Linear Functional Fixed-Points

Nikolaj Bjørner and Joe Hendrix

Microsoft, One Microsoft Way, Redmond, WA, 98074, USA
{nbjorner,johendri}@microsoft.com

Abstract. We introduce a logic of functional fixed-points. It is suitable for analyzing heap-manipulating programs and can encode several logics used for program verification with different ways of expressing reachability. While full fixed-point logic remains undecidable, several subsets admit decision procedures. In particular, for the logic of linear functional fixed-points, we develop an abstraction refinement integration of the SMT solver Z3 and a satisfiability checker for propositional linear-time temporal logic. The integration refines the temporal abstraction by generating safety formulas until the temporal abstraction is unsatisfiable or a model for it is also a model for the functional fixed-point formula.

1 Introduction

Software often manipulates heap allocated data structures of finite but potentially unbounded size, such as linked lists, doubly linked lists, and trees. To reason about such structures, invariants about the *reachable* heap contents can be necessary. Logics capable of expressing interesting heap properties often require some form of transitive closure, fixed-points, and/or 2^{nd}-order quantification. As is well known, complete first-order axiomatization of transitive closure is impossible [10], though approximations that suffice for ground validity of some fragments have been formulated. The approximations work directly with theories supported in the same (first-order) setting, but must rely on the capabilities of the generic first-order engine. A different approach is to directly use non-first order logics and rely on specialized decision procedures for these logics. Such specialized decision procedures do not suffice in practice when the invariants also require reasoning in the theories of arithmetic and arrays.

Contributions. This paper analyzes several different fixed-point logic fragments to identify expressive logics that still have good decidability and complexity results. On the practical side, we outline an integration procedure between propositional temporal logic checking and theory solvers.

– We formulate a logic called the *Equational Linear Functional Fixed Point Logic* (or $\mathsf{FFP(E)}$ for short). $\mathsf{FFP(E)}$ encodes several fixed point logics presented in recent literature on program verification.
– We establish that $\mathsf{FFP(E)}$ is PSPACE-complete modulo background theories that are in PSPACE by using a reduction from $\mathsf{FFP(E)}$ into propositional linear-time temporal logic. We show that two different extensions are NEXPTIME-hard and undecidable, respectively.

– We provide a decision procedure for FFP(E) that combines the SMT solver Z3 with a (symbolic) satisfiability checking of propositional linear time temporal formulas. The proposed integration generalizes the standard abstraction/refinement framework used in SMT solvers. Instead of relying on refining a propositional model, we here refine a propositional linear time model. An early stage prototype of the procedure is available.

The resulting approach can therefore be viewed as a marriage between the flexible axiomatization approach to fixed-points and specialized decision procedures. Our abstraction/refinement framework admits all axiomatizations allowed by other approaches, but furthermore provides a decision procedure for formulas that fall into FFP(E).

Example 1 (A simple example). *We illustrate the use of reachability predicates using a simple example also used in [16]. It exercises transitivity. We use* $\forall x : [a \xrightarrow{f} b].\varphi(x)$ *to say that* $f^n(a) \simeq b$ *for some* n*, and for every* $k < n$ *it is the case that* $\varphi(f^k(a))$*.*

```
procedure INIT-CYCLIC(head)
    $d(head) :=$ true; $curr := f(head)$;
    invariant $d(head) \wedge \forall x : [f(head) \xrightarrow{f} curr].d(x)$
    while $curr \neq head$ do
        $d(curr) :=$ true
        $curr := f(curr)$
    ensure $d(head) \wedge \forall x : [f(head) \xrightarrow{f} head].d(x)$
```

The invariant and post-condition can be established by verifying properties:

$$\forall x : [f(head) \xrightarrow{f} curr].d(x) \wedge d(curr) \;\rightarrow\; \forall x : [f(head) \xrightarrow{f} f(curr)].d(x)$$
$$head \simeq curr \wedge \forall x : [f(head) \xrightarrow{f} curr].d(x) \;\rightarrow\; \forall x : [f(head) \xrightarrow{f} head].d(x)$$

While these particular properties hardly require the full might of transitive closure reasoning, we are here interested in characterizing the limits of what can be solved in a sufficiently general language with fixed-points.

Related work. We refer to [3] for an extended summary of the extensive related work. An early paper was by Greg Nelson [14], who gave 8 axioms for a ternary reachability predicate. The axioms are sufficient for a verification example, but general completeness with respect to ground validity was left as an open question. Several recent extensions and variants for ground validity have been pursued in [12,9,17,16,8]. These also develop first-order axiomatizations and rely on specialized rewriting or quantifier instantiation engines for their rules. The approach based on first-order axioms is of course quite extensible, as one can throw in useful axioms at will without requiring an encoding into a fixed limited formalism. On the other hand, the approach is only as viable as the strength of the quantifier instantiation heuristics. Balaban et al. [2] use a small model theorem to derive a decision procedure. A different line of work is based on automata-based decision procedures. The PALE system [13] can reason about heap-allocated data structures using weak monadic second-order logic of graph types. The logic of reachable patterns [21] is a decidable and quite expressive logic that combines local reasoning with an extended form of regular expressions. Finally in several practical cases, Separation Logic [18] provides a compelling alternative to reachability predicates.

Paper structure. The rest of this paper is structured as follows. In Section 2, we formally define functional fixed-point logic (FFP), and briefly review results from temporal logic used later in the paper. In Section 3, we study different fragments of FFP to obtain decidability and complexity results. Our main focus in this section is to define linear functional fixed-point logic with equality, FFP(E). We also show that FFP(E) is closed under updates, subsumes several different logics for reasoning about heap invariants, and has a PSPACE-complete satisfiability problem. In Section 4, we describe our reference satisfiability solver for FFP(E) which works by integrating the SMT-based theorem prover Z3 with a decision procedure for propositional LTL. Finally, in Section 5, we summarize our results and discuss ways our results can be extended in future research.

