Linear Functional Fixed-points

Nikolaj Bjørner
Microsoft Research
nbjorner@microsoft.com

Joe Hendrix
Microsoft Corporation
johendri@microsoft.com

January 30, 2009

Technical Report
MSR-TR-2009-8

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

Linear Functional Fixed-points

Nikolaj Bjørner
Microsoft Research
nbjorner@microsoft.com

Joe Hendrix
Microsoft Corporation
johendri@microsoft.com

January 30, 2009

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. Logic 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 [18], 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.

1.1 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 $\mathsf{FFP(E)}$ 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 1.1 (A simple example). We illustrate the use of reachability predicates using a simple example also used in [25]. It exercises transitivity. We use $\forall x : [a \xrightarrow{f} b].\varphi(x)$ to say that there $f^n(a) \simeq b$ 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.

1.2 Related work

The literature on verification of heap manipulating programs is quite extensive. Greg Nelson formulated a first-order axiomatization of a ternary reachability predicate in [23]. The paper proposes 8 axioms for the ternary predicate. The axioms are sufficient for a verification example, but general completeness with respect to ground validity was left as an open question. The work has inspired a number of more recent extensions and variants. Ranise and Zarba [27] identify an NP-complete fragment of acyclic singly linked lists. McPeak and Necula [21] provide a decision procedure for heap properties that do not use pointer disequalities. It is designed for *local* heap properties; these are properties that use only a bounded fragment of the heap around distinguished elements. Bingham, Rakamarić and Hu [4, 25] develop a calculus and a set of inference rules for the binary reachability predicate and a ternary predicate that expresses reachability subject to visiting an auxiliary node. Their logic is closed under taking weakest pre-conditions, and the rules are amenable to integration with SMT solvers [26], but completeness of the inference rules was left as an open problem. Lahiri and Qadeer [15] use two auxiliary predicates to obtain a similar effect for well-founded lists. Their approach is extended with a set of practical axioms and proof rules for the case where linked data-structures use pointer-arithmetic [6]. Later Lahiri and Qadeer [16] provide a complete set of axioms for a quite general theory of linked list verification. They rely on the pattern-based quantifier instantiation engines Simplify [12] and Z3 [10] for implementing their procedure as a set of inference rules and axioms. The new set of inference rules also shows better performance than the ones proposed in earlier work. Several of the above extensions simulate subsets of the theory of linked data-structures using (incomplete) first-order axioms and inference rules. They rely on support from first-order theorem proving heuristics for ensuring that their encodings also provide a decision procedure. The approach 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.

A different line of work in the context of reasoning about linked data-structures takes as starting point decidable logics that can be mapped into automata-based decision procedures. The Pointer Asserting Logic Engine, PALE [22], can reason about heap-allocated data structures using weak monadic second-order logic of graph types. The tool reduces this logic to weak monadic second-order logic over trees and uses the MONA theorem prover [14] to verify correctness properties. The logic of reachable patterns [35] is a decidable and quite expressive logic that combines local reasoning with an extended form of regular expressions. Decidability is also shown by reducing the logic to equisatisfiable formulas in monadic second-order logic over ranked trees. Both of these logics are quite expressive, but their decision procedure have the high complexity associated with monadic second-order logic. The boundary between decidable and undecidable versions of first-order logics with transitive closure is investigated in [13].

A wide body of the related work combines predicate abstraction with the verification of heap properties, This includes work around the TVLA tool in [17], which proposes a set of axioms for acyclic lists, and a method based on predicate abstraction for singly linked lists [19]. Balaban et al. [2] use a small model theorem to derive a decision procedure.

Finally, the correctness of many heap-manipulating algorithms depends on the fact that different pointers refer to distinct memory structures. Separation logic [28] extends Hoare logic with an additional conjunction operator $*$ where $A * B$ indicates that properties A and B hold in separate sections of the heap. Separation logic has been used in many different automated reasoning techniques, including inductive theorem proving (e.g. [32]) and predicate abstraction (e.g. [34]). Recent work by Sims [29] extends separation logic with fixed-point operators to express recursive properties, albeit without presenting decidability results.

1.3 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

Functional Fixed-point Logic (FFP) extends quantifier-free first-order logic with the fixed-point operators μ and ν to define least and greatest fixed-points of monadic predicates. To be more specific, we let x range over bound variables, X ranges over bound monadic 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:

$$\begin{array}{rcl} t & ::= & f(t) \mid g(t) \mid c \mid x \\ atom & ::= & t \simeq t' \mid P(t) \ \mid R \\ \varphi, \psi & ::= & X(t) \mid atom \mid \neg atom \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \\ & \mid & (\mu X \,.\lambda x \,.\ \varphi[X])(t) \mid (\nu X \,.\, \lambda x \,.\, \varphi[X])(t) \end{array}$$

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:

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

where A and B are formulas that do not contain X, but may contain x. The greatest fixed-point operator is expressible using the least fixed-point operator by using the de-Morgan dual formulation:

$$\begin{array}{rcl} \nu X.\lambda x.(B \ \vee\ [A \wedge X(f(x))]) & = & \neg\mu X.\lambda x.(\neg B \ \wedge\ [\neg A \vee X(f(x))]) \\ & = & \neg\mu X.\lambda x.((\neg B \wedge \neg A) \vee [\neg B \wedge X(f(x))]) \end{array}$$

Convention 2.1. The following shorthands will be used throughout the paper:

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

Convention 2.2. The set of subformulas of a formula φ is denoted $SF(\varphi)$. 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 refer to. We say that a formula φ is *linear* if each subformula $\psi \in SF(\varphi)$ refers to 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.

The form $(A \,\mathcal{U}_{f,x}\, B)\,(a)$ restricts somewhat, but not completely, the types of fixed-point formulas that can be created. Distributivity of conjunction and disjunction can be used to combine multiple occurrences of $X(f(x))$ into a single one, and separate out disjuncts not containing X into a formula B. The format does restrict handling fixed-point expressions with subterms of the form $X(ffx)$. This is not always an essential restriction on the properties that can be formulated using FFP. The fragment is for instance closed under the weakest precondition predicate transformer (see Proposition 3.6). Our restriction does in principle limit the expressiveness a bit more than really required for our results. We elaborate on *using extended temporal logic* in the conclusion.

2.1 Normal forms

We first discuss how formulas can be normalized to simplify later exposition. We rename bound variables so that variables occurring in different contexts have different names while giving the same name to variables in nested quantifiers when possible. Furthermore, for each top-level application of $(\varphi \,\mathcal{U}_{f,x}\, \psi)\,(t)$ we can introduce a fresh variable 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 linear fixed-point expressions.

We will from hereon refer to the variables x, y, z as *flexible variables*. Atomic formulas containing unbound flexible variables are called *flexible*; otherwise, they are called *rigid* atoms. For example $x \simeq c$ and $f\,f(x) \simeq x$ are flexible atoms, while $c \simeq f(c')$ and $P(f(c))$ are rigid atomic formulas.

It will be convenient to use a shorthand for distributing a function application f over all free occurrences of flexible variables in a formula. We therefore use the notation $\textcircled{f}\psi$ as shorthand for the formula, where every free occurrence of a flexible variable x in ψ is replaced by $f(x)$. In more detail, we can define $\textcircled{f}$ recursively on the structure of formulas and terms:

$$\begin{array}{rcl}\textcircled{f}(\psi \vee \psi') & = & \textcircled{f}\psi \vee \textcircled{f}\psi' \\ \textcircled{f}(\psi \wedge \psi') & = & \textcircled{f}\psi \wedge \textcircled{f}\psi' \\ \textcircled{f}\neg\psi & = & \neg\textcircled{f}\psi \\ \textcircled{f}\left(\psi \,\mathcal{U}_{f,x}\, \psi'\right)(t) & = & \left(\psi \,\mathcal{U}_{f,x}\, \psi'\right)(\textcircled{f}(t)) \\ \textcircled{f}P(t) & = & P(\textcircled{f}t) \\ \textcircled{f}(t \simeq t') & = & (\textcircled{f}t) \simeq (\textcircled{f}t') \\ \textcircled{f}f(t) & = & f(\textcircled{f}t) \\ \textcircled{f}x & = & f(x) \\ \textcircled{f}c & = & c\end{array}$$

2.2 Propositional Linear-time Temporal Logic

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 LTL. We will therefore recall some basic definitions of LTL. A minimalistic formulation of (*future*) LTL takes the form:

$$\varphi \quad ::= \quad P \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \bigcirc\varphi \mid (\varphi \,\mathcal{U}\, \varphi)$$

We use p to range over propositional atoms and the operator $\bigcirc$ is the next-state operator, while $\mathcal{U}$ is the temporal *until* connective. Just as in convention 2.1, other connectives can be defined using $\mathcal{U}$. For example $\Diamond\psi := \textit{true}\,\mathcal{U}\,\psi$, $\Box\psi := \neg\Diamond\neg\psi$, and $\psi \Rightarrow \psi' := \Box(\psi \to \psi')$, $(\psi \Leftrightarrow \psi') := \Box(\psi \leftrightarrow \psi')$.

