Complete Instantiation-Based Interpolation

Nishant Totla
Indian Institute of Technology Bombay
nishant.totla@gmail.com

Thomas Wies
New York University
wies@cs.nyu.edu

Abstract

Craig interpolation has been a valuable tool for formal methods with interesting applications in program analysis and verification. Modern SMT solvers implement interpolation procedures for the theories that are most commonly used in these applications. However, many application-specific theories remain unsupported, which limits the class of problems to which interpolation-based techniques apply. In this paper, we present a generic framework to build new interpolation procedures via reduction to existing interpolation procedures. We consider the case where an application-specific theory can be formalized as an extension of a base theory with additional symbols and axioms. Our technique uses finite instantiation of the extension axioms to reduce an interpolation problem in the theory extension to one in the base theory. We identify a model-theoretic criterion that allows us to detect the cases where our technique is complete. We discuss specific theories that are relevant in program verification and that satisfy this criterion. In particular, we obtain complete interpolation procedures for theories of arrays and linked lists. The latter is the first complete interpolation procedure for a theory that supports reasoning about complex shape properties of heap-allocated data structures. We have implemented this procedure in a prototype on top of existing SMT solvers and used it to automatically infer loop invariants of list-manipulating programs.

Categories and Subject Descriptors D.2.4 [*Software Engineering*]: Software/Program Verification; F.3.1 [*Logics and Meaning of Programs*]: Specifying and Verifying and Reasoning about Programs; I.2.3 [*Artificial Intelligence*]: Deduction and Theorem Proving

General Terms Algorithms, Theory, Reliability, Verification

Keywords Craig Interpolants, Decision Procedures, Satisfiability Module Theories, Program Analysis, Data Structures

1. Introduction

In his pioneering work [37], McMillan recognized the usefulness of Craig interpolants [13] for the automated construction of abstractions of systems. Since then, interpolation-based algorithms have been developed for a number of problems in program analysis and verification [1, 15, 17, 24, 25, 31, 34, 39]. An important requirement for most of these algorithms is that interpolants are *ground* (i.e., quantifier-free). This is because the computed interpolants again serve as input to decision procedures that only support quantifier-free formulas. Modern SMT solvers implement ground interpolation procedures for the theories that are most commonly used in program verification. This includes theories such as linear arithmetic [8, 9, 23, 38], the theory of uninterpreted function symbols with equality [19, 38, 50], and combinations of such theories [11, 21, 38, 50]. However, many application-specific theories remain unsupported. This limits the class of problems and programs to which interpolation-based algorithms can be applied.

In this paper, we present a generic framework that enables the modular construction of ground interpolation procedures for application-specific theories via a reduction to existing interpolation procedures. We focus on cases where an application-specific theory can be formalized as an extension of a *base theory* with additional symbols and universally quantified axioms. As an example of such a *theory extension*, consider the theory of arrays over integers. Here, the base theory is the theory of linear integer arithmetic, the extension symbols are the array selection and update functions, and the extension axioms are McCarthy's read over write axioms [36], which give meaning to the extension symbols. Theory extensions often appear in practice, e.g., as part of the background theories of verification systems such as BOOGIE [3] and WHY [18], and the tools that are built on top of these systems.

Our starting point is the approach to instantiation-based interpolation for local theory extension presented in [47]. Local theory extensions [46] are extensions for which satisfiability of ground formulas can be decided via a reduction to satisfiability in the base theory. The reduction works by instantiating the extension axioms only with terms that already appear in the input formula. In [47], this instantiation-based reduction approach is applied to the problem of computing ground interpolants in local theory extensions. This technique is used, e.g., in the interpolation procedures underlying the software model checker ARMC [44] and the interpolating prover CSIsat [6]. In [47], the instantiation-based interpolation approach was shown to be complete for a syntactically defined class of local theory extensions. Unfortunately, many interesting application-specific theories do not fall into this class.

Instead of imposing syntactic restrictions, we identify a stronger condition on the theory extension than just locality to ensure completeness of instantiation-based interpolation. We then relate this condition to a semantic property of the models of the theory extension. We refer to this property as the *partial amalgamation property*. This property allows us to systematically construct theory extensions for which the instantiation-based approach produces a complete ground interpolation procedure. The resulting framework then applies to a more general class of theory extensions than [47].

