Fixed point formulae and solitaire games

Dietmar Berwanger and Erich Grädel
Mathematische Grundlagen der Informatik
RWTH Aachen

Abstract

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

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

1 Introduction

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

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

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

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

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

2 Least fixed point logic

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3 Model checking games

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

(1) Computing winning sets in parity games.

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

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

4 Solitaire games

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

5 Solitaire formulae in least fixed point logic

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

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

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

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

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

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

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

5.1 Complexity

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

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

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

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

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

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

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

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

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

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

5.2 Expressive power

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

and, if applicable,

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

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

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

and, if applicable,

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

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

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

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

for appropriate **gfp**-variables T.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

This completes the proof of Theorem 5.6.

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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