2 Preliminaries

Our results are mainly based on a reduction of FFP fragments into propositional linear-time temporal logic (LTL); and we rely on decision procedures for it. LTL [11] augments propositional logic with the temporal connectives $\mathcal{U}$, $\bigcirc$, $\Box$ and $\Diamond$. LTL models are represented as an infinite sequence of states $\sigma : s_0, s_1, s_2, \ldots$, where each state supplies an assignment to propositional atoms. Recall that the operator $\mathcal{U}$ is the least fixed-point solution to the equivalence $A\,\mathcal{U}\,B \equiv (B \vee [A \wedge \bigcirc(A\,\mathcal{U}\,B)])$, or directly: $A\,\mathcal{U}\,B \equiv \mu X\ .\ B \vee (A \wedge \bigcirc X)$.

Functional Fixed-point Logic (FFP) extends quantifier-free first-order logic with the fixed-point operator μ to define the least fixed-point of unary predicates. To be more specific, we let x range over bound variables, X ranges over bound unary predicates, f and g range over distinguished unary uninterpreted function symbols, a, b, c, c' range over constant terms, P ranges over unary predicates, R over predicates containing neither bound variables, nor the function symbols f, g. Then the set of formulas φ in FFP are given by the rules:

$$t ::= f(t) \mid g(t) \mid c \mid x \qquad atom ::= X(t) \mid t \simeq t' \mid P(t) \ \mid R$$

$$A, B, \psi, \psi', \varphi ::= atom \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid (\mu X\ .\lambda x\ .\ \ \varphi^{+}[X])(t)$$

where $\varphi^{+}[\_]$ is a positive context in φ.

The semantics of FFP follows the standard rules for evaluating fixed-point expressions. For example, a model $\mathcal{M}$ over a domain $\mathcal{A}$ satisfies $(\mu X\ .\lambda x\ .\ \ \varphi[X])(t)$ if $\mathcal{M}(t) \in \bigcap\{\mathcal{B} \subseteq \mathcal{A} \mid \mathcal{M}, [X \mapsto \mathcal{B}] \models \forall x\ .\ \varphi[X] \ \ \rightarrow\ \ X(x)\}$. FFP allows multiple different unary function symbols to be applied to the same bound variables, and allows multiple bound second-order predicates to appear in the same scope.

We will here restrict ourselves to a more modest fragment inspired by Linear Time Temporal Logic (LTL). In this fragment each fixed point expression has the form:

$$\mu X.\lambda x.(B \;\vee\; [A \wedge X(f(x))]),$$

where A and B are formulas that do not contain X, but may contain x. Intuitively, the function f is used as a next state transition.

Convention 2. *The following shorthands will be used throughout the paper:*

$$(A\,\mathcal{U}_{f,x}\,B)\,(c) = [\mu X.\lambda x.B \vee [A \wedge X(f(x))]]\,(c)$$

$$\begin{array}{rclrcl} (A\,\mathcal{W}_{f,x}\,B)\,(c) &=& \neg\,(\neg B\,\mathcal{U}_{f,x}\,\neg A \wedge \neg B)\,(c) & (\Box_{f,x}\,B)\,(c) &=& \neg((\Diamond_{f,x}\,\neg B)\,(c)) \\ \forall x : [a \xrightarrow{f} b].A &=& (A\,\mathcal{U}_{f,x}\,x \simeq b)\,(a) & (\Diamond_{f,x}\,B)\,(c) &=& (\text{true}\,\mathcal{U}_{f,x}\,B)\,(c) \\ \widetilde{\forall} x : [a \xrightarrow{f} b].A &=& (A\,\mathcal{W}_{f,x}\,x \simeq b)\,(a) & a \xrightarrow{f} b &=& (\Diamond_{f,x}\,x \simeq b)\,(a) \end{array}$$

The connective $\mathcal{W}$ is inspired by the weak until *connective from LTL here, A holds either forever or until B is reached. We also include strong and weak versions for the case when A holds on every value between a and b.*

Convention 3. *The set of subformulas of a formula φ is denoted* $\mathrm{SF}(\varphi)$*. The set of atomic subformulas of φ is denoted* $\mathrm{ASF}(\varphi)$*.*

We will later establish that formulas in this more modest fragment are in general undecidable, and so we will study various subsets of it. Of particular utility is restricting the number of free variables that a formula may contain. We say that a formula φ is *linear* if each subformula $\psi \in \mathrm{SF}(\varphi)$ has at most one free variable. As an example, the formula stating that c reaches an infinite number of elements, $(\Box_{f,x}\,(\Box_{f,y}\,y \not\simeq x)\,(f(x)))\,(c)$, is not a linear formula, because $y \not\simeq x$ has two free variables.

Normal forms of linear formulas. When a formula is linear, we can rename the variables in the formula to achieve a normal form which we use to simplify later exposition. Specifically, we give the same name to variables bound in nested quantifiers, while giving different names to variables bound in unrelated contexts. We are going to unfold fixed-points incrementally. In this context it is going to be useful to have an anchor on the bound variables. Therefore, for each top-level application of $(\varphi\,\mathcal{U}_{f,x}\,\psi)\,(t)$ we can introduce a fresh constant x that has the same name as the bound variable x, replace t by the variable, and add the constraint that $x \simeq t$. Thus, for instance

$$[x \not\simeq a\,\mathcal{U}_{f,x}\,P(x) \wedge (y \not\simeq b\,\mathcal{U}_{f,y}\,\neg P(y))\,(x)]\,(c) \wedge (\Diamond_{g,x}\,x \simeq b)\,(c)$$

is converted into

$$\begin{aligned}&[x \not\simeq a\,\mathcal{U}_{f,x}\,P(x) \wedge (x \not\simeq b\,\mathcal{U}_{f,x}\,\neg P(x))\,(x)]\,(x) \wedge x \simeq c \\ &\qquad\qquad\qquad\qquad\qquad\qquad \wedge (\Diamond_{g,y}\,y \simeq b)\,(y) \wedge y \simeq c.\end{aligned}$$

This transformation allows us to distinguish variables occurring in unrelated fixed-point expressions while identifying variables occurring in related sub-expressions.

Atomic formulas containing unbound variables are called *flexible*; otherwise, they are called *rigid* atoms. For example $x \simeq c$ and $ff(x) \simeq x$ are flexible atoms, while $c \simeq f(c')$ and $P(f(c))$ are rigid atomic formulas.