We discuss several non-trivial examples of theories that are relevant in program verification and to which our framework applies. In particular, we consider the theory of arrays with difference function [10]. Using our approach we obtain an alternative ground interpolation procedure for this theory. Unlike the procedure presented in [10], our procedure does not require a dedicated decision proce-

POPL'13, January 23-25, 2013, Rome, Italy.

dure for this specific array theory. Instead, it reduces the interpolation problem to existing interpolation procedures for uninterpreted functions and linear arithmetic.

The second example that we discuss in detail is a variation of Nelson's theory of linked lists with reachability predicates [41], which was studied more recently in [35]. We show that this theory does not admit ground interpolation, unless it is extended (among others) with an additional *join* function. Given two heap nodes, the join function returns the *join point* of the two list segments that start in the given nodes (if such a join point exists). Incidentally, the join function is not just of theoretical interest, but is also useful to express properties about the heap that are important for verifying frame conditions. We prove that our extended theory of linked lists with reachability has partial amalgamation. Using our approach we then obtain the first ground interpolation procedure for a theory that supports reasoning about complex shape properties of heap-allocated data structures. This interpolation procedure has promising applications in CEGAR-based shape analysis [5, 42] and may also provide a new perspective on the construction of shape domains in parametric shape analysis [45].

To show the feasibility of our approach, we have implemented a prototype of our interpolation framework and instantiated it for the theory of linked lists presented in this paper. We have successfully applied the resulting interpolation procedure to automatically infer loop invariants for the verification of list-manipulating programs.

Summary. The main contributions of this paper can be summarized as follows:

- We present a new framework to modularly construct interpolation procedures for application-specific theories.
- We present a model-theoretic criterion that allows us to identify the theories for which our interpolation framework is complete.
- We present examples of theories that are important for program verification and to which our framework applies. In particular, we present the first decidable theory for reasoning about complex shape properties of heap-allocated data structures that admits ground interpolation.
- We report on our experience with a prototype implementation of our framework, which we have successfully used to infer loop invariants of simple list-manipulating programs.

An extended version of this paper including proofs of key lemmas and theorems is available as a technical report [48].

2. Motivation and Overview

We motivate our approach using the concrete application of interpolation to the problem of inferring invariants for program verification. Consider the reverse function given in Figure 1. This function takes a pointer x to a singly-linked list as input, reverses the list, and then returns a pointer to the head of the reversed list.

Our goal is to verify that the reverse function preserves acyclicity, i.e., if the input list is acyclic, then so is the output list. We can express acyclicity of list x by saying that null is reachable from x by following the n pointer fields in the heap. Using the notation that we formally introduce in Section 5.2, this is denoted by the *reachability predicate* $x \xrightarrow{n} \mathsf{null}$. Hence, the property we want to verify is that if the pre-condition $x \xrightarrow{n} \mathsf{null}$ holds at the entry point of function reverse, then the same formula holds again at the return point.

The graph in Figure 1 depicts an intermediate state of the heap that is observed during the execution of reverse when the function is applied to an acyclic list of length six. This state is observed at the entry point of the while loop, after the first three iterations of the loop. An appropriate inductive loop invariant for a Hoare proof of

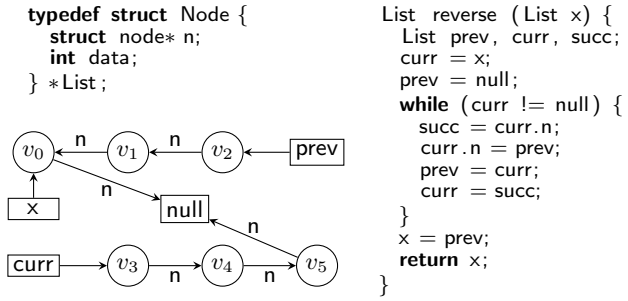

Figure 1. C code for in-place reversal of a linked list. The graph depicts a reachable program state at the entry point of the while loop in function reverse.

our verification goal must capture the situation depicted in Figure 1, but abstract from the concrete length of the list segments. That is, the loop invariant must express that the list segments pointed to by prev and curr are acyclic (in fact, only the former is strictly necessary for the proof), and that the two list segments are disjoint. An appropriate inductive loop invariant is given by the following formula:

$$\mathsf{prev} \xrightarrow{n} \mathsf{null} \wedge \mathsf{curr} \xrightarrow{n} \mathsf{null} \wedge (\mathsf{prev} \curlyvee^{n} \mathsf{curr}) = \mathsf{null} \qquad (1)$$

