then there is a formula $\hat{\varphi}(\vec{x}_{f_1}, y_{f_1}, \ldots, \vec{x}_{f_k}, y_{f_k}) \in$ IF-FOL such that for every structure $\mathbf{A}$ a set T is a maximal trump for $\hat{\varphi}$ if, and only if, there are functions $F_1, \ldots, F_k$ such that $\mathbf{A} \models \varphi(F_1, \ldots, F_k)$ and

$$T = \{(\vec{a}_1, b_1, \ldots, \vec{a}_k, b_k) : F_i(\vec{a}_i) = b_i \text{ for all } 1 \leq i \leq k\}.$$

Proof. We first present the standard translation of Σ^1_1 into independence-friendly first-order logic (in a less efficient but more transparent form than normal). Let ψ be a first-order formula containing a free function variable g. For the variable g, introduce variables $\vec{x}_0, y_0$, and for each of the $i = 1, \ldots, n$ applications $g(\vec{t}_i)$ of g introduce variables $\vec{x}_i, y_i$. Then the Σ^1_1 sentence $\exists g.\, \psi$ is translated to $\hat{\psi}_0 = \forall \vec{x}_0.\, \exists y_0.\, \hat{\psi}_1$, where

$$\hat{\psi}_i = \forall \vec{x}_i.\, \exists y_i.\, \mathrm{clone}(\vec{x}_{i-1}, y_{i-1}) \hat{\psi}_{i+1}$$

for $i = 1, \ldots, n$, and $\hat{\psi}_{n+1}$ is ψ with each g-containing atom $Q(g(\vec{t}_i))$ replaced by $\vec{x}_i = \vec{t}_i \Rightarrow Q(y_i)$.

We can stop short of the final $\exists f$, and translate ψ itself to $\hat{\psi}_1$, giving the lemma for one function symbol. By Lemma 11(i), the trumps for $\hat{\psi}_n$ are the sets of deals functional in $\vec{x}_{n-1}$ and y_{n-1} such that when extended by a Skolem function they are trumps for $\hat{\psi}_{n+1}$. Now $\hat{\psi}_{n+1}$ is classical, so its trumps are just the sets of deals satisfying it classically.

Now repeated application of Lemma 11 to $\hat{\psi}_{n-1}, \ldots, \hat{\psi}_1$ that the trumps for $\hat{\psi}_1$ are functional in $\vec{x}_0$ and y_0, with a Skolem function g such that $\hat{\psi}_{n+1}$ is true when every y_i is replaced by $g(\vec{x}_i)$, as required.

It remains to deal with multiple function variables, and to allow some of them to be explicitly closed by existential quantification. To manage multiple function symbols, it is easiest to process them in parallel: if we are dealing with g and h, then extend the above translation to work simultaneously with $\vec{x}, y$ for g and $\vec{u}, v$ for h, in the obvious way: put both sets of function-constraining conjuncts in, and both sets of $\forall\exists$ quantifiers. The two sets of variables should made independent of each other. (This would look much more obvious in Henkin quantifier notation.) If it is desired to quantify some of the function variables by $\exists$, then simply apply the final stage of the translation to those variables. q.e.d.

We are now ready to prove the main theorem of this section.

Theorem 13. Let $\mathcal{R}$ be a $(1,k)$-ary third-order variable and let $\varphi(\mathcal{R},f)$ be a Σ^1_1-inductive formula where f is a k-ary function symbol. Then there is a formula $\varphi^*(R,\vec{x},y) \in$ IF-LFP, where R is a $k+1$-ary second-order variable that only occurs positively in φ and $\vec{x}$ is a k-tuple of variables, such that the least fixpoint R^∞ of φ satisfies the following properties.

1. Every trump T in R^∞ is functional.
2. Every maximal trump encodes the graph of a function in $\mathcal{R}^\infty$ and, conversely,
3. for every function $f \in \mathcal{R}^\infty$ there is a trump T in R^∞ encoding the graph of f.