We will use a shorthand $ⓕ$ for distributing an application f over all free variables. For example, $ⓕ(x \simeq f(x) \wedge ⓕ(\psi \,\mathcal{U}_{f,x}\, \psi')\,(x))$ is short for $(f(x) \simeq f(f(x)) \wedge (\psi \,\mathcal{U}_{f,x}\, \psi')\,(f(f(x))))$.

3 Complexity Results for FFP Logics

We will here introduce various variants of FFP and summarize relevant complexity results. Fig. 1 summarizes how the examined fragments relate to each other in terms of generality. 2FFP(E) is the fragment of linear FFP allowing at most two functions f and g to be nested inside fixed-point operators. FFP(NL) does not allow nesting of functions inside fixpoint operators, but does allow non-linear subformulas. We will show that satisfiability of 2FFP(E) is undecidable while satisfiability of FFP(NL) is NEXPTIME-hard. FFP(E) is the linear fragment of FFP(NL), and FFP(PL) is the purely propositional fragment of FFP(E).

3.1 FFP(PL)

We first study the propositional fragment of FFP, called FFP(PL). It corresponds very closely to linear time temporal logic, the only real difference is that the temporal subformulas refer to an explicit *anchor*, as a constant.

Formulas in FFP(PL) have the form:

$$\begin{aligned} \varphi &::= P(t) \mid R \mid \varphi \wedge \varphi \mid \neg\varphi \mid \ (\varphi \,\mathcal{U}_{f,x}\, \varphi)\,(t) \\ t &::= f^n(x) \mid f^n(c) \end{aligned}$$

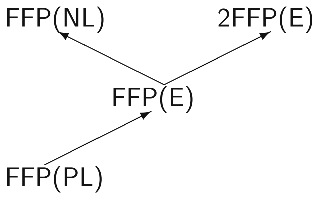

Fig. 1. Relative expressiveness of the FFP fragments

where x is a variable, $f^n(x)$ is the n-time application of f to x, c is an arbitrary *rigid* term (without variables), P is a unary predicate, and R is a relation using only rigid terms.

FFP(PL) is formulated to be very similar to propositional LTL. It is indeed very straightforward to translate formulas from LTL to FFP(PL) and to translate formulas from FFP(PL) into equisatisfiable formulas in LTL. The correspondence can be used to establish:

Theorem 4. FFP(PL) *is PSPACE complete.*

Proof (Sketch). We can embed LTL into FFP(PL) using transformations, such as $\Diamond\bigcirc P \mapsto (\Diamond_{f,x}\, P(f(x)))\,(x)$. Conversely, we can translate FFP(PL) formulas into LTL by dropping the explicit variable references and annotating predicates based on the context they appear, e.g.,

$$(\Diamond_{f,y}\, P(y))\,(d) \mapsto \Diamond P_d \quad \text{and} \quad (P(x) \,\mathcal{U}_{f,x}\, Q(f(x)))\,(ff(c)) \mapsto \bigcirc\bigcirc(P_c \,\mathcal{U} \bigcirc Q_c).$$

By reusing results from [19], we can take advantage of the reduction to LTL and obtain a few results on NP-complete subsets "for free". For instance, Sistla and Clarke show that linear-time temporal logic using only the operators $\Box$ and $\Diamond$ is NP-complete, and so is the case if formulas are in positive normal form using $\Diamond$ and $\bigcirc$. The corresponding $\mathsf{FFP(PL)}$ subsets are therefore also NP-complete.

3.2 $\mathsf{FFP(E)}$

We will now consider an extension of $\mathsf{FFP(PL)}$ by admitting equality predicates on terms containing bound variables. The resulting logic is called $\mathsf{FFP(E)}$. In contrast to $\mathsf{FFP(PL)}$, the embedding into LTL is less straightforward, since the equalities interact in an essential way with the models for the propositional temporal abstraction. Formulas in $\mathsf{FFP(E)}$ extend $\mathsf{FFP(PL)}$ by admitting atomic formulas that are equalities between terms containing variables, constants, and distinguished functions used in fixed-points. We use f and g to refer to the distinguished functions. For simplicity we will assume that formulas in $\mathsf{FFP(E)}$ use just two functions f and g in the fixed-points. The generalization to multiple functions is simple.

The operators $\textcircled{\scriptsize f}$ and $\textcircled{\scriptsize g}$ are used to limit the number possible equality predicates to consider. We also admit terms where the distinguished functions f and g are applied to a constant. Thus, formulas of $\mathsf{FFP(E)}$ are of the form:

$$\begin{aligned}\varphi, \varphi' ::= f^n(x) \simeq x \mid g^n(x) \simeq x \mid c \simeq f(c') \mid c \simeq g(c') \mid \textcircled{\scriptsize g}\varphi \mid \textcircled{\scriptsize f}\varphi \mid x \simeq c \mid c \simeq c' \\ \mid \; P(x) \mid R \mid \neg\varphi \mid \varphi \vee \varphi' \mid \varphi \wedge \varphi' \mid (\varphi \,\mathcal{U}_{f,x}\, \varphi')(x) \mid (\varphi \,\mathcal{U}_{g,x}\, \varphi')(x)\end{aligned}$$

such that the formulas φ and φ' in $(\varphi \,\mathcal{U}_{f,x}\, \varphi')(x)$ contain at most one free variable, which is x. So we require every bound variable to appear linearly.

We furthermore restrict applications of f and g so that they do not appear together in the same flexible formula. Specifically, for each subformula $(\varphi \,\mathcal{U}_{f,x}\, \varphi')(t)$, g may not appear in φ or φ'. A similar condition is also required where the roles of f and g are exchanged. Thus,

$$(\Diamond_{f,x}\, x \simeq a)(c) \wedge (\Diamond_{g,y}\, y \simeq b)(c)$$

is a legal formula in $\mathsf{FFP(E)}$, but

$$(\Diamond_{f,x}\, x \simeq a)(c) \wedge (\Diamond_{g,y} \textcircled{\scriptsize f}(y \simeq b))(c)$$

is not, because both f and g are used on the same variable y. In general, this restriction is necessary to ensure our complexity result in Theorem 10. However, when g does not directly refer to the same variable x, one can introduce fresh rigid predicates to normalize the formula into an equisatisfiable formula with this restricted form. For example,