The term $(\mathsf{prev} \curlyvee^{n} \mathsf{curr})$ denotes the *join point* of the list segments starting from prev and curr, i.e., $(\mathsf{prev} \curlyvee^{n} \mathsf{curr})$ is the first node that is reachable from both prev and curr by following n pointer fields in the heap, unless such a node does not exist, in which case its value is prev. The formula $(\mathsf{prev} \curlyvee^{n} \mathsf{curr}) = \mathsf{null}$, thus, implies the disjointness of the two list segments. Note that this formula cannot be expressed in terms of the reachability predicate, unless we allow universal quantification over heap nodes.

We next describe how to compute inductive loop invariants such as (1) using our instantiation-based interpolation approach.

2.1 Interpolation-Based Program Verification

Given an unsatisfiable conjunction of formulas $A \wedge B$, an interpolant for $A \wedge B$ is a formula I such that I is implied by A, the conjunction $I \wedge B$ is unsatisfiable, and I only speaks about common symbols of A and B. A popular approach to interpolation-based verification uses bounded model checking to generate infeasible error traces of the analyzed program. These infeasible error traces are then translated into unsatisfiable formulas $A \wedge B$, where A and B encode a partition of the trace into a prefix and suffix trace. An interpolant I for $A \wedge B$ then describes a set of program states that (1) includes all states that are reachable by executing the prefix of the trace and (2) only includes states from which no feasible execution of the suffix is possible. The interpolant I is then used as a candidate invariant to refine the search for additional infeasible error traces. This process is continued until a fixed point is reached, i.e., until an inductive invariant has been computed that proves the program correct. We illustrated this approach through an example.

The left-hand side of Figure 2 shows an error trace of function reverse that is obtained by unrolling the while loop three times. The first and last assume statements correspond to the pre-condition, respectively, the negated post-condition of reverse. This error trace is infeasible, i.e., there is no execution that reaches the end of the trace (note that a failing assume statement blocks an execution). The right-hand side of Figure 2 shows an encoding of this error trace into a first-order formula using static single assignments. Note that the symbols $\bullet \xrightarrow{\bullet} \bullet$, $\bullet.\bullet$, and $\bullet[\bullet := \bullet]$ are interpreted. That

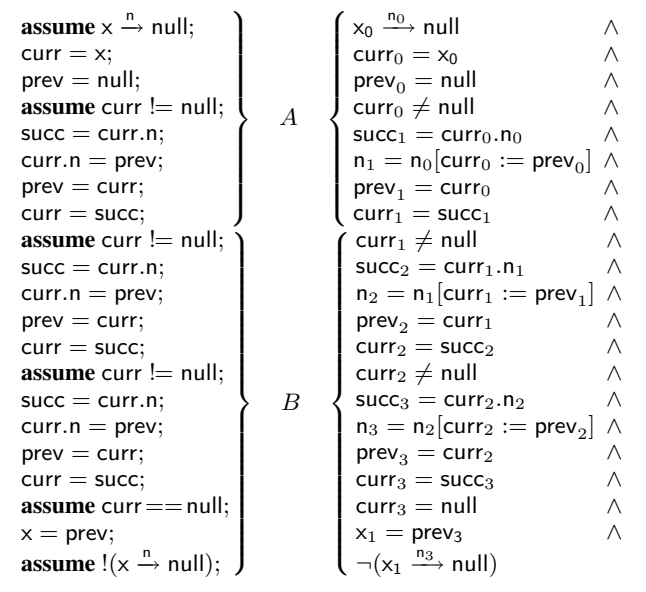

Figure 2. Spurious error trace of function reverse and its encoding as a trace formula

is, they are given meaning by a specific first-order theory, here the theory of linked lists with reachability that we introduce in Section 5.2. The symbol $\bullet \xrightarrow{\bullet} \bullet$ is interpreted as described above. The symbol $\bullet.\bullet$ denotes field dereference and the symbol $\bullet[\bullet := \bullet]$ field update. The remaining symbols such as $curr_0$ and n_1 are uninterpreted. We call this formula the *trace formula* of the error trace because the valuations of uninterpreted symbols that make the trace formula true exactly correspond to the feasible executions of the trace. Since the error trace is infeasible, its trace formula is unsatisfiable. We can now split the trace formula into two parts A and B, say, where A corresponds to the prefix of the trace up to the end of the first iteration of the while loop and B to the remainder of the trace. This is depicted in Figure 2. Since $A \wedge B$ is unsatisfiable, we can interpolate the two formulas. A possible ground interpolant for this choice of A and B is:

$$prev_1 \xrightarrow{n_1} \text{null} \wedge curr_1 \xrightarrow{n_1} \text{null} \wedge prev_1 \overset{n_1}{\curlyvee} curr_1 = \text{null} \quad (2)$$