FFP(PL)	PSPACE complete	Section 3.1
FFP(E)	PSPACE complete	Section 3.2
w2FFP(E)	PSPACE complete	Section 3.3
FFP(NL)	NEXPTIME hard	Section 3.4
2FFP(E)	Undecidable	Section 3.6

Table 1: FFP variants and their complexity

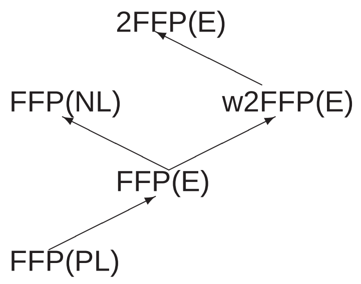

Figure 1: Relative expressiveness of the FFP fragments

Propositional models for linear-time temporal formulas consist of an infinite sequence of states $\sigma : s_0, s_1, s_2, \ldots$, such that each state s_i supplies an assignment to the propositional atoms. The satisfaction relation is extended to formulas in LTL for a model σ and position i:

$$\begin{array}{rll}
\sigma, i \models P & \text{iff} & P \in s_i \\
\sigma, i \models \psi \wedge \psi' & \text{iff} & \sigma, i \models \psi \text{ and } \sigma, i \models \psi' \\
\sigma, i \models \psi \vee \psi' & \text{iff} & \sigma, i \models \psi \text{ or } \sigma, i \models \psi' \\
\sigma, i \models \neg\psi & \text{iff} & \sigma, i \not\models \psi \\
\sigma, i \models \bigcirc\psi & \text{iff} & \sigma, i+1 \models \psi \\
\sigma, i \models \psi \,\mathcal{U}\, \psi' & \text{iff} & \text{for some } j \geq i,\ \sigma, j \models \psi',\ \text{and for all } i \leq k < j : \ \ \sigma, k \models \psi
\end{array}$$

Finally, a formula φ holds in model σ, if it holds at position 0. That is $\sigma, 0 \models \varphi$.

3 Complexity results for FFP logics

We will here introduce various variants of FFP and summarize complexity results for these. Table 1 summarizes the variants of FFP that we will be examining and associated complexity results.

Figure 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)

In this section, we 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 sub-formulas refer to an explicit *anchor*, as a constant.

Formulas in FFP(PL) have the form:

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

where x is a flexible 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 straight-forward 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 3.1. FFP(PL) *is* PSPACE *complete.*

Proof. In one direction, every propositional LTL formula can be replaced by a formula in FFP(PL) by using the transformations $\bigcirc^n P \mapsto P(f^n(x))$, and $\bigcirc^n(\varphi\,\mathcal{U}\,\psi) \mapsto (\varphi'\ \mathcal{U}_{f,x}\,\psi')\,(f^n(x))$, where φ' is the translation of φ and ψ' is the translation of ψ.

In the other direction, we reduce a ground FFP(PL) formula φ to an equisatisfiable LTL formula φ_{LTL} by eliminating the constants. This requires moving constants outside of fixed-point expressions, and decorating predicates P with a constant c determined by the context that P appears.

Our LTL formula is over predicates from two types of sources: (1) For each ground subformula $\rho(f^n(c))$ of the form $(\psi\,\mathcal{U}_{f,x}\,\psi')\,(f^n(c))$ or $P(f^n(c))$ appearing inside a fixed-point operator, we introduce a fresh rigid predicate $r_{\rho(f^n(c))}$. We call these ground subformulas *alien subformulas*, denoted $\mathrm{AlienSF}(\varphi)$. We let R denote the set of fresh rigid predicates, and let $\lceil\varphi\rceil$ denote the formula obtained by replacing ground predicates with the corresponding predicates in R. (2) For each uninterpreted predicate P and constant c, we introduce the LTL predicate P_c which is used to the context where P appears. The set of decorated predicates is denoted by $P(C)$.

To convert a formula ψ appearing beneath a fixed-point formula $(\varphi_1\,\mathcal{U}_{f,x}\,\varphi_2)\,(f^n c)$, we define the mapping LTL_c as follows.

$$\begin{aligned} \mathrm{LTL}_c(\psi \wedge \psi') &= \mathrm{LTL}_c(\psi) \wedge \mathrm{LTL}_c(\psi) \\ \mathrm{LTL}_c(\neg\psi) &= \neg\mathrm{LTL}_c(\psi) \\ \mathrm{LTL}_c((\varphi\,\mathcal{U}_{f,x}\,\psi)\,(f^n(x))) &= \bigcirc^n(\mathrm{LTL}_c(\varphi)\,\mathcal{U}\,\mathrm{LTL}_c(\psi)) \\ \mathrm{LTL}_c(P(f^n(x))) &= \bigcirc^n P_c \\ \mathrm{LTL}_c(\rho(f^n(d))) &= r_{\rho(f^n(d))} \end{aligned}$$

This function uses the rigid predicates R to replace ground predicates appearing inside the formula and decorates each uninterpreted predicate with the constant c.

To convert a ground formula φ, we define the formula $\mathrm{LTL}(\varphi)$ structurally,

$$\begin{array}{rlrl} \mathrm{LTL}(\psi \wedge \psi') &= \mathrm{LTL}(\psi) \wedge \mathrm{LTL}(\psi) & \mathrm{LTL}(\neg\psi) &= \neg\mathrm{LTL}(\psi) \\ \mathrm{LTL}((\varphi\,\mathcal{U}_{f,x}\,\psi)\,(f^n(c))) &= \bigcirc^n(\mathrm{LTL}_c(\varphi)\,\mathcal{U}\,\mathrm{LTL}_c(\psi)) & \mathrm{LTL}(P(f^n(c))) &= \bigcirc^n P_c \end{array}$$

The FFP(PL) formula φ is mapped to the LTL formula

$$\varphi_{LTL} = \mathrm{LTL}(\varphi) \wedge \bigwedge_{r \in R} \Box(r \iff \bigcirc r) \wedge \bigwedge_{\rho(f^n(c)) \in \mathrm{AlienSF}(\varphi)} r_{\rho(f^n(c))} \iff \mathrm{LTL}(\rho(f^n(c)))$$

Each FFP model $\mathcal{M}$ can be mapped to an LTL sequence $\sigma_{\mathcal{M}} : s_0, s_1, s_2, \ldots$ over rigid predicates R and flexible predicates $P(C)$ such that:

$$P_c \in s_i \iff \mathcal{M} \models P(f^i(c)) \qquad\qquad r_{\rho(f^n(c))} \in s_i \iff \mathcal{M} \models \rho(f^{i+n}(c))$$

It follows that $\mathcal{M}$ models φ iff $\sigma_{\mathcal{M}}$ models φ_{LTL} by a structural induction on subformulas of φ.

Conversely, each LTL sequence $\sigma : s_0, s_1, s_2, \ldots$ satisfying φ_{LTL} can be mapped to a FFP model $\mathcal{M}$ over the set of terms $T_{F\cup C} = \{\, f^n(c) \mid c \in C, n \in \mathbb{N} \,\}$ where $\mathcal{M} \models p(f^i(c)) \iff P_c \in s_i$. One can also show that σ models φ_{LTL} iff $\mathcal{M}_\sigma$ models φ. Consequently, φ and φ_{LTL} are equisatisfiable. $\square$

The conversions between FFP(PL) and LTL formulas are straightforward and can be done in linear time. Moreover, there are well-known fragments of linear temporal logic that have corresponding NP-complete fragments of FFP(PL). It is well-known that LTL using only the operators $\Box$ and $\Diamond$ is NP-complete. Given an FFP(PL) only containing the fixed-point operators $\Diamond_{f,x}$ and $\Box_{f,x}$, our reduction from FFP(PL) to LTL used in Theorem 3.1 will result in a LTL formula only using $\Diamond$ and $\bigcirc$

It is also the case that formulas only using $\Diamond$ in positive occurrences, and $\bigcirc$ on atomic sub-formulas is also NP-complete [30]. We can strengthen this result in a way that extends [30] in a trivial way for LTL, but it is useful for FFP(PL): if the operator $\bigcirc$ never occurs in the scope of $\Box$ or $\Diamond$, the fragment of LTL remains NP-complete. To sketch the argument: we can guess, check and combine models for the temporal subformulas using $\Box$ and $\Diamond$. An occurrence of $\bigcirc$ allows shifting an existing model by introducing an arbitrary new initial state. Formulas in FFP(PL) that do not use the distinguished function f on flexible variables fall into this fragment.

Similarly, FFP(PL) over formulas φ where $\Diamond_{f,x}$ is only used in positive occurrences and f is only applied to constants is NP-complete. This is because we can reduce φ to the following equisatisfiable LTL formula:

$$\begin{aligned}\varphi_{LTL} \quad = \quad & \mathrm{LTL}(\varphi) \wedge \bigwedge_{P(f^n(c)) \in \mathrm{AlienSF}(\varphi)} r_{P(f^n(c))} \iff \bigcirc^n P_c \\ & \wedge \bigwedge_{\left(\Diamond_{f,x}\,\psi\right)(f^n(c)) \in \mathrm{AlienSF}(\varphi)} r_{\left(\Diamond_{f,x}\,\psi\right)(f^n(c))} \implies \bigcirc^n \Diamond \mathrm{LTL}_c(\psi).\end{aligned}$$

The proof that φ and φ_{LTL} are equisatisfiable is similar to that used in Theorem 3.1.