Proof. Let $\varphi(\mathcal{R},f)$ be as in the statement of the theorem. Without loss of generality we assume that φ has the form

$$\varphi(\mathcal{R},f_0) := \varphi_0(f_0) \vee \exists f_1 \ldots \exists f_n \big((\bigwedge_{i=1}^{n} \mathcal{R} f_i) \wedge \varphi_1 \big)$$

so that $\mathcal{R}$ does not occur in φ_0 or φ_1. (See [EbbFlu99] for a proof of this normal form for existential first-order inductions. The proof for this case is analogous.) The formula φ is translated into a formula $\hat{\varphi}(R,\vec{x},y) \in$ IF-LFP defined as follows:

$$\hat{\varphi}(R,\vec{x},y) \quad := \quad \forall \vec{x}_1 . \exists y_1 . \mathrm{clone}(\vec{x},y) \big(\psi_0(\vec{x},y) \ \vee \ \psi_1(R,\vec{x}_1,y_1) \big)$$

where

$$\psi_0(\vec{x},y) \quad := \quad \forall \vec{x}_{f_0} \exists y_{f_0} \, \mathrm{clone}(\vec{x},y) \ \hat{\varphi}_0(\vec{x}_{f_0}, y_{f_0})$$

and

$$\psi_1(R,\vec{x}_1,y_1) \quad := \quad \forall \vec{x}_{f_0} \exists y_{f_0} \, \mathrm{clone}(\vec{x}_1,y_1) \ \psi_1'(\vec{x}_{f_0}, y_{f_0})$$

and

$$\begin{aligned} \psi_1'(R,\vec{x}_{f_0},y_{f_0}) \quad := \quad & \forall \vec{x}_{f_1} \exists y_{f_1} \ldots \forall \vec{x}_{f_n} \exists y_{f_n} \\ & \textstyle\bigwedge_{i=1}^{n} (\forall \vec{x}' \exists y' \mathrm{clone}(\vec{x}_{f_i}, y_{f_i}) R \vec{x}' y') \ \wedge \\ & \hat{\varphi}_1(\vec{x}_{f_0}, y_{f_0}, \vec{x}_{f_1}, y_{f_1}, \ldots, \vec{x}_{f_n}, y_{f_n}). \end{aligned}$$

We claim that the formula $\hat{\varphi}$ satisfies the properties stated in the theorem. Let $\mathbf{A}$ be a structure with universe A. By Lemma 11(i), the trumps T for are functional in $\vec{x}$ and y, with Skolem function g such that g satisfies ψ_0 or ψ_1.

We show by ordinal induction that every maximal trump in R^α is the graph of a function in $\mathcal{R}^\alpha$ and, conversely, the graph of every function in $\mathcal{R}^\alpha$

is a trump in R^α. For limit ordinals (including 0) this follows immediately from the induction hypothesis.

Now let $\alpha = 1$. We show first that every maximal trump in R^1 is the graph of a function in $\mathcal{R}^1$. By definition, $R^0 = \varnothing$ and therefore the subformula ψ_1 can not be satisfied by any trump. Hence, the only trumps in R^1 are the functional trumps for ψ_0. By Lemma 12, a set $T \subseteq A^{k+1}$ is a trump for $\hat{\varphi}_0$ if, and only if, there is a function $f : A^k \to A$ such that $(\mathbf{A}, f) \models \varphi_0$ and for all deals $(\vec{a}, b) \in T$ we have $f(\vec{a}) = b$. Hence, the maximal trumps for $\hat{\varphi}_0$ are precisely the graphs of functions satisfying φ_0. Thus, by Part (ii) of Lemma 11, we get that the maximal trumps for $\forall\vec{x}_1\exists y_1 \mathrm{clone}(\vec{x}, y)\ \hat{\varphi}_0$ are the graphs of functions satisfying φ_0.

Conversely, let f be a function satisfying φ_0. By Lemma 12 the trump $T := \{(\vec{a}, b) : f(\vec{a}) = b\}$ is a trump for $\hat{\varphi}_0$ and hence for $\forall\vec{x}_1\exists y_1 \mathrm{clone}(\vec{x}, y)\ \hat{\varphi}_0$.