Note that this is a valid interpolant for A and B. In particular, it only speaks about uninterpreted symbols that are common to both A and B. Further note that (modulo renaming of variables) formula (2) exactly corresponds to the inductive loop invariant (1) of reverse. Formula (2) is also the exact interpolant that the prototype implementation of our interpolation framework produces for this particular conjunction $A \wedge B$. We next describe through an example how our interpolation framework works in detail.

2.2 Instantiation-Based Interpolation through an Example

Our interpolation framework is parameterized by a theory extension. This theory extension consists of the base theory, for which we assume that a ground interpolation procedure exists, and the symbols and axioms that extend the base theory. In our example, we consider the theory of linked lists with reachability, where the base theory is the *empty theory*. That is, the base theory only supports uninterpreted constants and equality. The extension symbols are the symbols $\bullet \overset{\bullet}{\curlyvee} \bullet$, $\bullet.\bullet$, and $\bullet[\bullet := \bullet]$, which we described earlier, as well as the *constrained* reachability predicate $\bullet \xrightarrow{\bullet/\bullet} \bullet$. Intuitively, $a \xrightarrow{f/c} b$ means that b is reachable via an f-path from a that does not go through c. In particular, $a \xrightarrow{f} b$ is simply a shorthand for $a \xrightarrow{f/b} b$. The meaning of the extension symbols is given by the extension axioms shown in Figure 6 of Section 5.2. All free variables appearing in these axioms are implicitly universally quantified. Note in particular how the constrained reachability predicate is used to define reachability with respect to an updated field $f[u := v]$ in terms of reachability with respect to field f. In the following, we denote this set of extension axioms by $\mathcal{K}$.

Instantiation-based interpolation reduces the computation of interpolants in the theory extension to the problem of computing interpolants in the base theory, thus, effectively building new interpolation procedures by reusing existing ones. The reduction works by *turning* the interpreted extension symbols into uninterpreted ones. This is accomplished by generating finitely many ground instances $\mathcal{K}[T]$ of the extension axioms $\mathcal{K}$ for a finite set of terms T that is computed from the input formula $A \wedge B$. The set of terms T is chosen such that the formula $\mathcal{K}[T] \wedge A \wedge B$ is already unsatisfiable in the base theory. If in addition, the set $\mathcal{K}[T]$ does not contain instances that mix non-shared symbols of A and B, then $\mathcal{K}[T]$ can be divided into $\mathcal{K}[T] = \mathcal{K}[T_A] \cup \mathcal{K}[T_B]$ where $\mathcal{K}[T_A]$ contains only symbols of A and $\mathcal{K}[T_B]$ only symbols of B. That is, we obtain an instance $A_0 \wedge B_0$ of an interpolation problem for the base theory (modulo uninterpreted functions) where $A_0 = \mathcal{K}[T_A] \wedge A$ and $B_0 = \mathcal{K}[T_B] \wedge B$. We then compute a ground interpolant I_0 for $A_0 \wedge B_0$ using the interpolation procedure for the base theory. Finally, from I_0 we reconstruct a ground interpolant I for $A \wedge B$.

We illustrate this approach by computing an interpolant for the following formula $A \wedge B$:

$$\underbrace{c \xrightarrow{f} a \wedge a.f = c}_{A} \wedge \underbrace{c \xrightarrow{f} b \wedge \neg b \xrightarrow{f} c}_{B} \quad (3)$$

Note that this conjunction is unsatisfiable in the theory of linked lists with reachability because A implies that c lies on an f cycle, while B implies that this is not the case.

From the formula $A \wedge B$ we compute the sets of terms T_A and T_B that we use to instantiate the extension axioms. In our example, we use $T_A = \{a, c, c.f\}$ and $T_B = \{b, c, c.f\}$. Figure 3 then shows the resulting sets of ground clauses $A_0 = A \cup \mathcal{K}[T_A]$ and $B_0 = B \cup \mathcal{K}[T_B]$. Note that we omit all instances of extension axioms that are not needed to prove unsatisfiability of $A_0 \wedge B_0$.