3.2 FFP(E)

We will now consider an extension of FFP(PL) by admitting equality predicates on terms containing bound variables. The resulting logic is called FFP(E). In contrast to FFP(PL), the embedding into LTL is less straight-forward, since the equalities interact in an essential way with the models for the propositional temporal abstraction. Formulas in FFP(E) extend FFP(PL) by admitting atomic formulas that are equalities between terms containing flexible variables, constants, and a distinguished function f.

The operator Ⓕ will be convenient to limit the number possible equality predicates we need to consider. We also admit terms where the distinguished function f is applied to a constant. Thus, formulas of FFP(E) are of the form:

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

such that the formulas φ and φ' in $(\varphi \,\mathcal{U}_{f,x}\, \varphi')\,(x)$ contain at most one free flexible variable, which is x.

It is not hard to see that we can build all combinations of equality predicates using one flexible variable x, the function f and up to two constants c and c' using the 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 flexible 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 $\neg P(f(c))$. This is despite $P(f(c))$ not directly appearing in the formula.

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

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

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

3.2.1 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 3.2 (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 3.3 (Reachability Invariants [23]). Nelson introduced a ternary predicate $u \xrightarrow[w]{f} v$ which indicates that u reaches v without going through w. It can be used 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 3.4 (Well-Founded Reachability [15]). Lahiri and Qadeer define a logic for the context of lists that are well-founded with respect to a distinguished predicate. Their observation is that even data-structures, such as cyclic lists contain a distinguished element, *the head element*. So predicates that refer to the transitive closure from some list element can be cut off around such distinguished elements. the head. We say that the list is well-founded with respect to some predicate BS (blocking set), and they can define the function $B(u)$ that takes a list element u to the first element in BS that is reachable from u. Another predicate that they found to be useful is $R(u,v)$ that holds, when v is reachable from u without hitting the blocking set BS. FFP(E) allows formulating these predicates directly:

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

Example 3.5 (btwn_f [25]). Bingham, Rakamarić and Hu use a predicate that is somewhat different from Nelson's reachability predicate. Instead of *not* visiting an auxiliary node, their main reachability predicate requires that for nodes a, b, and c, there is a path from a to c following f, and that b occurs before c on such a path. The predicate has the following encoding:

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

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

Proposition 3.6. FFP(E) *is closed under pointwise functional updates.*

Proof. Let f be a function that is updated at point u to have value v. Thus, let $f' := \lambda y.\mathtt{if}\ y \simeq u\ \mathtt{then}\ v\ \mathtt{else}\ f(y)$. Then

$$\begin{array}{rl}
& (A \;\mathcal{U}_{x,f}\; B)\,(a)[f := f'] \\
= & \{\text{ unfold definition of } \mathcal{U} \ \} \\
& \mu R.\lambda x.(B \vee (A \wedge R(f(x))))(a)\,[f := f'] \\
= & \{\text{ Let } A' = A[f := f'], B' = B[f := f'] \ \} \\
& \mu R.\lambda x.(B' \vee (A' \wedge R(f'(x))))(a) \\
= & \{\text{ unfold definition of } f' \text{ under } R \ \} \\
& \mu R.\lambda x.\left(B' \vee \left(A' \wedge \left[\begin{array}{ll} & x \simeq u \wedge R(v) \\ \vee & x \not\simeq u \wedge R(f(x)) \end{array}\right]\right)\right)(a) \\
= & \{\ R(v) \text{ must visit } B' \text{ before revisiting } u \ \} \\
& \mu R.\lambda x.\left(\begin{array}{ll} & B' \\ \vee & x \simeq u \wedge ((x \not\simeq u \wedge A')\;\mathcal{U}_{x,f}\; B')\,(v) \\ \vee & x \not\simeq u \wedge A' \wedge R(f(x)) \end{array}\right)(a) \\
= & \{\text{ fold back to } \mathcal{U} \text{ format}\} \\
& \left(A'' \;\mathcal{U}_{x,f}\; B''\right)(a)
\end{array}$$

where

$$\begin{array}{ll}
A'': & x \not\simeq u \wedge A' \\
B'': & \left(\begin{array}{ll} & B' \\ \vee & x \simeq u \wedge ((x \not\simeq u \wedge A')\;\mathcal{U}_{x,f}\; B')\,(v) \end{array}\right)
\end{array}$$

□

3.2.2 Complexity results

Our main result in this section is the following:

Theorem 3.7. *The satisfiability problem of* FFP(E) *is* PSPACE*-complete.*

Proof. The theorem follows from Lemmas 3.13, 3.14, 3.15, and the fact that LTL is PSPACE-complete [30]. The lemmas are established in this section. They are combined in a sequence of transformations illustrated below:

$$\begin{tikzcd} \varphi \arrow[r, "\text{Def 3.9}"] & Tab(\varphi) \arrow[r, "\text{Def 3.11}"] & \varphi_{PTL} \arrow[r, "\text{Def 3.12}"] & \varphi_{PTL} \cup \mathsf{FFP(E)}_{PTL} \end{tikzcd}$$

The transformations ensure that

$$\varphi \text{ is satisfiable mod. } \mathsf{FFP(E)} \text{ iff } \varphi_{PTL} \cup \mathsf{FFP(E)}_{PTL} \text{ is satisfiable mod. LTL.}$$

The first transformation takes a formula φ and produces an equisatisfiable formula $Tab(\varphi)$, which replaces $\mathcal{U}$ and $\textcircled{f}$ subformulas by new propositional atoms. It also produces an acceptance condition $\mathcal{F}$, which is a set of formulas. The second transformation replaces $\Diamond_{f,x}$ and $\Box_{f,x}$ and $\textcircled{f}$ by their LTL siblings $\Diamond, \Box$, and $\bigcirc$. We have to add additional constraints to ensure that the resulting formula is equisatisfiable. This is done in the last transformation. $\Box$

Definition 3.8 (Temporal abstraction)**.** We use the function $\lceil\varphi\rceil$ to convert a temporal formula into a non-temporal formula. It replaces each sub-formula that uses a temporal connective by a new predicate. In more detail the formula $\lceil\varphi\rceil$ is defined by cases:

$$\begin{array}{rcl} \lceil\psi \wedge \psi'\rceil &=& \lceil\psi\rceil \wedge \lceil\psi'\rceil \\ \lceil\psi \vee \psi'\rceil &=& \lceil\psi\rceil \vee \lceil\psi'\rceil \\ \lceil\neg\psi\rceil &=& \neg\lceil\psi\rceil \\ \lceil\big(\psi \;\mathcal{U}_{f,x}\, \psi'\big)\,(x)\rceil &=& \lceil\big(\psi \;\mathcal{U}_{f,x}\, \psi'\big)\rceil(x) \\ \lceil\textcircled{f}\psi\rceil &=& \lceil\textcircled{f}\psi\rceil \end{array}$$

We let $P_{\text{abs}}(\varphi)$ denote the new predicates introduced for the temporal subformulas of φ. Specifically, $\mathrm{AbsP}(\varphi)$ contains a formula $\lceil\rho\rceil$ for each formula $\rho \in SF(\varphi)$ with the form $\rho = \textcircled{f}\psi$ and $\rho = \psi \;\mathcal{U}_{f,x}\, \psi'$.

Definition 3.9 (Tableau normal form $Tab(\varphi)$)**.** Given a formula φ in $\mathsf{FFP(E)}$, we define the tableau normal form of φ as the tuple $\langle\lceil\varphi\rceil, Next, Inv, \mathcal{F}\rangle$. It corresponds to the formula

$$Tab(\varphi) :\ \lceil\varphi\rceil \wedge Next \wedge Inv \wedge \bigwedge_{F\in\mathcal{F}} \Box\Diamond F \tag{1}$$

where $Next$ is the conjunction of the formulas

$$\begin{array}{rl} \lceil\textcircled{f}\psi\rceil \Leftrightarrow \textcircled{f}\lceil\psi\rceil & \text{for } \textcircled{f}\psi \in SF(\varphi) \\ \lceil\big(\psi \;\mathcal{U}_{f,x}\, \psi'\big)\rceil(x) \Leftrightarrow \big(\lceil\psi'\rceil \ \vee\ (\lceil\psi\rceil \wedge \textcircled{f}\lceil\big(\psi \;\mathcal{U}_{f,x}\, \psi'\big)\rceil(x))\big) & \text{for } (\psi \;\mathcal{U}_{f,x}\, \psi')\,(x) \in SF(\varphi) \\ R \Leftrightarrow \textcircled{f}R & \text{for each rigid predicate } R \end{array}$$

R ranges over not just the rigid atoms in φ and ground equations in φ, but also a predicate $\ell(c)$ for each flexible predicate ℓ and constant c. Additionally, note that ℓ ranges over not just the uninterpreted predicates in φ, but also the predicates $\mathrm{AbsP}(\varphi)$ introduced by temporal abstraction.