Now let $\alpha > 1$ be a successor ordinal, *i.e.*, $\alpha = \beta{+}1$ for some $\beta > 0$. Again we first show that every maximal trump in R^α is the graph of a function $f \in \mathcal{R}^\alpha$. This is clear for all trumps of ψ_0 as they are already contained in R^1. Now consider the formula ψ_1. For simplicity we assume without loss of generality that all tuples $\vec{x}_{f_0}, \ldots, \vec{x}_{f_n}$ are of arity k. By Lemma 12, the maximal trumps for $\hat{\varphi}_1$ are the sets $T \subseteq A^{(n+1)(k+1)}$ such that there are functions $f_0, \ldots, f_n$ satisfying φ_1 and for all tuples $(\vec{a}_0, b_0, \ldots, \vec{a}_n, b_n)$ we have $(\vec{a}_0, b_0, \ldots, \vec{a}_n, b_n) \in T$ if, and only if, $f_i(\vec{a}_i) = b_i$ for all i. Further, applying the induction hypothesis and Lemma 11, we get that the maximal trumps for $\bigwedge_{i=1}^n(\forall\vec{x}'\exists y' \mathrm{clone}(\vec{x}_{f_i}, y_{f_i})\ R\vec{x}'y')$ are sets T of deals such that there are functional trumps $T_1, \ldots, T_n \in R^\beta$ which, by induction hypothesis, are the graphs of functions $f_1, \ldots, f_n \in \mathcal{R}^\beta$, and for all tuples $(\vec{a}_1, b_1, \ldots, \vec{a}_n, b_n)$ we have $(\vec{a}_1, b_1, \ldots, \vec{a}_n, b_n) \in T$ if, and only if, $f_i(\vec{a}_i) = b_i$ for all i. Thus, every maximal trump of $\bigwedge_{i=1}^n(\forall\vec{x}'\exists y' \mathrm{clone}(\vec{x}_{f_i}, y_{f_i})\ R\vec{x}'y') \wedge \hat{\varphi}_1$ is functional in $\vec{x}_{f_0}$ and y_{f_0} and this function f satisfies φ_1 for some interpretation of the variables $f_1, \ldots, f_n$ by functions from $\mathcal{R}^\beta$. Therefore f is contained in $\mathcal{R}^\alpha$. Thus, by Lemma 11, every maximal trump for ψ_1 encodes the graph of such a function $f \in \mathcal{R}^\alpha$.

The converse is again easily seen. For every function $f \in \mathcal{R}^\alpha \backslash \mathcal{R}^\beta$ choose functions $f_1, \ldots, f_n \in R^\beta$ so that $\mathbf{A} \models \varphi_1(f_0, \ldots, f_n)$. By induction hypothesis, the graphs of these functions are trumps in R^β. Hence, we can choose these trumps in the conjunction $\bigwedge_{i=1}^n(\forall\vec{x}'\exists y' \mathrm{clone}(\vec{x}_i, y_i)\ R\vec{x}'y')$. Lemma 12, then, establishes the claim. q.e.d.

4 Independence-Friendly vs. Partial Fixpoint Logic

By definition, independence-friendly fixpoint logic is a least fixpoint logic. However, contrary to the fixpoint logics usually considered in finite model

theory, here the fixpoints are not sets of elements but sets of trumps and therefore essentially third-order objects. In particular, it is no longer guaranteed that any fixpoint induction closes in polynomially many steps in the size of the structure – to the contrary, it may take an exponential number of steps to close. We will see below, that this greatly increases the expressive power of IF-LFP compared to normal least fixpoint logics.

As a first step in this direction we relate independence-friendly fixpoint logic to partial fixpoint logic. Partial fixpoint logic is an important logic in finite model theory. Among the various fixpoint logics commonly considered in finite model theory, it is the most expressive subsuming logics such as LFP and IFP and, on ordered structures, even second-order logic SO.

Definition 14 (Partial Fixpoint Logic). *Partial fixpoint logic* (PFP) is defined as the extension of first-order logic by the following formula building rule. If $\varphi(R, \vec{x})$ is a formula with free first-order variables $\vec{x} := x_1, \ldots, x_k$ and a free second-order variable R of arity k, then $\psi := [\mathbf{pfp}_{R,\vec{x}}\, \varphi](\vec{t})$ is also a formula, where $\vec{t}$ is a tuple of terms of the same length as $\vec{x}$. The free variables of ψ are the variables occurring in $\vec{t}$ and the free variables of φ other than $\vec{x}$.