$$(\Diamond_{f,x}\, x = c \wedge (\Box_{g,x}\, P(x))(b))(a)$$

can be expressed as the equisatisfiable formula

$$(\Diamond_{f,x}\, x = c \wedge r)(a) \wedge (r \iff (\Box_{g,x}\, P(x))(b)).$$

It is not hard to see that we can build all combinations of equality predicates using one variable x, the function f and up to two constants c and c' using the base cases $f^n(x) \simeq x$, $x \simeq c$, $c \simeq f(c')$, $c \simeq c'$ and the operator $\textcircled{f}\psi$. For example $\varphi[f(c) \simeq ff(c'), f(x) \simeq c]$ is equisatisfiable to the formula $c_1 \simeq f(c) \wedge c_2 \simeq f(c') \wedge \varphi[c_1 \simeq f(c_2), \textcircled{f}(x \simeq c)]$. We will use the operator $\textcircled{f}$ in two ways: in a *temporal view* and a *ground view.* In the temporal view, we do not normalize the formula with respect to the definition of $\textcircled{f}$; the use of $\textcircled{f}$ is essential for bridging FFP(E) with LTL. In the ground view, we distribute $\textcircled{f}$ over the free variables such that $\textcircled{f}$ gets eliminated.

Even very small examples can show how the interaction between equalities makes checking the satisfiability of FFP(E) more complex. For example, the following unsatisfiable formula illustrates how distinct constants can not be as easily partitioned as was done in FFP(PL):

$$(\Diamond_{f,x}\, x \simeq c \wedge \textcircled{f}P(x))\,(a) \wedge (\Diamond_{f,x}\, x \simeq c \wedge \neg\textcircled{f}P(x))\,(b).$$

The first conjunct implies that $P(f(c))$, but the second required $\neg P(f(c))$, although $P(f(c))$ does not directly appear in the formulas.

To illustrate how equalities and predicates may interact, consider the formula:

$$\left(\Diamond_{f,x}\, x \simeq f^3(x)\right)(a) \wedge \left(\Diamond_{f,x}\, x \simeq f^2(x)\right)(a) \wedge (\Box_{f,x}\, (\textcircled{f}P(x) \iff \neg P(x)))\,(a)$$

The first two conjuncts require that $f^{i+3}(a) = f^i(a)$ and $f^{j+2}(a) = f^j(a)$ for some i and j in $\mathbb{N}$. Collectively, this implies that $f(f^k(a)) = f^k(a)$ for $k \geq \min(i, j)$. Conversely, the third clause requires that the value of P changes at each dereference, and consequently, $f(f^k(a))$ cannot equal $f^k(a)$.

Expressivity of FFP(E). We make a case that the FFP(E) logic is quite general and expressive. It subsumes several (but not all) logics recently proposed for reasoning about heaps. We summarize some of the properties that can be expressed in FFP by encoding logics from the literature on verification of heap manipulating programs.

Example 5 (Transitive Closure of f). *Suppose we let $f^*(a, b)$ mean that there is a sequence of 0 or more applications of f to a that produces an element b. That is, $f^*(a, b) \equiv \exists n\ .\ f^n(a) \simeq b$. This can be easily represented:*

$$f^*(a, b) \equiv (\Diamond_{f,x}\, x \simeq b)\,(a) \equiv a \xrightarrow{f} b$$

Example 6 (Reachability Invariants [14]). *Nelson introduced a ternary predicate $u \xrightarrow[w]{f} v$ which indicates that u reaches v without going through w. It can be used to verify a program that computes the union of two sets represented as doubly linked lists, and can be expressed in* FFP(E) *as: $u \xrightarrow[w]{f} v \equiv (x \not\simeq w\ \mathcal{U}_{f,x}\, x \simeq v)\,(u)$.*

Example 7 (Well-Founded Reachability [7]). *Lahiri and Qadeer use* blocking set *predicates BS to identify distinguished elements in potentially cyclic data structures. Blocking sets generalize Nelson's blocking variable. Under the assumption that every node eventually reaches a blocker, they can define the program* *verification-friendly functions and predicates:* $\mathsf{FFP(E)}$ *allows formulating these predicates directly:*

$$\begin{aligned} B(u) \simeq v &\equiv (\neg BS(x)\ \mathcal{U}_{f,x}\ x \simeq v)\,(u) \wedge BS(v) \\ R(u,v) &\equiv (\neg BS(f(x))\ \mathcal{U}_{f,x}\ x \simeq v)\,(u) \end{aligned}$$

Example 8 (btwn_f **[16]).** *The dual to Nelson's reachability predicate is the predicate* $\mathrm{btwn}_f(a,b,c)$ *which requires* b *is visited before* c*. It is versatile for a wide range of program verification cases, and can be defined as:*

$$\mathrm{btwn}_f(a,b,c) \equiv (x \not\simeq c\ \mathcal{U}_{f,x}\ x \simeq b)\,(a) \wedge b \xrightarrow{f} c$$

$\mathsf{FFP(E)}$ is also closed under the weakest-precondition predicate transformer, when a pointwise update is made to the function f:

Proposition 9. $\mathsf{FFP(E)}$ *is closed under pointwise functional updates. As we show in [3]: if* $\varphi[f]$ *is in* $\mathsf{FFP(E)}$*, then for constants* u *and* v*, the formula* $\varphi[\lambda y.\mathtt{if}\ y \simeq u\ \mathtt{then}\ v\ \mathtt{else}\ f(y)]$ *can also be expressed in* $\mathsf{FFP(E)}$*.*

Theorem 10. *The satisfiability problem of* $\mathsf{FFP(E)}$ *is PSPACE-complete.*

Proof (Sketch [3]). We reduce $\mathsf{FFP(E)}$ to LTL, and use the PSPACE-completeness result from [19]. The reductions follow 3 transformations Tableau, Erase, and Embed:

$$\begin{tikzcd} \varphi \arrow[r, "\text{Tableau}"] & Tab(\varphi) \arrow[r, "\text{Erase}"] & \varphi_{PTL} \arrow[r, "\text{Embed}"] & \varphi_{PTL*} \end{tikzcd}$$

Each of the transformations can be done in polynomial time and space, with the last transformation being the most expensive, requiring quadratic space.