The formula Inv is the conjunction of

$$\lceil\big(\psi \;\mathcal{U}_{f,x}\, \psi'\big)\rceil(x) \ \Rightarrow\ (\lceil\psi'\rceil \ \vee\ \lceil\psi\rceil) \qquad \text{for each } (\psi \;\mathcal{U}_{f,x}\, \psi')\,(x) \in SF(\varphi) \tag{2}$$

$$\lceil\psi'\rceil \ \Rightarrow\ \lceil\big(\psi \;\mathcal{U}_{f,x}\, \psi'\big)\rceil(x) \qquad \text{for each } (\psi \;\mathcal{U}_{f,x}\, \psi')\,(x) \in SF(\varphi) \tag{3}$$

The acceptance conditions $\mathcal{F}$ are partitioned into the sets $\mathcal{F}(x), \mathcal{F}(y), \ldots$ for each of the flexible variables $x, y, \ldots$ and the sets $\mathcal{F}(x)$ comprises of the formulas

$$\lceil\big(\psi \;\mathcal{U}_{f,x}\, \psi'\big)\rceil(x) \ \rightarrow\ \lceil\psi'\rceil \qquad \text{for each } (\psi \;\mathcal{U}_{f,x}\, \psi')\,(x) \in SF(\varphi). \tag{4}$$

The tableau normal form separates the constraints on the unfoldings of f. These are captured in the relations $Next$, while the acceptance conditions are enforced using the set $\mathcal{F}$. The tableau normal form corresponds to the well-known propositional tableau construction for linear-time temporal logic [3, 20]. The expansion preserves satisfiability.

Lemma 3.10. *Let φ be a formula in* FFP(E)*, then φ is equisatisfiable with $Tab(\varphi)$.*

Definition 3.11 (Propositional linear-time temporal logic erasure φ_{PTL})**.** The purpose with definition 3.12 is to create a formula in LTL that is equisatisfiable to the corresponding FFP(E) formula. The LTL erasure is defined over the structure of formulas in FFP(E) by replacing subformulas of the form $\textcircled{f}\psi$ by $\bigcirc\psi$, by replacing subformulas of the form $(\psi \,\mathcal{U}_{f,c}\, \psi')\,(x)$ by $\psi\,\mathcal{U}\,\psi'$, $(\Box_{f,x}\,\varphi)\,(x)$ are replaced by $\Box\varphi$ and $(\Diamond_{f,x}\,\varphi)\,(x)$ is replaced by $\Diamond\varphi$. Finally, atomic subformulas are treated as different propositional atoms. In particular, the interpretation of the relation $\simeq$ is no longer defined by the theory of equality. We say that the atoms have a propositional interpretation.

Definition 3.12 (The propositional completion $\varphi_{PTL} \cup \text{FFP(E)}_{PTL}$)**.** Let us assume that φ contains the atomic subformulas $f^n(x) \simeq x$, $c \simeq f(c')$, $x \simeq c$, together with rigid atoms that we will refer to as R, and flexible literals that we will refer to as $\ell(x)$, where x is the flexible variable that is used in ℓ. Let $\mathcal{F}$ be the set of acceptance conditions from definition 3.9, the tableau normal form of φ. The propositional completion of φ is the formula $\varphi_{PTL} \cup \text{FFP(E)}_{PTL}$ obtained by adding the conjunctions:

$$x \simeq c \;\Rightarrow\; \bigcirc\,(x \not\simeq c \;\mathcal{W}\; F) \qquad F \in \mathcal{F}(x) \tag{5}$$
$$x \simeq c \wedge \ell(x) \;\Rightarrow\; \ell(c) \tag{6}$$
$$f^n(x) \simeq x \Rightarrow (\ell(x) \leftrightarrow \bigcirc^n \ell(x)) \tag{7}$$
$$f^n(x) \simeq x \wedge f^m(x) \simeq x \;\Rightarrow\; f^{\gcd(m,n)}(x) \simeq x \tag{8}$$
$$f^n(x) \not\simeq x \wedge x \simeq c \;\Rightarrow\; \bigcirc^n(x \not\simeq c) \tag{9}$$
$$f^n(x) \not\simeq x \wedge f^m(x) \simeq x \;\Rightarrow\; f^{n-m}(x) \not\simeq x \qquad n > m \tag{10}$$
$$f^n(x) \simeq x \Rightarrow \bigcirc(f^n(x) \simeq x) \tag{11}$$
$$c \simeq f(c') \wedge x \simeq c' \;\Rightarrow\; \bigcirc(x \simeq c) \tag{12}$$
$$x \simeq c \;\Rightarrow\; (x \simeq c' \;\leftrightarrow\; c \simeq c') \tag{13}$$
$$\Box E - taut \tag{14}$$

The last set of invariants (14) are the set of tautologies that can be formed using the rigid predicates. Since each invariant uses at most one pair of atomic subformulas in φ, we have

Lemma 3.13 (Size)**.** *The size of the resulting formula $\varphi_{PTL} \cup \text{FFP(E)}_{PTL}$, $|\varphi_{PTL} \cup \text{FFP(E)}_{PTL}|$, is in the order of $O(|\varphi|^2)$.*

The additional conjunctions added as a result of the propositional embedding maintain satisfiability. That is:

Lemma 3.14 (Soundness)**.** *If φ is satisfiable in* FFP(E)*, then $\varphi_{PTL} \cup \text{FFP(E)}_{PTL}$ is satisfiable in the empty theory.*

In fact, the reduction is not only equisatisfiable, but we have the converse:

Lemma 3.15 (Propositional Completeness)**.** *If the propositional completion of the formula $\varphi_{PTL} \cup \text{FFP(E)}_{PTL}$ is satisfiable in the empty theory, then φ is satisfiable in the theory of* FFP(E)*.*

Proof. If the abstraction $\varphi_{PTL} \cup \mathsf{FFP(E)}_{PTL}$ is satisfiable, then there is a sequence of states $\sigma : s_0, s_1, s_2, s_3, \ldots$, satisfying $\varphi_{PTL} \cup \mathsf{FFP(E)}_{PTL}$, where each state s_i assigns truth values to the propositional atoms in $\varphi_{PTL} \cup \mathsf{FFP(E)}_{PTL}$. We will examine the assignment to the atoms in each state from σ and extract a model $\mathcal{M}$ for the original formula φ. We will then establish that the propositional model is also a model when equalities are interpreted.

The model for φ is built by processing each flexible variable x and creating an interpretation for f that is relevant for x. Notice that the truth assignments implied by each of the states s_i can be partitioned corresponding to each of the flexible variables. Let us assume that the flexible variables are x, y, z, then the states are partitioned into $s_i(x), s_i(y)$, and $s_i(z)$ for each state $s_i \in \sigma$. The state $s_i(x)$ contains all the assignments to the atomic subformulas using the flexible variable x.

Consider the cases:

1. There are two states $s_i(x)$ and $s_j(x)$, that contain the atom $x \simeq c$. Without loss of generality, we can assume that $s_i(x)$ is the first state that contains the equality $x \simeq c$, and that $s_j(x)$ is the next state after $s_i(x)$ that contains the same equality. Furthermore, we can also assume that there are no repeated equalities between $s_i(x)$ and $s_j(x)$. Invariant (5) entails that all acceptance conditions related to the flexible variable x are satisfied between $s_i(x)$ and $s_j(x)$. Also, every state between $s_i(x)$ and $s_j(x)$ either entails no equality atoms, or if it entails an equality atom $x \simeq c'$, there is no other state between $s_i(x)$ and $s_j(x)$ that entails the same atom. We can fix the interpretation of f from c by introducing fresh distinct elements $a_0, a_1, \ldots, a_i, a_{i+1}, \ldots, a_{j-1}$, such that $\mathcal{M}(c) = a_i$, as well as $\mathcal{M}(f(a_k)) = a_{k+1}$ for $k < j$, and $\mathcal{M}(f(a_{j-1})) = a_i$. Furthermore, if some state s_k between s_0 and s_j contains an equality $x \simeq c'$, then set $\mathcal{M}(c') = a_k$. The interpretation $\mathcal{M}$ is extended to satisfy the propositional interpretation by examining the following cases for the interpretation of the atomic subformulas in a state s_k:

   - $x \simeq c$ - by construction, $\mathcal{M}(c) = a_k$.
   - $x \not\simeq c$ - by construction, each state is consistent with respect to the theory of equality, and the interpretation respects disequalities.
   - $\pm P(x)$ - whenever $s_k(x)$ entails $P(x)$, then extend the interpretation by updating $\mathcal{M}(P) := \mathcal{M}(P) \cup \{a_k\}$ if $P(x) \in s_k(x)$.
   - $f^n(x) \simeq x$ - invariant (7) implies that $x \simeq c$ is a member of every n states. In other words, $j - i$ divides n. Thus, the constructed interpretation satisfies $\mathcal{M}, a_k \models f^n(x) \simeq x$ for every element a_k.
   - $f^n(x) \not\simeq x$ - invariant (10) implies that the period length $j - i$ does not divide n. Thus, $\mathcal{M}, a_k \models f^n(x) \not\simeq x$ if $(f^n(x) \not\simeq x) \in s_k(x)$.