Having defined the syntax, we now turn to the definition of the semantics. Let $\psi := [\mathbf{pfp}_{R,\vec{x}}\, \varphi](\vec{t})$ be a formula and let $\mathbf{A}$ be a finite structure with universe A providing an interpretation of the free variables of φ other than $\vec{x}$. Consider the following sequence of stages induced by φ on $\mathbf{A}$, where F_φ is the functional defined by φ.

$$\begin{aligned} R^0 &:= \varnothing \\ R^{\alpha+1} &:= F_\varphi(R^\alpha) \end{aligned}$$

As there are no restrictions on φ, this sequence need not reach a fixpoint. In this case, ψ is equivalent on $\mathbf{A}$ to false. Otherwise, if the sequence becomes stationary and reaches a fixpoint R^∞, then for any tuple $\vec{a} \in A$,

$$\mathbf{A} \models [\mathbf{pfp}_{R,\vec{x}}\, \varphi](\vec{a}) \text{ if, and only if, } \vec{a} \in R^\infty.$$

As mentioned above, PFP is among the fixpoint logics commonly considered in finite model theory the most expressive – especially on ordered structures. A central issue in finite model theory is to relate the expressive power of logics to the computational complexity of classes of structures definable in the logic. Of particular interest are so-called *capturing results*.

Definition 15. A logic $\mathcal{L}$ captures a complexity class $\mathfrak{C}$ if every class of finite structures definable in $\mathcal{L}$ can be decided in $\mathfrak{C}$ and conversely, for every class $\mathcal{C}$ of finite structures which can be decided in $\mathfrak{C}$ there is a sentence $\varphi \in \mathcal{L}$ such that for all structures $\mathbf{A}$, $\mathbf{A} \models \varphi$ if, and only if, $\mathbf{A} \in \mathcal{C}$.

Capturing results are important as in the case that a logic $\mathcal{L}$ captures a complexity class $\mathfrak{C}$, the logic provides a logical characterisation of the complexity class, *i.e.*, a characterisation independent of any machine models or time or space bounds. In particular, non-expressibility results on $\mathcal{L}$ transfer directly into non-definability results on $\mathfrak{C}$. As such results are notoriously hard to come by, capturing results provide an interesting alternative for proving non-definability of problems in a complexity class.

Much effort has been spent on capturing results and for all major complexity classes such results have been found (see [EbbFlu99] for a summary). However, in many cases it could only be shown that a logic captures a complexity class on the class of ordered structures.

Theorem 16 (Abiteboul, Vianu, [AbiVia89]). Partial fixpoint logic captures PSPACE on the class of finite ordered structures.

As every class of structures definable in second-order logic is decidable in the polynomial time hierarchy, it follows immediately that PFP contains SO on ordered structures. One feature that makes PFP so expressive is its ability to define fixpoint inductions of exponential length in the size of the structure. We show next that every formula of PFP is equivalent to one in IF-LFP.

Theorem 17. For every formula $\varphi \in$ PFP there is an equivalent formula $\psi \in$ IF-LFP.

Proof. It is known that every PFP formula is equivalent to one with a single fixpoint, so we need deal only with a PFP formula $\mathbf{pfp}_{R,\vec{x}}\varphi(R,\vec{x})$, where φ is first order. We assume that the fixpoint of φ always exists. See [EbbFlu99] for a proof that both assumptions can be made without loss of generality.

To calculate a partial fixpoint, one needs to check whether two consecutive stages of the inductive approximation are equal, and so one needs to use the stages both positively and negatively. We get round this by building up the relation and its complement simultaneously. Let φ^p be the negation normal form of φ and φ^n be the negation normal form of $\neg\varphi$. Further, let k be the arity of $\vec{x}$, *i.e.*, the number of free variables in φ. Consider the following Σ^1_1 formula $\psi(P,\mathtt{f})$, where P is a third-order relation symbol and $\mathtt{f}$ is a k-ary function symbol:

$$\begin{aligned}\psi(P,\mathtt{f}) := \quad & \forall\vec{x}\big((\varphi^p(\varnothing,\vec{x}) \wedge \mathtt{f}(\vec{x}) = 1) \vee (\varphi^n(\varnothing,\vec{x}) \wedge \mathtt{f}(\vec{x}) = 0)\big) \vee \\ & \exists \mathtt{f}' \in P \forall\vec{x}\big((\varphi^p(\vec{x}, R\vec{u}/\mathtt{f}'(\vec{u}) = 1, \neg R\vec{u}/\mathtt{f}'(\vec{u}) = 0) \wedge \mathtt{f}(\vec{x}) = 1) \\ & \vee(\varphi^n(\vec{x}, R\vec{u}/\mathtt{f}'(\vec{u}) = 1, \neg R\vec{u}/\mathtt{f}'(\vec{u}) = 0) \wedge \mathtt{f}(\vec{x}) = 0)\big)\end{aligned}$$

By $\varphi^p(\vec{x}, R\vec{u}/\mathtt{f}'(\vec{u}) = 1, \neg R\vec{u}/\mathtt{f}'(\vec{u}) = 0)$ we mean the formula obtained from φ^p by replacing every positive occurrence of an atom $R\vec{u}$, for some tuple

$\vec{u}$ of terms, by $\mathtt{f}'(\vec{u}) = 1$ and every negative occurrence of an atom $R\vec{u}$, for some tuple $\vec{u}$ of terms, by $\mathtt{f}'(\vec{u}) = 0$. Thus the formula ψ obtained in this way is positive in P and its least fixpoint exists. We claim that for all functions $\mathtt{f} \in R^\infty$ the set $\{\vec{a} : \mathtt{f}(\vec{a}) = 1\}$ is a stage in the induction on φ. This is clear for the function $\mathtt{f}$ satisfying $((\varphi^p(\varnothing, \vec{x}) \wedge \mathtt{f}(\vec{x}) = 1) \vee (\varphi^n(\varnothing, \vec{x}) \wedge \mathtt{f}(\vec{x}) = 0))$ as $\mathtt{f}$ encodes the first stage of the induction on φ. Further, if $\mathtt{f}' \in P^\infty$ encodes a stage R^α of the induction on φ in the sense described above, then the function $\mathtt{f}$ satisfying

$$(\varphi^p(\vec{x}, R\vec{u}/\mathtt{f}'(\vec{u}) = 1, \neg R\vec{u}/\mathtt{f}'(\vec{u}) = 0) \wedge \mathtt{f}(\vec{x}) = 1) \vee$$
$$(\varphi^n(\vec{x}, R\vec{u}/\mathtt{f}'(\vec{u}) = 1, \neg R\vec{u}/\mathtt{f}'(\vec{u}) = 0) \wedge \mathtt{f}(\vec{x}) = 0)$$

encodes the next stage $R^{\alpha+1}$.

Thus the formula $[\mathbf{pfp}_{R,\vec{x}}\varphi](\vec{x})$ is equivalent to the formula

$$\vartheta(\vec{x}) := \exists \mathtt{f}\ \mathtt{f}(\vec{x}) = 1 \wedge [\mu P(\mathtt{f}).\psi](\mathtt{f}) \wedge \forall \vec{x}(\varphi(\vec{x}, R\vec{u}/\mathtt{f}(\vec{u}) = 1) \leftrightarrow \mathtt{f}(\vec{x}) = 1)$$

stating that there is a function $\mathtt{f}$ in the fixpoint of the Σ^1_1-formula ψ, $\mathtt{f}$ is the partial fixpoint of φ, and $\vec{x}$ occurs in this partial fixpoint, *i.e.*, $\mathtt{f}(\vec{x}) = 1$. Now, the theorem follows from Lemma 12. q.e.d.

We have already mentioned that pure independence-friendly logic is equivalent to Σ^1_1 and therefore an ordering on the universe of a structure can be defined in IF-LFP even on classes of otherwise unordered structures. Thus the theorem above implies that IF-LFP contains SO on all rather than just ordered structures.

Corollary 18. On finite structures, every formula of SO is equivalent to a formula in IF-LFP.

In the next section we will derive some further corollaries of this theorem concerning the model-checking complexity of IF-LFP.