Tableau. The tableau normal form [11] of a formula φ is a conjunction of the form:

$$Tab(\varphi) : \ \lceil\varphi\rceil \wedge Next \wedge Inv \wedge \bigwedge_{F \in \mathcal{F}} \Box\Diamond F$$

where $\lceil\varphi\rceil$ replaces the top-most occurrences of $\textcircled{f}$ and $\mathcal{U}_{f,x}$ subformulas by fresh auxiliary propositional variables, $Next$ encodes an accessibility relation over the original and auxiliary propositional variables, Inv is a state invariant that ensures that the interpretation of temporal connectives is consistent on every state; and the set $\mathcal{F}$ contains the set of acceptance conditions:

$$\lceil(\psi\ \mathcal{U}_{f,x}\ \psi')\rceil(x) \ \rightarrow\ \lceil\psi'\rceil \ \text{ for each } (\psi\ \mathcal{U}_{f,x}\ \psi')\,(x) \in \mathrm{SF}(\varphi).$$

Erase. The propositional erasure converts an $\mathsf{FFP(E)}$ formula φ into an LTL formula φ_{PTL} by using the transformations:

$$((\psi\ \mathcal{U}_{f,x}\ \psi')\,(x))_{PTL} = \psi_{PTL}\,\mathcal{U}\,\psi'_{PTL} \qquad \text{and} \qquad (\textcircled{f}\psi)_{PTL} = \bigcirc\psi_{PTL}.$$

Embed. The heart of the reduction is to add a sufficient set of axioms such that the LTL formula is equisatisfiable with the $\mathsf{FFP(E)}$ formula. The formula φ_{PTL*} is obtained by enumerating the atomic subformulas of φ and adding the conjunctions, whenever there are atomic formulas matching the preconditions of the implications:

$$
\begin{aligned}
x \simeq c \;\Rightarrow\; \bigcirc\,(x \not\simeq c\;\mathcal{W}\;F) \quad F \in \mathcal{F}(x) & \qquad (1)\\
f^n(x) \simeq x \Rightarrow (\ell(x) \leftrightarrow \bigcirc^n \ell(x)) & \qquad (2)\\
f^n(x) \simeq x \wedge f^m(x) \simeq x \;\Rightarrow\; f^{\gcd(m,n)}(x) \simeq x & \qquad (3)\\
f^n(x) \not\simeq x \wedge x \simeq c \;\Rightarrow\; \bigcirc^n (x \not\simeq c) & \qquad (4)\\
f^n(x) \not\simeq x \wedge f^m(x) \simeq x \;\Rightarrow\; f^{n-m}(x) \not\simeq x \quad n > m & \qquad (5)\\
f^n(x) \simeq x \Rightarrow \bigcirc(f^n(x) \simeq x) & \qquad (6)\\
x \simeq c \wedge \ell(x) \;\Rightarrow\; R_{\ell,c} & \qquad (7)\\
x \simeq c \;\Rightarrow\; (x \simeq c' \;\leftrightarrow\; c \simeq c') & \qquad (8)\\
c \simeq f(c') \wedge x \simeq c' \;\Rightarrow\; \bigcirc(x \simeq c) & \qquad (9)\\
x_{f,c} \simeq c \wedge (R_{\ell c} \;\rightarrow\; \ell(x_{f,c})) & \qquad (10)\\
\Box E{-}taut & \qquad (11)
\end{aligned}
$$

Let us motivate the conditions a bit. The condition (1) ensures that all acceptance conditions mentioning x are visited between two anchor states where x is equal to a constant anchor c. It can still be the case that f is periodic on x, but x is not constrained to be a constant; conditions (2), (3), and (6) handle such cases and enforce that once f is periodic it remains so, it satisfies congruence closure over the period, and that states are identical after n steps. Conditions (4) and (5) ensure that the state literals are consistent with disequalities $f^n(x) \not\simeq x$. The last set of invariants (11) are the set of tautologies needed to axiomatize the theory of equality for the rigid predicates. They are the counterpart to the axioms (7)–(9) which are used to axiomatize the theory of equality inside temporal subformulas. The conditions (7) and (10) are used to force evaluations after each constant anchor to be consistent across different variables. They use the rigid predicate $R_{\ell,c}$, which is introduced for every literal $\ell(x)$ and constant c. Suppose that one state s_1 requires the variable x to satisfy $x \simeq c$. Another state s_2 requires a different variable y to satisfy $y \simeq c$. These states must satisfy the same literals $\ell(x)$ respectively $\ell(y)$. To ensure that also the states following s_1 and s_2 evaluate the same literals in tandem we use condition (10) which uses the new variable $x_{f,c}$ and will involve all literals associated with x and y, and any other variable that anchors with c.

Let us assume that the variables are x, y, z. To extract a model $\mathcal{M}$ of φ from an LTL model $\sigma = s_1, s_2, s_3 \ldots$ of φ_{PTL*}, we partition each state $s_i \in \sigma$ into $s_i(x)$, $s_i(y)$, and $s_i(z)$. The state $s_i(x)$ contains all the assignments to the atomic subformulas using the variable x. For each variable x, there are 3 cases to consider: (1) $x \simeq c$ appears in both $s_i(x)$ and $s_j(x)$ for some indices $i < j$ but $x \simeq c$ is not in any other state before $s_j(x)$. (2) $f^n(x) \simeq x$ appears in some state $s_j(x)$, but not in any earlier states; and (3) neither cases (1) or (2) apply and consequently each equality $x \simeq c$ appears in at most one state in σ.

In the first case, we add fresh elements $a_1, \ldots, a_{j-1}$, set $f(a_k) = a_{k+1}$ for $k < j-1$, set $f(a_{j-1}) = a_i$, and assign predicates based on the assignments to the states $s_1(x), \ldots, s_{j-1}$. We are guaranteed acceptance conditions are satisfied by the axioms (1). In the second case, we add fresh elements $a_1, \ldots, a_{i+n-1}$, and assign f and the predicates in the obvious way, with $f(a_{i+n-1}) = a_i$. We are guaranteed consistency by the axioms (2)–(6). In the final case, we add an infinite sequence of fresh elements $a_1, a_2, \ldots$.

3.3 Extensions to FFP(E)

In this section, we analyze the complexity of two extensions to FFP(E).