To see why the conjunction $A_0 \wedge B_0$ is unsatisfiable, suppose that $a = c$. Then clause 2 in A_0 implies $c.f = c$. If on the other hand $a \neq c$, then clauses 4 and 2 imply $a \xrightarrow{f} c$. Hence together with 8 and 6 this implies $a \xrightarrow{f/a} c.f \vee c.f \xrightarrow{f} a$. If $c.f \xrightarrow{f} a$, then 3 implies $c.f \xrightarrow{f} c$. If $a \xrightarrow{f/a} c.f$ then from 7 follows again $c.f \xrightarrow{f} c$ because otherwise clause 5 gives $a = c$, which contradicts the assumption. Thus, A_0 implies the formula $I \equiv c = f.c \vee c.f \xrightarrow{f} c$. Using similar reasoning we can show that $I \wedge B_0$ is unsatisfiable. Since I only speaks about common symbols of A_0 and B_0, it is an interpolant for $A_0 \wedge B_0$ and hence also for $A \wedge B$.

Note that in the above derivation of the interpolant I, all function and predicate symbols in A_0 and B_0 where treated as uninterpreted symbols, i.e., we can compute I by applying an interpolation procedure for the theory of uninterpreted functions with equality to the formula $A_0 \wedge B_0$. We thus reduced the problem of computing ground interpolants in the theory of linked lists with reachability to computing ground interpolants in the combination of the base theory (which is empty in our case) with the theory of uninterpreted functions and equality.

The crux of this instantiation-based reduction approach is whether it is indeed always possible to compute sets of terms T_A and T_B from $A \wedge B$ such that the reduced formula $A_0 \wedge B_0$ is an interpolation problem for the base theory. That is, to find T_A and T_B

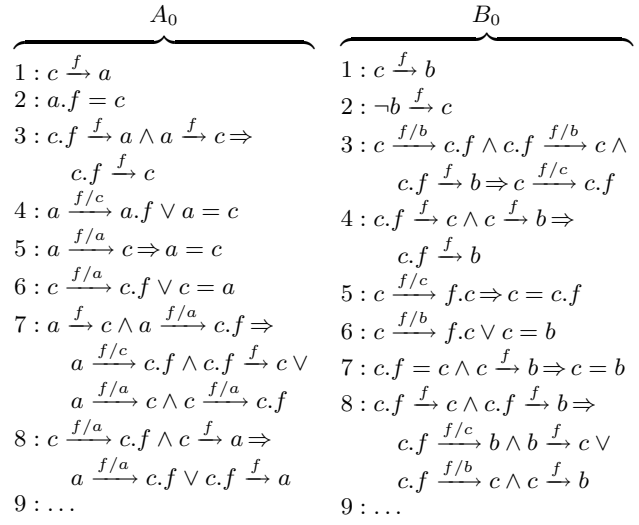

Figure 3. Interpolation problem $A_0 \wedge B_0$ that is obtained from (3) after instantiation of the extension axioms. All function and predicate symbols are uninterpreted.

such that (1) $A_0 \wedge B_0$ is unsatisfiable and (2) A_0, B_0 do not share terms that are not already shared by A, B. It is here where our semantic completeness criterion of partial amalgamation comes into play. It allows us to systematically construct these sets of terms.

3. Preliminaries

In the following, we define the syntax and semantics of formulas. We further recall the notions of partial structures and (Ψ-)local theory extensions as defined in [28, 46]. Finally, we define the problem that interpolation problems we are considering.

Sorted first-order logic. We present our problem in sorted first-order logic with equality. A *signature* Σ is a tuple (S, Ω), where S is a countable set of sorts and Ω is a countable set of function symbols f with associated arity $n \geq 0$ and associated sort $s_1 \times \cdots \times s_n \rightarrow s_0$ with $s_i \in S$ for all $i \leq n$. Function symbols of arity 0 are called *constant symbols*. We assume that all signatures contain a dedicated sort $\mathsf{bool} \in S$ and dedicated equality symbols $=_s \in \Omega$ of sort $s \times s \rightarrow \mathsf{bool}$ for all sorts $s \in S \setminus \{\mathsf{bool}\}$. Note that we generally treat predicate symbols of sort $s_1, \ldots, s_n$ as function symbols of sort $s_1 \times \ldots \times s_n \rightarrow \mathsf{bool}$ and we typically drop sort annotations from equality symbols. Terms are built as usual from the function symbols in Ω and (sorted) variables taken from a countably infinite set X that is disjoint from Ω. A term t is said to be *ground*, if no variable appears in t.