5 Complexity of Independence-Friendly Fixpoint Logic

In this section we analyse the complexity of IF-LFP on finite structures, both with respect to data and model-checking complexity. By data-complexity we understand the complexity of deciding for a fixed formula $\varphi \in$ IF-LFP and a given structure $\mathbf{A}$ whether $\mathbf{A} \models \varphi$. In particular, the input only consists of the structure $\mathbf{A}$. By model-checking we mean the problem of deciding for a given finite structure $\mathbf{A}$ and formula $\varphi \in$ IF-LFP whether $\mathbf{A} \models \varphi$. Here, both φ and $\mathbf{A}$ are part of input.

We begin our analysis with data-complexity. In [Bra03], the first author already noticed that any given formula of IF-LFP can be evaluated in time exponential in the size of the structure. For, every fixpoint $\mu R(\vec{x}).\varphi$ (or $\nu R(\vec{x}).\varphi$) can be evaluated in time linear in the number of trumps for φ and therefore exponential in the size of the structure.

Proposition 19. IF-LFP has exponential time data-complexity.

We aim at a much stronger result. Not only will we show that IF-LFP is ExpTime-complete with respect to data-complexity but we will prove that it actually captures ExpTime, *i.e.*, every class of structures decidable by an exponential time Turing-machine can be defined in IF-LFP and vice versa every class of structures definable in IF-LFP can be decided in deterministic exponential time.

Theorem 20. IF-LFP captures ExpTime.

Proof. We follow the usual approach to show capturing results in finite model theory by simulating the run of a Turing-machine by a fixpoint induction in IF-LFP. Let M be an exponentially time-bounded Turing-machine over the alphabet $\{0,1\}$. On any input of size n, M can make at most 2^{n^k} steps, for some constant k independent of the input. We first show how to simulate the run of M on any input structure $\mathbf{A}$ by an third-order induction on a Σ^1_1-formula, *i.e.*, by a formula of the form $\mu R(\mathtt{s},\mathtt{p},c).\varphi(R,\mathtt{s},\mathtt{p},c)$, where $\varphi \in \Sigma^1_1$, $\mathtt{s}$ and $\mathtt{p}$ are k-ary function symbols, c is an individual variable and R is a third-order relation symbol. (Strictly speaking, individual variables c are not allowed in Σ^1_1-inductions as defined in Definition 9. However, these can easily be replaced by nullary function variables.) Throughout this proof we use typewriter font for function variables and italics font for individual variables.

As an ordering on the universe A of the structure $\mathbf{A}$ is definable in Σ^1_1 we assume without loss of generality that $\mathbf{A}$ is ordered and that there are two constants 0 and 1 interpreted by distinct elements. Further, we assume that we are given an ordering on the space of functions from A^k to $\{0,1\}$. Again such an ordering can easily be defined in Σ^1_1.

Given this ordering on the function space, we can code the content of the Turing-tape in a relation $P(\mathtt{p},c)$ such that $(\mathtt{p},c) \in P$ if, and only if, $\mathtt{p}$ is the ith function with respect to the ordering on the space of k-ary functions, $c \in \{0,1\}$, and the ith cell on the Turing-tape contains c. With this, we can encode the evolution of the Turing-tape during a run of M in a relation $R(\mathtt{s},\mathtt{p},c)$, such that if $\mathtt{s}$ is the ith function with respect to the ordering on the function space, then the set $\{(\mathtt{p},c) : (\mathtt{s},\mathtt{p},c) \in R\}$ encodes the Turing-tape after M has made i steps. To define this relation inductively, we need a formula $\mathtt{init}(\mathtt{p},c)$ which defines the encoding of the

input structure $\mathbf{A}$ on the Turing-tape with the head reading position 0 and a formula $\mathtt{next}(R, \mathtt{s}, \mathtt{p}, c)$ which defines for any given $\mathtt{s}$ in $(\mathtt{p}, c)$ the successor configuration of the configuration stored in R for time step $\mathtt{s}$. Finally, we need a formula $\mathtt{accept}(R, \mathtt{s})$ which is true for $\mathtt{s}$ and R if the configuration coded in R for time step $\mathtt{s}$ is accepting.