2. There are no repeated states containing $x \simeq c$, for any c, but there is a state that contains $f^n(x) \simeq x$ for some n. Invariant (11) implies that all states after the first occurrence of $f^n(x) \simeq x$ contain this same equality. From invariant (8), we can assume that n divides every other m, such that $f^m(x) \simeq x$ is in the suffix. Invariant (7) implies that every n states satisfies precisely the same atomic formulas. So the acceptance conditions $\mathcal{F}(x)$ are satisfied within the loop of length at most n. Let us build an interpretation for f by considering the prefix of states $s_0, \ldots, s_{j-n-1}$ leading up to the looping suffix, followed by the states $s_{j-n}, \ldots, s_j$ used in the looping suffix. We introduce the fresh elements $a_0, \ldots, a_j$ and constrain $\mathcal{M}(c)$ to be a_k, if the state $s_k(x)$ contains the equality $x \simeq c$. By our assumptions, this can only be the case if s_k is among the states $s_0, \ldots, s_{j-n-1}$ (in other words $0 \leq k < j - n$). For the remaining cases we have:

   - $x \not\simeq c$ - by construction.

- $\pm P(x)$ - by construction: $\mathcal{M}(P) := \mathcal{M}(P) \cup \{a_k\}$ if $P(x) \in s_k(x)$.
- $f^m(x) \simeq x$ - by construction.
- $f^m(x) \not\simeq x$ from invariant (10) it follows that we can assume $m < n$, but $a_k \neq a_{k+m}$ is implied by the construction.

3. Neither case 1 or 2 apply, so every equality $x \simeq c$ occurs in at most one state in $\sigma(x)$, and there is no state containing $f^n(x) \simeq x$ for any n. We build an interpretation for f by selecting an infinite sequence $a_0, a_1, \ldots, a_i, \ldots$ of fresh distinct elements and assigning $\mathcal{M}(c) = a_i$ if state $s_i(x)$ contains the atom $x \simeq c$ (there is at most one such state). As before, we extend $\mathcal{M}$ to satisfy predicates by assigning $\mathcal{M}(P) := \mathcal{M}(P) \cup \{a_k\}$ if $P(x) \in s_k(x)$. By the assumptions, the suffix contains no state implying $f^n(x) \simeq x$ for any of the atomic predicates in φ. Finally, the construction ensures that every state implies $f^n(x) \not\simeq x$ for arbitrary n.

   Notice that when the propositional model σ contains a periodic suffix, we do not need an infinite number of fresh distinct elements in the construction. It just suffices to select a period of length greater than n for any subformula of the form $f^n(x) \simeq x$. This ensures that the model satisfies $f^n(x) \not\simeq x$ in every state.

A structural induction over the formulas implies that the partial interpretation built so far also satisfies the non-atomic (temporal) formulas in the states $s_i(x)$. More specifically, invariant (13) implies that the interpretation of x is consistent with congruences over f.

Suppose now that we have fixed the interpretation of f for the flexible variable x and wish to process y. There are two cases to consider:

1. There is a state $s_i(y)$ that contains an equality $y \simeq c$, but there was a state $s_j(x)$ previously used to construct $\mathcal{M}$ that contained $x \simeq c$. There are two sub-cases:

   (a) The state $s_i(y)$ does not contain equalities of the form $f^n(y) \simeq y$ and $s_i(y)$ is the first state to contain a previously visited equality. We build a model for f based on the states $s_0(y), \ldots, s_{i-1}(y)$ by introducing fresh elements $a_1, \ldots, a_{i-1}$ into $\mathcal{M}$ and fixing f as before ($f(a_0) = a_1, f(a_1) = a_2, \ldots, f(a_{i-1}) = \mathcal{M}(c)$).

   (b) The state $s_i(y)$ does contain an equality of the form $f^n(y) \simeq y$. In this case we look for the first state among $s_0(y), s_1(y), \ldots, s_i(y)$ that contains the equality $f^n(y) \simeq y$. This first state gets aligned with the matching state for x.

   The construction relies on invariant (6). It implies that if y ever enters a state that satisfies an equality $y \simeq c$ previously satisfied by x, then the interpretation for f on y henceforth can be determined by the interpretation of f on c.

2. There is no state $s(y)$ that implies a previously encountered equality. In this case we build a model just as we did for x.

□

3.3 w2FFP(E)

w2FFP(E) is the fragment of FFP, that requires every bound variable to appear linearly, just as for FFP(E), but allows different flexible variables to use different functions. The same flexible variable is still required to use just the same function symbol. We therefore call the fragment the *weak* 2-function extension of FFP(E). Thus,

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

is a legal formula in w2FFP(E), but

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

is not, because both g and f are used on the same flexible variable y. For simplicity we will assume that formulas in w2FFP(E) use just two functions f and g in the fixed-points. The generalization to multiple functions is straight-forward.

Theorem 3.16. *The satisfiability problem of* w2FFP(E) *is* PSPACE*-complete.*

Proof. Let us examine where we relied on the use of single function symbol in the proof of lemma 3.15. All auxiliary safety conditions reference a single flexible variable. So the accessibility relation $\textcircled{f}$ was unique determined by the flexible variable. The safety condition (6) bridges the interpretation between flexible variables. Their accessibility relations could in w2FFP(E) potentially be associated with two different functions. For example, x could be associated with accessibility function f, and y could be associated with g. If x is associated with f, then the atomic literal $\ell(x)$ is also associated with f. Let us for every constant c and function f introduce the flexible variable $x_{f,c}$, and for every literal of the form $\ell(x)$ and constant c introduce a fresh predicate constant $R_{\ell c}$. Then we can replace safety condition (6) by the satisfiability preserving constraints:

$$x \simeq c \wedge \ell(x) \;\Rightarrow\; R_{\ell c} \tag{15}$$

$$x_{f,c} \simeq c \wedge (R_{\ell c} \;\rightarrow\; \ell(x_{f,c})) \tag{16}$$

We say that the flexible variable $x_{f,c}$ *owns* f at the constant c.

Given a propositional model for $\varphi_{PTL} \cup \text{FFP(E)}_{PTL}$ we can now extract a model for the functions f and g by examining first f and then g. For the function f, we build an interpretation for f by examining each variable $x_{f,c}$.

□

3.4 FFP(NL)

3.5 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 second-order monadic logic with a single function symbol is decidable [5] (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 that FFP(NL) is at least NEXPTIME-hard.

Theorem 3.17. *The satisfiability problem of* FFP(NL) *is* NEXPTIME*-hard.*

Proof. Our proof is inspired by a similar construction for LRP [35].

Given a tiling problem $\mathcal{T} = (T, R, D)$ with $T = \{T_1, \ldots, T_k\}$ and a natural number $n \in \mathbb{N}$, it is an NEXPTIME-complete problem to decide whether there is a tiling compatible with $\mathcal{T}$ on a square grid of size $2^n \times 2^n$.

We can reduce the bounded tiling problem to the satisfiability problem of an FFP problem of size $O(n^2)$. The FFP signature Σ for our problem has constants s and t and unary predicates $T \cup X \cup Y$

where $X = \{X_0, \ldots, X_{n-1}\}$ and $Y = \{Y_0, \ldots, Y_{n-1}\}$. Our intention is that every square tiling grid can be mapped to a Σ-model. The predicates X and Y are used to encode the horizontal and vertical coordinates of a square in the grid, and the predicates T are used to encode the tile associated to the grid. The constant s denotes the top-left origin and the constant t denotes the bottom-right corner. The formula φ contains the following constraints:

$$\begin{array}{rcl}
\varphi & = & Xs \simeq 0 \wedge Ys \simeq 0 \wedge Xt \simeq m \wedge Yt \simeq m \wedge ft \simeq t \\
& \wedge & \widetilde{\forall} x : [s \xrightarrow{f} t].Xfx \simeq Xx + 1 \wedge Yfx \simeq \mathrm{ite}(Xx \simeq m, Yx + 1, Yx) \\
& \wedge & \widetilde{\forall} x : [s \xrightarrow{f} t]. \left(\bigvee_{1 \le i \le k} \left(\bigwedge_{1 \le j < i} \neg T_j x \wedge T_i x \wedge \bigwedge_{i < j \le k} \neg T_j x \right) \right) \\
& \wedge & \widetilde{\forall} x : [s \xrightarrow{f} t].Xx \simeq m \vee \bigvee_{(i,j) \in R} T_i x \wedge T_j fx \\
& \wedge & \left(\left(\left(\Diamond_y \, Xy \simeq Xx \wedge Yy \simeq Yx + 1 \wedge \bigvee_{(i,j) \in D} T_i x \wedge T_j \right) (y) \right) \mathcal{U}_{f,x} \, Yx \simeq m \right) (s) \\
& \wedge & T_0(s) \wedge T_k(t)
\end{array}$$

□

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

Proposition 3.18. *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.*

We will use this result to establish that FFP(NL) is incomparable the *Logic of Reachable Patterns (LRP)* [35]. LRP allows specifying properties that require traveling both forwards and backwards along an edge whereas our logic only allows reasoning forwards. So in our logic one can always extend models with additional nodes that can reach other elements in our model, while this is not true for LRP. For example, the LRP sentence $\neg c[\xleftarrow{f}]\bot$ implies that no node can reach c via f. This sentence is not expressible in FFP. On the other hand, LRP has a finite model property whereas by Prop. 3.18, our logic does not. Sentences in FFP such as $(\Box_{f,x} \; (\Box_{f,y} \, x \not\simeq y) \, (f(x))) \, (c)$ which are only satisfiable by infinite models are not expressible in LRP. These observations imply the following:

Corollary 3.19. *The expressiveness of* FFP *and LRP is incomparable.*

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).