FFP(NL) is the fragment of FFP that admits only a single function symbol f with fixed-point expressions, but allows different bound variables to appear together in the same scope. We can reduce FFP(NL) to monadic second order logic by translating each fixed-point expression $(\mu R.\lambda x.C[R])t$ into an equivalent second-order expression $(\forall Z)\ (\forall x.Z(x) \iff C[Z](x)) \implies Z(t)$.

Both weak and strong monadic second-order logic with a single function symbol is decidable [4] (Corollary 7.2.11 and 7.2.12). So FFP(NL) logic is decidable. The second-order theory of one unary function is on the other hand not elementary recursive. It does not necessarily follow that FFP(NL) is non-elementary as well, but we establish in [3] that FFP(NL) is at least NEXPTIME-hard. Our proof is inspired by a similar construction for LRP [21], and reduces the NEXPTIME-complete problem of deciding whether a tiling problem $\mathcal{T}$ admits a tiling compatible with $\mathcal{T}$ on a square grid of size $2^n \times 2^n$.

FFP(NL) does not enjoy the finite model property. For example:

Proposition 11. *The sentence* $(\Box_{f,x}\ (\Box_{f,y}\, x \not\simeq y)\,(f(x)))\,(c)$ *is satisfiable by an infinite model, but unsatisfiable for finite models.*

In [3], we use this result to show that FFP(NL) is incomparable with the *Logic of Reachable Patterns (LRP)* [21]. LRP allows reasoning backwards along edges, and consequently can specify properties that FFP(NL) cannot. However, LRP has the finite model theory whereas by Prop. 11, FFP(NL) does not. We are not aware of any matching lower and upper bounds on the complexity of FFP(NL), neither do we know if the weak theory (that only admits finite models) of FFP(NL) is any easier than full FFP(NL).

We also consider the fragment of FFP where multiple function symbols are allowed to be associated with the temporal connectives and we are allowed to nest different functions over the variables. We call this fragment 2FFP(E). Among other things, this logic allows us to encode arbitrarily large grids. For example, we can express that functions f and g commute over all nodes reachable from a given constant c

$$(\Box_{f,x}\,[\Box_{g,y}\, f(g(y)) \simeq g(f(y))]x)\,(c)$$

We show in [3] that the satisfiability problem for this logic is undecidable. The proof uses a commonly used reduction from tiling problems. It is very similar to results in [6,21].

4 SMT Solver Integration

This section describes how Theorem 10 can be used to provide a decision procedure for $\mathsf{FFP(E)}$ together with a background theory $\mathcal{T}$. The theorem suggests a direct embedding of $\mathsf{FFP(E)}$ into LTL, but this is not always necessary, and we here examine how intermediary steps can be used. The approach we will present is analogous to how SAT solvers may be combined with decision procedures, except, instead of extracting conflict clauses for a propositional SAT solver, we extract temporal safety formulas to refine a propositional LTL abstraction.

4.1 FFP and Theories

Our formulation of $\mathsf{FFP(E)}$ uses auxiliary constants a, b, c but does not say whether there are any additional constraints on the constants. This is a convenient simplification for presenting $\mathsf{FFP(E)}$, but we would like integrating $\mathsf{FFP(E)}$ with other theories, such as the theories $\mathcal{T}$ of arrays and linear arithmetic. These theories can directly be combined in a Nelson-Oppen [15] framework of signature disjoint, stably infinite theories. In this setting $\mathsf{FFP(E)}$ can treat subterms that use symbols from other theories as constants. The theory EUF of uninterpreted theories is half-way built into $\mathsf{FFP(E)}$ because it relies on congruence closure over unary functions. SMT solvers most commonly provide general purpose congruence closure decision procedures, and these can be used directly for the unary functions used in the $\mathsf{FFP(E)}$ fragment.

We here observe that the Nelson-Oppen combination result can also be used for $\mathsf{FFP(E)}$. Formally, $\mathsf{FFP(E)}$ is not a first-order theory, yet it can still satisfy the conditions for a Nelson-Oppen combination: First, $\mathsf{FFP(E)}$ is stably infinite. In other words, if a formula φ over $\mathsf{FFP(E)}$ has a model, it has a model of any larger cardinality. This follows because the evaluation of φ only depends on values of the original auxiliary constants a, b, c, their forward closure under a fixed set of unary functions f, g, and the values of unary predicates (P, Q) over the closure. The original model will already fix their interpretation. Second, we will assume that the unary predicates and functions used for $\mathsf{FFP(E)}$ formulas are not used in other theories. This restriction is required for completeness. For example, the combination result does not apply to the formula $(\Box_{f,x}\, \mathit{read}(x) \leq \mathit{read}(f(x)))\,(a)$ which uses arithmetical relation $\leq$ and the shared function read.

To summarize, we have:

Theorem 12. *Let φ be a formula over $\mathsf{FFP(E)}$ + $\mathcal{T}$, where $\mathcal{T}$ is stably infinite decidable theory whose signature is disjoint from the unary functions and predicates used by $\mathsf{FFP(E)}$. Then satisfiability of φ is decidable.*

Proof (Sketch). (1) We first *purify* φ to create a conjunction $\varphi_{\mathsf{FFP(E)}} \wedge \varphi_{\mathcal{T}}$, such that $\varphi_{\mathsf{FFP(E)}}$ uses only the vocabulary from $\mathsf{FFP(E)}$, and $\varphi_{\mathcal{T}}$ uses only the vocabulary from $\mathcal{T}$. The two conjuncts may share constants $a, b, c, \ldots$. (2) Then apply Theorem 10 to $\varphi_{\mathsf{FFP(E)}}$ to obtain φ_{PTL*}. (3) Create the conjunction φ_{Frame} comprising of the frame conditions $\Box(a \simeq b) \vee \Box(a \not\simeq b)$ for each pair of constants a, b that occur in both conjuncts. The resulting formula $\varphi_{PTL*} \wedge \varphi_{\mathcal{T}} \wedge \varphi_{Frame}$ is equisatisfiable with φ. Furthermore, since $\mathsf{FFP(E)}$ is stably infinite we can reconstruct the Nelson-Oppen combination result and observe that it is satisfiable iff there is a propositional LTL model σ for $\varphi_{PTL*} \wedge \varphi_{Frame}$ and model $\mathcal{M}$ for $\varphi_{\mathcal{T}}$, such that $\sigma \models a \simeq b$ iff $\mathcal{M} \models a \simeq b$ for each pair of shared variables a, b.