A Σ-*atom* A is a Σ-term of sort bool. We use infix notation for atoms built from the equality symbol. A Σ-*formula* F is defined via structural recursion as either one of A, $\neg F_1$, $F_1 \wedge F_2$, or $\forall x : s.F_1$, where A is a Σ-atom, F_1 and F_2 are Σ-formulas, and $x \in X$ is a variable of sort $s \in S$. We typically omit sort annotations from quantifiers if this causes no confusion. We use syntactic sugar for Boolean constants ($\top$, $\bot$), disjunctions ($F_1 \vee F_2$), implications ($F_1 \Rightarrow F_2$), and existential quantification ($\exists x.F_1$).

Total and partial structures. Given a signature $\Sigma = (S, \Omega)$, a *partial* Σ-*structure* M is a function that maps each sort $s \in S$ to a non-empty set $M(s)$ and each function symbol $f \in \Omega$ of sort $s_1 \times \cdots \times s_n \rightarrow s_0$ to a partial function $M(f) : M(s_1) \times \cdots \times M(s_n) \rightharpoonup M(s_0)$. We denote by $[M]$ the *support* of M which is the non-disjoint union of the interpretation of all sorts in M. We assume that all partial structures interpret the sort bool by the two-element set of Booleans $\{0, 1\}$. We further assume that all structures M interpret each symbol $=_s$ by the equality relation on $M(s)$. A partial structure M is called *total structure* or simply *structure* if it interprets all function symbols by total functions. For a Σ-structure M where Σ extends a signature Σ_0 with additional sorts and function symbols, we write $M|_{\Sigma_0}$ for the Σ_0-structure obtained by restricting M to Σ_0. Given partial Σ-structures M and N, a *weak embedding* of M into N is a total injective function $h : [M] \rightarrow [N]$ such that for all $f \in \Sigma$, and $a_1, \ldots, a_n \in M$, if $M(f)$ is defined on $(a_1, \ldots, a_n)$ then $N(f)$ is defined on $(h(a_1), \ldots, h(a_n))$ and $h(M(f)(a_1, \ldots, a_n)) = N(f)(h(a_1), \ldots, h(a_n))$. If h is a weak embedding between M and N, then we denote this by $h : M \rightarrow N$. We say that M *weakly embeds* into N if a weak embedding of M into N exists. A weak embedding between total structures is simply called *embedding*. If M weakly embeds into N and $[M] \subseteq [N]$, we call M a (partial) *substructure* of N, which (abusing notation) is denoted by $M \subseteq N$.

Given a total structure M and a *variable assignment* $\beta : X \rightarrow M$, the evaluation $t_{M,\beta}$ of a term t in M, β is defined as usual. For the evaluation of a ground term t in M we write just $M(t)$. A quantified variable of sort s ranges over all elements of $M(s)$. From the interpretation of terms the notions of satisfiability, validity, and entailment of atoms, formulas, clauses, and sets of clauses in total structures are derived as usual. In particular, we use the standard interpretation of propositional connectives in classical logic. We write $M, \beta \models F$ if M satisfies F under β where F is a formula, a clause, or a set of clauses. We write $M \models F$ if F is valid in M. In this case we also call M a *model* of F. The interpretation $t_{M,\beta}$ of a term t in a partial structure M is as for total structures, except that if $t = f(t_1, \ldots, t_n)$ for $f \in \Omega$ then $t_{M,\beta}$ is undefined if either $t_{i\,M,\beta}$ is undefined for some i, or $(t_{1\,M,\beta}, \ldots, t_{n\,M,\beta})$ is not in the domain of $M(f)$. We say that a partial structure M *weakly satisfies* a literal L under β, written $M, \beta \models_w L$, if (i) L is an atom A and either $A_{M,\beta} = 1$ or $A_{M,\beta}$ is undefined, or (ii) L is a negated atom $\neg A$ and either $A_{M,\beta} = 0$ or $A_{M,\beta}$ is undefined. The notion of weak satisfiability is extended to (sets of) clauses as for total structures. A clause C (respectively, a set of clauses) is *weakly valid* in a partial structure M if M weakly satisfies C for all assignments β. We then call M a *weak partial model* of C.