3.6 2FFP(E)

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 flexible 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 that the satisfiability problem for this logic is undecidable.

$$
\begin{array}{ll}
 & \{\ r \text{ and } d \text{ commute. }\} \\
 & \widetilde{\forall} x : [s \xrightarrow{r} \mathit{null}].\widetilde{\forall} y : [x \xrightarrow{d} \mathit{null}].rdy \simeq dry \\
\wedge & \{\ \text{right-most edge is straight. }\} \\
 & \widetilde{\forall} x : [s \xrightarrow{r} \mathit{null}].\widetilde{\forall} y : [x \xrightarrow{d} \mathit{null}].ry \simeq \mathit{null} \Leftrightarrow rdy \simeq \mathit{null} \\
\wedge & \{\ \text{bottom edge is straight. }\} \\
 & \widetilde{\forall} x : [s \xrightarrow{r} \mathit{null}].\widetilde{\forall} y : [x \xrightarrow{d} \mathit{null}].dy \simeq \mathit{null} \Leftrightarrow dry \simeq \mathit{null} \\
\wedge & \{\ \text{null loops on self. }\} \\
 & r(\mathit{null}) \simeq \mathit{null} \wedge d(\mathit{null}) \simeq \mathit{null} \\
\wedge & \{\ \text{each node has one type. }\} \\
 & \widetilde{\forall} x : [s \xrightarrow{r} \mathit{null}].\widetilde{\forall} y : [x \xrightarrow{d} \mathit{null}]. \bigvee_{i\in[1,k]} T_i(y) \wedge \bigwedge_{j\in[1,k]\setminus\{i\}} \neg T_j(y) \\
\wedge & \{\ \text{types are right compatible. }\} \\
 & \left(\left(\widetilde{\forall} y : [x \xrightarrow{d} \mathit{null}]. \bigvee_{(i,j)\in R} T_i(y) \wedge T_j(ry)\right) \ \mathcal{U}_{r,x}\, rx \simeq \mathit{null}\right)(s) \\
\wedge & \{\ \text{Types are down compatible. }\} \\
 & \widetilde{\forall} x : [s \xrightarrow{r} \mathit{null}]. \left(\bigvee_{(i,j)\in D} T_i(y) \wedge T_j(dy)\ \mathcal{U}_{d,y}\, dy \simeq \mathit{null}\right)(x) \\
\wedge & \{\ \text{Top-left node has type } T_0.\ \} \\
 & T_0(s) \\
\wedge & \{\ \text{Bottom right node has type } T_k.\ \} \\
 & (\Diamond_{r,x}\ (\Diamond_{d,y}\, y \not\simeq \mathit{null} \wedge ry \simeq dy \simeq \mathit{null} \wedge T_k(y))\,(x))\,(s)
\end{array}
$$

Figure 2: Tiling problem encoding

Theorem 3.20. *The satisfiability of* 2FFP(E) *is undecidable.*

Proof. We will create a tiling problem with functions d (down) and r (right), and use two points s (start) and *null*. Figure 2 shows the encoding.

□

4 Integration with the SMT solver Z3

This section describes a decision procedure for FFP(E). It uses an integration of a LTL checker and a solver for a background theory $\mathcal{T}$. The structure of the integration is similar to how SAT solvers may be combined with decision procedures. A reference implementation of the integration is available from `http://research.microsoft.com/en-us/people/nbjorner/ffpsrc.zip`.

4.1 FFP and theories

Our formulation of FFP uses auxiliary constants a, b, c, c' but does not say whether there are any additional constraints on the constants. The constants are used instead of adding to the signature of FFP alien, composite, terms from other theories than FFP. Other theories that could be of interest in the context of program analysis and verification are for instance the theory of arithmetic, term algebras, bit-vectors and arrays. We use $\mathcal{T}$ to refer to "other theories" than FFP.

The Nelson-Oppen combination result [24] applies to FFP, because FFP is stably infinite. It allows for us to use this abstraction because the only information that FFP requires from the interface terms is that there is a $\mathcal{T}$-consistent partition that is also consistent with the FFP portion of the formula we wish to check. The *interface* terms here comprises of all alien terms and subterms of the form $f(t)$, where t is a interface term.

Proposition 4.1. *Let $\mathcal{T}$ be a stably infinite theory. Let φ be a formula, with the set of rigid, interface terms $\mathit{Terms} = t_1, \ldots, t_n$ over $\mathcal{T}$, and set $R_1, \ldots, R_m$ of rigid alien predicates. Then φ is satisfiable over FFP $+ \mathcal{T}$ if and only if there is a partition $\mathit{Partition}(\mathit{Terms})$ of the set of terms Terms and an assignment α_i of the predicates R_i to* true *or* false*, such that*

$$\varphi[\alpha_i/R_i] \wedge \mathit{Partition}(\mathit{Terms}) \wedge \bigwedge_i (\alpha_i \leftrightarrow R_i)$$

is consistent over FFP $+ \mathcal{T}$ *if and only if*

$$\mathit{Partition}(\mathit{Terms}) \wedge \bigwedge_i (\alpha_i \leftrightarrow R_i) \textit{ is consistent with } \mathcal{T}$$

and

$$\varphi[\alpha_i/R_i] \wedge \mathit{Partition}(\mathit{Terms}) \textit{ is consistent with } \mathsf{FFP(E)}.$$

Note that we require that the alien terms and predicates be rigid. It is for example not allowed to nest alien function symbols over flexible variables. On the other hand, it is allowed to nest the functions used by FFP within rigid terms, since terms can be purified by introducing extra constants and equalities. For example, $\varphi[f(c) + 3]$ is equisatisfiable with $c' \simeq f(c) \wedge \varphi[c' + 3]$, where c' is a fresh constant symbol.

4.2 Abstraction/refinement solver combinations

Most modern SMT solvers, including Z3 [10], integrate theory solvers with a propositional SAT solver, based on state-of-the-art techniques for SAT solving. The integration of theory solvers and the SAT solver can be described using a simple exchange:

The SAT solver treats each atom in a formula as propositional atoms. 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 a 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 (not necessarily unique) subset $L' \subseteq L$ such that $\mathcal{T} \wedge L'$ is inconsistent and for every subset $L'' \subset L'$, it is the case that $\mathcal{T} \wedge L''$ is consistent. The SAT solver can in exchange learn the clause $\neg L'$ and has to search for a propositional model that avoids L'. The basic integration can be described using the figure below:

$$\begin{tikzcd}\boxed{\text{SAT}} \arrow[r, shift left, "s"] & \boxed{\mathcal{T}} \arrow[l, shift left, "\neg L'"]\end{tikzcd}$$

We will here describe an analogous abstraction/refinement based on LTL. Instead of adding propositional clauses, we rely on adding propositional temporal safety properties. Thus, given a formula φ, we create an initial propositional abstraction. When a LTL satisfiability checker returns a temporal model σ for φ it is checked for $\mathcal{T}$-consistency (we assume this includes consistency with respect to the theory of ground equations), as well as consistency with respect to the safety properties from definition 3.12. If the propositional model σ violates any of the $\mathcal{T}$+safety property checks, the integration produces a temporal safety property that is conjoined to the initial propositional abstraction. Conversely, the BDD-based satisfiability checker we use for LTL is also able of characterizing the set of feasible initial states using a predicate Init. The combination of these assignments can be used to constrain future checks for $\mathcal{T}$ consistency. The combination approach is illustrated below:

$$\begin{tikzcd}\boxed{\text{LTL}} \arrow[r, shift left, "\mathit{Init}, \sigma"] & \boxed{\mathcal{T}} \arrow[l, shift left, "\text{Safety blockers}"]\end{tikzcd}$$

4.3 Checking Satisfiability of LTL formulas

There are to date a variety of methods for checking satisfiability of LTL formulas. Some are based on building ω-automata, or alternating ω-automata [31]. Any method can in principle be used for our approach; the important feature is that the satisfiability checker produces a propositional temporal model σ. We will here describe a method based on a more traditional construction that is based on creating tableaux. It is simple to implement using a symbolic BDD package.

Our approach is to build a μ-calculus formula that can be evaluated using symbolic model-checking techniques. The technique is analogous to reducing LTL checking to fair CTL model checking [7]. The starting point for the construction is a temporal tableau $\mathit{Tab}(\varphi)$. For this purpose associate propositional variables $\vec{u}$ with each of the atomic formulas used in $\mathit{Tab}(\varphi)$. This includes subformulas of the form $\lceil(\psi \ \mathcal{U}_{f,x}\ \psi')\rceil(x)$ and $\lceil \textcircled{f}\psi \rceil$. If u is an atomic subformula, we associate u' with the subformula $\textcircled{f}u$.