Due to space restrictions we refrain from giving the formulae here. Similar formulae are widely used in the finite model theory community to encode the run of polynomial time Turing-machines in LFP. See [Grä$_0$07] or [EbbFlu99]. The Σ^1_1-formulae needed here can be obtained from these via trivial modifications implementing the above encoding of Turing-tapes and time steps. Now the run of M on input $\mathbf{A}$ is accepting if, and only if, the formula

$$\exists \mathtt{s}, \mathtt{p}\ [\mu R(\mathtt{s}, \mathtt{p}, c).\ \begin{array}{l}(\mathtt{s} = \text{MIN} \wedge \mathtt{init}(\mathtt{p}, c)) \vee \\ (\exists \mathtt{s}'\ (\mathtt{s} = \mathtt{s}' + 1 \wedge \mathtt{next}(\mathtt{s}', \mathtt{p}, c))) \vee\ ](\mathtt{s}, \mathtt{p}, \text{ACC}) \\ (\exists \mathtt{saccept}(R, \mathtt{s}) \wedge c = \text{ACC})\end{array}$$

is true in $\mathbf{A}$, where MIN denotes the minimal element in the ordering on the function space, $+1$ refers to the successor relation with respect to this ordering, and ACC is an arbitrary element distinct from 0 and 1 used to mark an accepting configuration. q.e.d.

Clearly, if a logic $\mathcal{L}$ captures a complexity class $\mathfrak{C}$, then the evaluation problem of $\mathcal{L}$ must be $\mathfrak{C}$-complete with respect to data complexity. Thus we get the following simple corollary.

Corollary 21. IF-LFP has ExpTime-complete data-complexity.

Capturing results relate the expressive power of a given logic to the computational complexity of the classes of structures definable in the logic. The study of data complexity corresponds to the study of the computational complexity of a problem, where the size of the program or algorithm used to solve a problem is ignored.

However, when actually evaluating a formula in a structure this approach is not satisfactory. For instance, monadic Datalog and monadic second-order logic (MSO) have the same expressive power on trees and therefore the same data-complexity on trees, but whereas monadic Datalog programs can be evaluated in time linear both in the size of the Datalog program and the input tree, the evaluation of MSO-formulae is PSpace-complete on trees. In fact, it was shown in [GroSch$_{10}$03] that any translation of a MSO-formula on trees to an equivalent monadic Datalog program necessarily increases the formula size non-elementary.

Thus, data-complexity only gives limited information about the complexity of actually evaluating a formula in a structure. We therefore continue

our complexity analysis of IF-LFP with the study of its model-checking complexity. In particular, we will prove that model-checking for IF-LFP is hard for exponential space. For an upper bound, it is easily seen that for any given structure $\mathbf{A}$ and formula φ the formula can be evaluated in $\mathbf{A}$ using space doubly exponential in $|\varphi|$ and exponential in $|\mathbf{A}|$. For, every evaluation of a (least or greatest) fixpoint only needs enough space to store all possible trumps, and the number of trumps is bounded by $O(2^{\mathbf{A}^{|\varphi|}})$.

Theorem 22. Every formula $\varphi \in$ IF-LFP can be evaluated in a structure $\mathbf{A}$ in space doubly exponential in $|\varphi|$ and exponential in $|\mathbf{A}|$.

The theorem gives an upper bound on the model checking complexity of IF-LFP. We have seen in Section 4 above that every formula of PFP is equivalent to one of IF-LFP. Further, the translation is polynomial in the size of the PFP-formula. Consequently, model-checking for IF-LFP is at least as complex as it is for PFP. As model-checking for PFP is known to be hard for exponential space – in fact even complete for exponential space – we get the following theorem.

Theorem 23. The model-checking problem for IF-LFP is hard for exponential space.

6 Conclusion

In this paper we studied the computational complexity of various problems related to IF-LFP. As we have seen, adding independence to least fixpoint logic increases the expressive power and complexity significantly. Another indicator for this is the translation of formulae of PFP to formulae of IF-LFP. This showed that IF-LFP is even more expressive than second-order logic – unless, of course, PSPACE = EXPTIME.