4.2 Abstraction/Refinement Solver Combinations

The proof of Theorem 12 could suggest applying the transformations from Theorem 10 eagerly. The drawback is that a potential quadratic number of new formulas are created in the process. We will therefore develop a method that integrates with an SMT solver, such as Z3 [5], in a more incremental way. The integration is a bit similar to how state-of-the-art SAT solving techniques are used in SMT solvers: The SAT solver treats each interpreted atom in a formula as a propositional atom. It provides propositional models that assign each atom to *true* or *false*. We will use s to refer to a propositional model, and it will be represented as the set of atomic formulas that are true in the propositional model. A propositional model s of a formula φ corresponds to a conjunction of literals:

$$L := \bigwedge_{a \in \mathrm{ASF}(\varphi), a \in s} a \;\wedge \bigwedge_{a \in \mathrm{ASF}(\varphi), a \notin s} \neg a$$

The theory solvers check the propositional models for $\mathcal{T}$ consistency. If the conjunction is $\mathcal{T}$-unsatisfiable, then there is a minimal subset $L' \subseteq L$ such that $\mathcal{T} \wedge L'$ is inconsistent. The SAT solver can in exchange learn the clause $\neg L'$ and has to search for a propositional model that avoids L'.

An incremental integration of a solver for LTL with a $\mathcal{T}$ solver is a simple generalization. Instead of refining a propositional model s, we here refine a propositional temporal model σ that is generated by a propositional LTL solver.

1. *Purify*: From the original formula φ create the purified and separted conjunction $\varphi_{\mathsf{FFP(E)}} \wedge \varphi_{\mathcal{T}}$.
2. *Erase*: Create a temporal abstraction φ_{PTL} from $\varphi_{\mathsf{FFP(E)}}$. This formula does not contain the embedding axioms.
3. *Incremental Embed*: If φ_{PTL} has a temporal model σ, then check (a) that each state is consistent with $\varphi_{\mathcal{T}}$, (b) that σ evalutes rigid subformulas to

In the limit, this procedure produces $\varphi_{PTL*} \wedge \varphi_{Frame}$. On the other hand, it allows first adding partial constraints that increase the complexity of checking propositional temporal satisfiability only incrementally. It also allows interposing partial checks on σ that result in a modest cost for the LTL checking phase.

Example 13 (Neighbor consistency). *Suppose the model σ contains the sequence of states $s_0, s_1, \ldots$, and suppose that s_0 contains the state assignment $c \simeq f(c') \wedge x \simeq c' \wedge \neg P(c)$, and s_1 contains the state assignment $P(x)$. The states cannot be neighbors because the conjunction*

$$c \simeq f(c') \wedge x \simeq c' \wedge \neg P(c) \wedge \textcircled{f} P(x) \;\equiv\; c \simeq f(c') \wedge x \simeq c' \wedge \neg P(c) \wedge P(f(x))$$

is contradictory. To rule out this case, it suffices to add the safety formula

$$c \simeq f(c') \wedge x \simeq c' \wedge \neg P(c) \;\Rightarrow\; \neg \bigcirc P(x),$$

or equivalently strengthen the accessiblity relation Next*.*

Example 14 (Cross-state consistency). *The two states s_1 and s_2 are contradictory if s_1 entails the assignment $P(x) \wedge x \simeq c$ and state s_2 entails the assignment $\neg P(y) \wedge y \simeq c$ for potentially different variables x and y. Such a situation is ruled out if we apply safety condition (7) for every pair of variables x, y, and every literal $\ell(x)$, but cross-state consistency checking will also capture this case. The resulting safety condition is in this case*

$$\Box\neg(P(x) \wedge x \simeq c) \;\vee\; \Box\neg(\neg P(y) \wedge y \simeq c)$$

A set of relevant tests are in order of their overhead:

State consistency: Each state can be checked for consistency in isolation. If the state is not $\mathcal{T}$ consistent, we can constrain the LTL abstraction by adding an invariant to rule out the inconsistent state. The saturation condition E-taut is contained in the state consistency check.

Cross-state consistency: The conjunction of states can be checked for consistency. If the conjunction is inconsistent, then generate a disjunction of invariants to rule out the inconsistent combination of states. This check requires creating fresh copies of variables so that their evaluation is not fixed across states. Cross-state consistency implies state consistency, but requires checking a larger formulas.

Neighbor consistency: Each pair of consecutive states s_i, s_{i+1} can be checked for relative consistency. In contrast to cross-state consistency, we only need one copy of the variables; the occurrences of the variable x in s_{i+1} is replaced by $f(x)$. If some pair of states is found to not be consecutive, one can add an additional safety constraint. The safety constraint does not increase the set of states reachable in the temporal tableau. Neighbor and cross-state consistency can be used to enforce the frame condition φ_{Frame}.

Saturation consistency: In the spirit of approaches based on first-order encodings of fixed-point logics [8,12,16] we can saturate φ_{FFP} using a first-order approximation. In the context of $\mathsf{FFP(E)}$, we can enforce some properties of $\mathcal{U}$ by instantiating the axiom:

$$\forall x \;.\; \lceil(\psi \;\mathcal{U}_{f,x}\, \psi')\rceil(x) \;\leftrightarrow\; \psi' \;\vee\; (\psi \wedge \lceil(\psi \;\mathcal{U}_{f,x}\, \psi')\rceil(f(x))).$$

There are two ways to instantiate this axiom in a way that avoids indefinite unfoldings. The first is to instantiate it whenever there is a ground subformula in $\varphi_{\mathsf{FFP(E)}}$ congruent to $\lceil(\psi \;\mathcal{U}_{f,x}\, \psi')\rceil(f(t))$ for some t. The second is to instantiate it when both $f(t)$ and $\lceil(\psi \;\mathcal{U}_{f,x}\, \psi')\rceil(t)$ appear in $\varphi_{\mathsf{FFP(E)}}$. The Z3 solver allows controlling such instantiations using so called E-matching on patterns. Note that saturation consistency can in some cases completely replace all other checks. This is the case when there is a complete axiomatization for ground queries using quantifiers, as in [8].