Thus, the temporal tableau induces relations $\lceil\varphi\rceil(\vec{u})$, $\mathit{Inv}(\vec{u})$, $\mathit{Next}(\vec{u}, \vec{u}')$, and the set of relations $F(\vec{u}) \in \mathcal{F}$.

Let us introduce the following shorthands:

$$\begin{aligned}
\langle \mathit{pre} \rangle P &:= \lambda\vec{u}.\exists\vec{u}' \,.\, \mathit{Inv}(\vec{u}) \wedge \mathit{Inv}(\vec{u}') \wedge \mathit{Next}(\vec{u}, \vec{u}') \wedge P(\vec{u}'),\\
\langle \mathit{post} \rangle P &:= \lambda\vec{u}.\exists\vec{u}_0 \,.\, \mathit{Inv}(\vec{u}_0) \wedge \mathit{Inv}(\vec{u}) \wedge P(\vec{u}_0) \wedge \mathit{Next}(\vec{u}_0, \vec{u}),\\
\langle \mathit{pre}^* \rangle P &:= \mu X \,.\, \lambda\vec{u} \,.\, P(\vec{u}) \vee \langle \mathit{pre} \rangle X(\vec{u}),\\
\langle \mathit{post}^* \rangle P &:= \mu X \,.\, \lambda\vec{u} \,.\, P(\vec{u}) \vee \langle \mathit{post} \rangle X(\vec{u}).
\end{aligned}$$

The set of initial states that contain an accepting path can then be defined as Init, where:

$$\mathit{Rec} := \nu R \,.\, \langle \mathit{post} \rangle (\lambda\vec{u} \,.\, \bigwedge_{F \in \mathcal{F}} (F(\vec{u}) \wedge \langle \mathit{post}^* \rangle (R)(\vec{u}))) \tag{17}$$

$$\mathit{Init} := \lceil\varphi\rceil(\vec{u}) \wedge \langle \mathit{pre}^* \rangle \mathit{Rec}(\vec{u}) \tag{18}$$

We can build the relation for Init by evaluating the propositional μ-calculus fixed-point expressions using a BDD package. The resulting relation for Init summarizes the set of propositional evaluations that admit accepting models. So if $Init$ is empty, the propositional formula is unsatisfiable; otherwise, one can extract a model σ consisting of a *prefix* $s_0, s_1, \ldots, s_i$, and a periodic *suffix* $s_{i+1}, \ldots, s_j$ that is repeated. In other words, σ is of the form $s_0, s_1, \ldots, s_i, (s_{i+1}, \ldots, s_j)^\omega$. Each state in the propositional model evaluates the atomic sub-formulas in the original formula φ to either *true* or *false*.

We will now describe the steps taken for checking and refine a propositional model σ.

4.4 Refining the LTL abstraction

As described above, models produced by the LTL abstraction are refined using checks for $\mathcal{T}$ consistency, as well as the auxiliary safety constraints from definition 3.12. The refinement steps are described in the following.

4.4.1 State consistency

Every state s in a propositional model σ assigns the atomic formulas in φ to either *true* or *false*. We can check whether the assignment is $\mathcal{T}$-consistent for arbitrary stably infinite theories $\mathcal{T}$. So given a state s, let L' be a minimal $\mathcal{T}$-inconsistent subset of the literals associated with s. We add the invariant $\Box\neg L'$ to φ and re-check the formula for satisfiability modulo propositional linear-time temporal logic.

Checking state consistency in the theory of equality allows for adding some of the auxiliary invariants from definition 3.12 as a side-effect. In particular, the invariant (13) is checked and added as a consequence of checking state consistency.

We should notice that the invariants that are produced need not correspond to a well-formed formula in FFP(E).

Example 4.2 (State consistency). Suppose a state s contains the following assignment:

$$P(x) \ \wedge \ x \simeq c \ \wedge \ y \simeq c \ \wedge \ \neg P(y)$$

The theory of equality is required in order to detect the contradiction. So the $\mathcal{T}$ solver is expected to produce the invariant $\Box\neg(P(x) \ \wedge \ x \simeq c \ \wedge \ y \simeq c \ \wedge \ \neg P(y))$. Notice that it does not correspond to a formula form FFP(E) because it uses two flexible variables x and y.

4.4.2 Cross state consistency

Cross-state consistency generalizes state consistency. Instead of checking consistency of a single state s, we check the joint consistency of the states $s_0, s_1, \ldots, s_j$ in the propositional model σ. This is done by checking the consistency of the literals $L_0(\vec{x}_0) \wedge L_1(\vec{x}_1) \wedge \ldots \wedge L_j(\vec{x}_j)$, where the set of literals associated with each state is instantiated by a set of different flexible variables ($\vec{x}_0, \vec{x}_1, \ldots, \vec{x}_j$). If a cross-state constraint is $\mathcal{T}$-unsatisfiable, then add the safety condition:

$$\Box\neg L_1 \vee \ldots \vee \Box\neg L_k \qquad \text{For } \mathcal{T}\text{-unsatisfiable states } L_1, \ldots, L_k$$

Cross-state consistency allows blocking states that are not mutually consistent.

Example 4.3 (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 flexible variables x and y. Such a situation is ruled out if we apply safety condition (6) for every pair of flexible 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

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

4.4.3 Neighbor consistency

Example 4.4 (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

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

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

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

The safety condition (11) uses only one atom. It can therefore be compiled directly into the $Next$ relation. On the other hand, to maintain the safety condition (12) $c \simeq f(c') \wedge x \simeq c' \;\Rightarrow\; \bigcirc(x \simeq c)$, we may potentially need to introduce the new atom $x \simeq c$. The number of such atoms can be quadratic in the the number of constants and variables. We therefore defer imposing this safety condition, and instead check propositional models σ for neighbor consistency. This is achieved by checking each pair of neighboring states $\langle s_k, s_{k+1} \rangle$, for $k = 0, \ldots, j-1$, and $\langle s_j, s_i \rangle$ for consistency by checking $L_1 \wedge \textcircled{\scriptsize f} L_2$.

$$\Box \neg (L_1 \wedge \bigcirc L_2) \qquad \text{For } \mathcal{T}\text{-unsatisfiable successors } L_1, L_2$$

4.4.4 Interface terms

Definition 3.12 requires potentially producing equality literals corresponding to all pairs of interface terms, to ensure that the safety conditions are enforced. We apply a model-based approach for introducing such equality literals [9]: an equality $t \simeq t'$ between two interface terms is added only if the states in σ are cross-state consistent with a model that evaluates t to the same value as t'.

4.4.5 Embedding consistency

Example 4.5 (Embedding consistency). Suppose we have the formula

$$(\Diamond_{f,x}\, ffx \simeq x \wedge P(x))\,(x) \wedge (\Diamond_{f,x}\, (\Box_{f,x} \neg P(x))\,(x))\,(x) \wedge x \simeq a.$$

It says that from a, there is a sequence of f applications that reach $P(x)$ and $ffx \simeq x$, but also eventually $\neg P(x)$ holds for every sequence of f applications. The (uncluttered) LTL version:

$$\varphi_{PTL} : \Diamond(ffx \simeq x \wedge P(x)) \wedge (\Diamond \Box \neg P(x)) \wedge x \simeq a$$

is satisfiable if $\simeq$ is left un-interpreted. But after adding an instance of axiom 7 and 11, we obtain the prepositionally unsatisfiable formula:

$$\varphi_{PTL} \;\wedge\; (ffx \simeq x \Rightarrow (P(x) \leftrightarrow \bigcirc\bigcirc P(x)) \;\wedge\; (ffx \simeq x \Rightarrow \bigcirc ffx \simeq x)$$

The additional safety conditions from definition 3.12 are checked by a custom solver for FFP(E). The solver checks that each of the invariants holds for the propositional path σ.

The condition (5) requires one of the more interesting checks. Recall, that we assume σ is of the form $s_0, s_1, \ldots, s_i, (s_{i+1}, \ldots, s_j)^\omega$. The path $s_0, s_1, \ldots, s_i, \ldots, s_j, s_i, \ldots, s_j$ is checked for an occurrence of the first repeated equality $x \simeq c$. The test succeeds if there is no state containing an equality of the form $x \simeq c$, otherwise, it suffices checking the state sequence corresponding to the first repeated equality. Each of the acceptance conditions $F \in \mathcal{F}(x)$ is checked with the sub-sequence. If some condition F does not evaluate to *true* in the sub-sequence, we add the conditions corresponding to (5).