Looking at the various proofs given for the results, it becomes clear that the common technique used in all proofs was to use independent quantification to define functions and then show that these functions can be passed through the fixpoint induction. This suggests that there might be a more general relation between independence-friendly logic and second-order logic, namely that the two logics are actually equivalent. Showing this, however, requires a careful analysis of the rôle of negation in independence friendly logics and is far from obvious. This is part of ongoing work.

It is also notable that all our hardness results involve constructions requiring only least fixed points. For LFP, it is well-known that all properties can be expressed with a single least fixed point. An analogous theorem of IF-LFP has not been shown.

References.

[AbiVia89] Serge **Abiteboul** and Victor **Vianu**, Fixpoint Extensions of First-Order Logic and Datalog-like Languages, *in:* [Mey89, p. 71–79]

[BaaMak$_1$03] Matthias **Baaz** and Johann A. **Makowsky** (*eds.*), Computer Science Logic, Proceedings of the 17th International Workshop, CSL 2003, 12th Annual Conference of the EACSL, and 8th Kurt Güdel Colloquium, KGC 2003, Vienna, Austria, August 25-30, 2003, Springer 2003 [Lecture Notes in Computer Science 2803]

[Bra99] Julian C. **Bradfield**, Fixpoints in Arithmetic, Transition systems and Trees, **Theoretical Informatics and Applications** 33 (1999), p. 341–356

[Bra00] Julian C. **Bradfield**, Independence: Logics and Concurrency, *in:* [CloSch$_{11}$00, p. 247–261]

[Bra03] Julian C. **Bradfield**, Parity of Imperfection, *in:* [BaaMak$_1$03, p. 72–85]

[BraFrö02] Julian C. **Bradfield** and Sibylle B. **Fröschle**, Independence-Friendly Modal Logic and True Concurrency, **Nordic Journal of Computing** 9 (2002), p. 102–117

[CloSch$_{11}$00] Peter **Clote** and Helmut **Schwichtenberg** (*eds.*), Computer Science Logic, Proceedings of the 14th Annual Conference of the EACSL, Fischbachau, Germany, August 21-26, 2000, Springer 2000 [Lecture Notes in Computer Science 1862]

[EbbFlu99] Heinz-Dieter **Ebbinghaus** and Jörg **Flum**, Finite Model Theory, 2nd edition, Springer 1999

[End70] Herbert B. **Enderton**, Finite Partially Ordered Quantifiers, **Zeitschrift für Mathematische Logik und Grundlagen der Mathematik** 16 (1970), p. 393–397

[Grä$_0$07] Erich **Grädel**, Finite Model Theory and Descriptive Complexity, *in:* [Grä$_0$+07, p. 125–230],

[Grä$_0$+07] Erich **Grädel**, Phokion G. **Kolaitis**, Leonid **Libkin**, Maarten **Marx**, Joel **Spencer**, Moshe Y. **Vardi**, Yde **Venema**, and Scott **Weinstein**, Finite Model Theory and Its Applications, Springer 2007 [EATCS Series Texts in Theoretical Computer Science]

[GroSch$_{10}$03] Martin **Grohe** and Nicole **Schweikardt**, Comparing the Succinctness of Monadic Query Languages over Finite Trees, *in:* [BaaMak$_1$03, p. 226–240]

[HinSan$_1$96] Jaakko **Hintikka** and Gabriel **Sandu**, A Revolution in Logic?, **Nordic Journal of Philosophical Logic** 1 (1996), p. 169–183

[Hod$_0$97] Wilfried **Hodges**, Compositional Semantics for a Language of Imperfect Information, **Logic Journal of the IGPL** 5 (1997), p. 539–563

[Mey89] Albert **Meyer** (*ed.*), Proceedings of the 4th Annual IEEE Symposium on Logic in Computer Science, Asilomar, California, June 5-8, 1989, IEEE Computer Society Press 1989

[Pie01] Ahti-Veikko **Pietarinen**, Semantic Games in Logic and Language, *PhD thesis*, University of Helsinki 2001

[Wal70] Wilbur J. **Walkoe**, Finite Partially-Ordered Quantification, **The Journal of Symbolic Logic** 35 (1970), p. 535–555

Received: April 2nd, 2005;
In revised version: September 16th, 2005;
Accepted by the editors: October 31st, 2005.