Embedding consistency: Finally, the auxiliary axioms (1)-(11) required for a full embedding can be added lazily by only adding axioms that are violated by the current model.

5 Conclusions and Future Work

In this paper, we have introduced several ground first-order logics with fixed-points, and shown how satisfiability for the functional fixed-point logic with equality $\mathsf{FFP(E)}$ can be reduced to checking satisfiability of linear-time temporal formulas. Furthermore, we have developed and implemented an abstraction/refinement framework that integrates an LTL solver with an SMT solver to efficiently solve $\mathsf{FFP(E)}$ satisfiability problems directly.

Our choice of LTL as the target is a matter of convenience that was useful for identifying NP-complete subsets of $\mathsf{FFP(PL)}$ in Section 3.1. We suspect that one can extend those techniques to identify fragments of $\mathsf{FFP(E)}$ with an NP-complete decision problem. Our reduction to $\mathsf{FFP(E)}$ satisfiability checking was reduced to checking satisfiability of tableau normal forms. It is well-known that the tableau construction captures more than LTL; it also allows for handling formulas in the extended temporal logic, ETL [20]. In ETL, we can for instance express $\forall n \geq 0.P(f^{2n}(a))$. It is expressible as $(\nu X \lambda x.X(ff(x)) \wedge P(x))(a)$, but does not correspond to a formula in $\mathsf{FFP(E)}$. Nevertheless, the satisfiability of such formulas can be checked using the same apparatus developed in this paper.

While simple extensions of $\mathsf{FFP(E)}$ are undecidable, there are decidable classes of formulas that can be formulated using functional fixed-points, yet they cannot be formulated in $\mathsf{FFP(E)}$. For example [8] studies a fragment based on the predicate $\forall x : [a \xrightarrow{f} b].\varphi$ that allows multiple functions and variables to interact. Among other things, their predicate allows one to specify the formula

$$\forall x : [a \xrightarrow{f} nil].\Big(x = nil \vee \forall y : [f(x) \xrightarrow{f} nil].y \not\simeq x\Big)$$

which states that the elements in the list from a to nil are distinct. The formula refers simultaneously to multiple dereference functions. The reduction to LTL does not work when there are multiple bound variables: The LTL reduction requires that at most one variable is affected in the tableau state transitions. We are investigating whether *freeze* quantifiers, which were developed in the context of real-time temporal logic [1] can be applied. We thank the reviewers, Sergio Mera and Margus Veanes for valuable feedback.

References

1. Alur, R., Henzinger, T.A.: A really temporal logic. J. ACM 41(1), 181–204 (1994)
2. Balaban, I., Pnueli, A., Zuck, L.D.: Shape analysis of single-parent heaps. In: Cook, B., Podelski, A. (eds.) VMCAI 2007. LNCS, vol. 4349, pp. 91–105. Springer, Heidelberg (2007)
3. Bjørner, N., Hendrix, J.: Linear functional fixed-points. Technical Report MSR-TR-2009-8, Microsoft Research (2009)
4. Börger, E., Grädel, Gurevich: The Classical Decision Problem. Springer, Heidelberg (1996)
5. de Moura, L., Bjørner, N.: Z3: An Efficient SMT Solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008)
6. Immerman, N., Rabinovich, A.M., Reps, T.W., Sagiv, S., Yorsh, G.: The boundary between decidability and undecidability for transitive-closure logics. In: Marcinkowski, J., Tarlecki, A. (eds.) CSL 2004. LNCS, vol. 3210, pp. 160–174. Springer, Heidelberg (2004)
7. Lahiri, S.K., Qadeer, S.: Verifying properties of well-founded linked lists. In: Principles of Programming Languages (POPL 2006), pp. 115–126 (2006)
8. Lahiri, S.K., Qadeer, S.: Back to the future: revisiting precise program verification using SMT solvers. In: POPL, pp. 171–182. ACM, New York (2008)
9. Lev-Ami, T., Immerman, N., Reps, T.W., Sagiv, S., Srivastava, S., Yorsh, G.: Simulating reachability using first-order logic with applications to verification of linked data structures. In: Nieuwenhuis, R. (ed.) CADE 2005. LNCS, vol. 3632, pp. 99–115. Springer, Heidelberg (2005)
10. Libkin, L.: Elements of Finite Model Theory. Springer, Heidelberg (2004)
11. Manna, Z., Pnueli, A.: Temporal Verification of Reactive Systems: Safety. Springer, Heidelberg (1995)
12. McPeak, S., Necula, G.C.: Data structure specifications via local equality axioms. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 476–490. Springer, Heidelberg (2005)
13. Møller, A., Schwartzbach, M.I.: The pointer assertion logic engine. In: Programming Language Design and Implementation (PLDI 2001), pp. 221–231 (2001)
14. Nelson, G.: Verifying Reachability Invariants of Linked Structures. In: Principles of Programming Languages (POPL 1983), pp. 38–47 (1983)
15. Nelson, G., Oppen, D.C.: Simplification by cooperating decision procedures. ACM Transactions on Programming Languages and Systems 1(2), 245–257 (1979)
16. Rakamarić, Z., Bingham, J., Hu, A.J.: An inference-rule-based decision procedure for verification of heap-manipulating programs with mutable data and cyclic data structures. In: Cook, B., Podelski, A. (eds.) VMCAI 2007. LNCS, vol. 4349, pp. 106–121. Springer, Heidelberg (2007)

17. Ranise, S., Zarba, C.G.: A Theory of Singly-Linked Lists and its Extensible Decision Procedure. In: SEFM 2006, pp. 206–215 (2006)
18. Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: 17th LICS, pp. 55–74. IEEE Computer Society, Los Alamitos (2002)
19. Sistla, A.P., Clarke, E.M.: The complexity of propositional linear temporal logics. J. ACM 32(3), 733–749 (1985)
20. Wolper, P.: Specification and synthesis of communicating processes using an extended temporal logic. In: POPL, pp. 20–33 (1982)
21. Yorsh, G., Rabinovich, A.M., Sagiv, S., Meyer, A., Bouajjani, A.: A logic of reachable patterns in linked data-structures. In: Aceto, L., Ingólfsdóttir, A. (eds.) FOSSACS 2006. LNCS, vol. 3921, pp. 94–110. Springer, Heidelberg (2006)