4.5 Constraining the FFP(E) abstraction

A formula is obviously unsatisfiable modulo FFP(E)+$\mathcal{T}$ if $\lceil\varphi\rceil$ is already unsatisfiable modulo $\mathcal{T}$. Any partial axiomatization for FFP(E) can be used for checking $\lceil\varphi\rceil$. So in principle we can add any set of ad-hoc axioms used in [16, 21, 25] to further constrain propositional models. By using Z3's trigger-based quantifier instantiation mechanism, these axioms can be instantiated on demand based on the current ground subformulas. To see how how this facility can be used in the context of FFP(E), consider the following unfolding axiom for $\mathcal{U}$:

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

If we instantiate the quantifier whenever there is a ground sub-formula of the form $\lceil(\psi\ \mathcal{U}_{f,x}\,\psi')\rceil(t)$, it will produce another ground subterm of the form $\lceil(\psi\ \mathcal{U}_{f,x}\,\psi')\rceil(f(t))$, triggering an indefinite set of instantiations. Z3 allows controlling instantiations based on *patterns*, known from the Simplify theorem prover [12]. Z3 uses efficient term indexing techniques for implementing E-matching based quantifier instantiation [8]. Universally quantified axioms are instantiated only when the current state of the search contains one or more ground terms matching a set of patterns that use the bound variables. The pattern $\{\lceil(\psi\ \mathcal{U}_{f,x}\,\psi')\rceil(f(x))\}$ allows instantiating the quantifier if there is a ground term of the form $\lceil(\psi\ \mathcal{U}_{f,x}\,\psi')\rceil(f(t))$. An instantiation based on this pattern is expected to be unproblematic as it unfolds subformulas of ψ and ψ' in $(\psi\ \mathcal{U}_{f,x}\,\psi')\,(f(t))$. The multi-pattern $\{\lceil(\psi\ \mathcal{U}_{f,x}\,\psi')\rceil(x), f(x)\}$ consists of two terms using x. A multi-pattern is instantiated when there are ground terms both of the form $f(x)$ as well as $\lceil(\psi\ \mathcal{U}_{f,x}\,\psi')\rceil(x)$. Again, an instantiation based on this pattern is expected to not introduce recurrent opportunities for matching. The resulting pattern annotated formula can be written:

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

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 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 a LTL solver with a SMT solver to efficiently solve 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 FFP(PL) in Section 3.1. We suspect that one can extend those techniques to identify fragments of FFP(E) with a NP-complete decision problem. Our reduction to 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 [33]. In ETL, we can for instance express the formula $\forall n\geq 0.P(f^{2n}(a))$. It is expressible as $(\nu X\lambda x.X(ff(x))\wedge P(x))(a)$, but does not correspond to a formula in FFP(E). Nevertheless, the satisfiability of such formulas can be checked using the same apparatus developed in this paper.

While simple extensions of FFP(E) are undecidable, there are decidable classes of formulas that can be formulated using functional fixed-points, yet they cannot be formulated in FFP(E). For example [16] 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. Theorem 3.17 implies that the more general logic with until operators is already NEXPTIME-hard with a single function symbol, and simple versions with two function symbols are not decidable (Theorem 3.20). The reduction to LTL does not work when there are multiple bound variables: The LTL reduction requires that at most one flexible 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] and hybrid logic [11], can be applied.

There are ad-hoc ways to extend our methodology to handle fixed-points in the context of analyzing low-level software [6]. In this context, one seeks transitive closures of functions that interact with pointer arithmetic with mostly constant offsets. For example, we may want to compute the transitive closure of $head, f(head + 12), f(f(head + 12) + 12), f(f(f(head + 12) + 12) + 12), \ldots$. There is a direct way to simulate $f(x + 12)$, using a separate function symbol $f_{12}(x)$. The approach is complete when other uses of f are limited, but would like to understand more precisely the limits of how arithmetic (of offsets) can be mixed with fixed-points in a systematic way.

References

[1] R. Alur and T. A. Henzinger. A really temporal logic. *JACM*, 41(1):181–204, 1994.

[2] I. Balaban, A. Pnueli, and L. D. Zuck. Shape analysis of single-parent heaps. In *VMCAI*, LNCS 4349, pages 91–105. Springer, 2007.

[3] M. Ben-Ari, A. Pnueli, and Z. Manna. The temporal logic of branching time. *Acta Inf.*, 20:207–226, 1983.

[4] J. D. Bingham and Z. Rakamaric. A logic and decision procedure for predicate abstraction of heap-manipulating programs. In E. Allen Emerson and K. S. Namjoshi, editors, *VMCAI*, LNCS 3855, pages 207–221. Springer, 2006.

[5] E. Börger, Grädel, and Gurevich. *The Classical Decision Problem*. Springer, 96.

[6] S. Chatterjee, S. K. Lahiri, S. Qadeer, and Z. Rakamaric. A reachability predicate for analyzing low-level software. In O. Grumberg and M. Huth, editors, *TACAS*, volume 4424 of *LNCS*, pages 19–33. Springer, 2007.

[7] E. M. Clarke, O. Grumberg, and Kiyoharu Hamaguchi. Another look at ltl model checking. In D. L. Dill, editor, *CAV*, LNCS 818, pages 415–427. Springer, 1994.

[8] L. de Moura and N. Bjørner. Efficient E-matching for SMT Solvers. In *CADE'07*. Springer, 2007.

[9] L. de Moura and N. Bjørner. Model-based Theory Combination. In *SMT'07*, 2007.

[10] L. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *TACAS*, LNCS 4963. Springer, 2008.

[11] S. Demri and R. Lazic. Ltl with the freeze quantifier and register automata. In *LICS*, pages 17–26. IEEE Computer Society, 2006.

[12] D. Detlefs, G. Nelson, and J. B. Saxe. Simplify: a theorem prover for program checking. *J. ACM*, 52(3):365–473, 2005.

[13] N. Immerman, A. M. Rabinovich, T. W. Reps, S. Sagiv, and G. Yorsh. The boundary between decidability and undecidability for transitive-closure logics. In *CSL*, LNCS 3210, pages 160–174, 2004.

[14] N. Klarlund and A. Møller. *MONA Version 1.4 User Manual*. BRICS, Department of Computer Science, Aarhus University, January 2001. Notes Series NS-01-1. Available from `http://www.brics.dk/mona/`. Revision of BRICS NS-98-3.

[15] S. K. Lahiri and S. Qadeer. Verifying properties of well-founded linked lists. In *Principles of Programming Languages (POPL '06)*, pages 115–126, 2006.

[16] S. K. Lahiri and S. Qadeer. Back to the future: revisiting precise program verification using smt solvers. In *POPL*, pages 171–182. ACM, 2008.

[17] T. Lev-Ami, N. Immerman, T. W. Reps, S. Sagiv, S. Srivastava, and G. Yorsh. Simulating reachability using first-order logic with applications to verification of linked data structures. In *CADE*, LNCS 3632, pages 99–115, 2005.

[18] L. Libkin. *Elements of Finite Model Theory*. Springer, 2004.

[19] R. Manevich, E. Yahav, G. Ramalingam, and S. Sagiv. Predicate abstraction and canonical abstraction for singly-linked lists. In R. Cousot, editor, *VMCAI*, LNCS 3385, pages 181–198. Springer, 2005.

[20] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer, 1995.

[21] S. McPeak and G. C. Necula. Data structure specifications via local equality axioms. In *In CAV*, pages 476–490. Springer, 2005.

[22] A. Møller and M. I. Schwartzbach. The pointer assertion logic engine. In *Programming Language Design and Implementation (PLDI '01)*, pages 221–231, 2001.

[23] G. Nelson. Verifying Reachability Invariants of Linked Structures. In *Principles of Programming Languages (POPL '83)*, pages 38–47, 1983.

[24] G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, 1979.

[25] Z. Rakamarić, J. Bingham, and A. J. Hu. An inference-rule-based decision procedure for verification of heap-manipulating programs with mutable data and cyclic data structures. In *VMCAI '06*, LNCS 4349, pages 106–121. Springer, 2007.

[26] Z. Rakamaric, R. Bruttomesso, A. J. Hu, and A. Cimatti. Verifying heap-manipulating programs in an smt framework. In K. S. Namjoshi, T. Yoneda, T. Higashino, and Y. Okamura, editors, *ATVA*, LNCS 4762, pages 237–252. Springer, 2007.

[27] S. Ranise and C. G. Zarba. A theory of singly-linked lists and its extensible decision procedure. In *SEFM '06*, pages 206–215, 2006.

[28] J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *17th LICS*, pages 55–74. IEEE Computer Society, 2002.

[29] É. Sims. Extending separation logic with fixpoints and postponed substitution. *Theoretical Computer Science*, 351(2):258–275, 2006.

[30] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985.

[31] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *LICS*, pages 332–344. IEEE Computer Society, 1986.

[32] C. Varming and L. Birkedal. Higher-order separation logic in isabelle/holcf. *Electr. Notes Theor. Comput. Sci.*, 218:371–389, 2008.

[33] P. Wolper. Specification and synthesis of communicating processes using an extended temporal logic. In *POPL*, pages 20–33, 1982.

[34] H. Yang, O. Lee, J. Berdine, C. Calcagno, B. Cook, D. Distefano, and P. W. O'Hearn. Scalable shape analysis for systems code. In Aarti Gupta and Sharad Malik, editors, *CAV*, LNCS 5123, pages 385–398. Springer, 2008.

[35] G. Yorsh, A. M. Rabinovich, S. Sagiv, A. Meyer, and A. Bouajjani. A logic of reachable patterns in linked data-structures. LNCS 3921. Springer.