Theories and theory extensions. A *theory* $\mathcal{T}$ over signature Σ is simply a set of Σ-formulas. We consider theories $\mathcal{T}$ defined as a set of Σ-formulas that are consequences of a given set of clauses $\mathcal{K}$. We call $\mathcal{K}$ the *axioms* of the theory $\mathcal{T}$ and we often identify $\mathcal{K}$ and $\mathcal{T}$. For a theory $\mathcal{T}$ and formulas (clauses, sets of clauses) F and G, we use $F \models_{\mathcal{T}} G$ as a short-hand for $\mathcal{T} \cup F \models G$.

Let $\Sigma_0 = (S_0, \Omega_0)$ be a signature and assume that signature $\Sigma_1 = (S_0 \cup S_e, \Omega_0 \cup \Omega_e)$ extends Σ_0 by new sorts S_e and function symbols Ω_e. We call the elements of Ω_e *extension symbols* and terms starting with extension symbols *extension terms*. A theory $\mathcal{T}_1$ over Σ_1 is an *extension* of a theory $\mathcal{T}_0$ over Σ_0, if $\mathcal{T}_1$ is obtained from $\mathcal{T}_0$ by adding a set of (universally quantified) Σ_1-clauses $\mathcal{K}$.

Ψ-local theory extensions. The following definition captures one specific variant of (Ψ-)local theory extensions that is discussed together with other variants of this notion in [46] and [28].

Let $\mathcal{T}$ be a theory over signature $\Sigma_0 = (S_0, \Omega_0)$ and $\mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$ a theory extension of $\mathcal{T}_0$ with finite $\mathcal{K}$ and signature $\Sigma_1 = (S_0 \cup S_e, \Omega_0 \cup \Omega_e)$. In the following, when we refer to a set of ground clauses G, we assume they are over the signature Σ_1^c which extends Σ_1 with a set of new constant symbols Ω^c. For a set of clauses $\mathcal{K}$, we denote by $\mathsf{st}(\mathcal{K})$ the set of all ground subterms that appear in $\mathcal{K}$. An *embedding closure* for $\mathcal{T}_1$ is a function Ψ associating with a set of (universally quantified) clauses $\mathcal{K}$ and a finite set of ground terms T a finite set $\Psi(T)$ of ground terms

such that (i) all ground subterms in $\mathcal{K}$ and T are in $\Psi(T)$; (ii) Ψ is monotone, i.e., for all sets of ground terms T, T' if $T \subseteq T'$ then $\Psi(T) \subseteq \Psi(T')$; (iii) Ψ is idempotent, i.e., for all sets of ground terms T, $\Psi(\Psi(T)) \subseteq \Psi(T)$. (iv) Ψ is compatible with any map h between constants, i.e., $\Psi(h(T)) = h(\Psi(T))$ where h is homomorphically extended to terms. For a set of ground clauses G, we denote by $\Psi(G)$ the set $\Psi(\mathsf{st}(G))$. Let $\mathcal{K}[\Psi(G)]$ be the set of instances of $\mathcal{K}$ in which all extension terms are in $\Psi(G)$. We say that $\mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$ is a Ψ-*local theory extension* if there exists an embedding closure Ψ such that for every finite set of ground clauses G, $\mathcal{T}_1 \cup G \models \bot$ iff $\mathcal{T}_0 \cup \mathcal{K}[\Psi(G)] \cup G \models \bot$. Theory extension $\mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$ is a *local theory extension*, if it is a Ψ-local extension, where Ψ is defined as $\Psi(T) = \mathsf{st}(\mathcal{K}) \cup \mathsf{st}(T)$.

Craig interpolation modulo theories. We use a notion of Craig interpolation modulo theories where interpreted symbols are considered to be shared between formulas. Let Σ be a signature and $\mathcal{T}$ a Σ-theory. Let further Σ^c be the signature Σ extended with fresh constant symbols Ω^c. We say that a Σ^c-term t is *shared* between two sets of Σ^c-terms T_A and T_B, if all constants from Ω^c in t appear in both T_A and T_B, i.e., $\mathsf{st}(t) \cap \Omega^c \subseteq \mathsf{st}(T_A) \cap \mathsf{st}(T_B)$. We say that t is T_A-*pure* if $\mathsf{st}(t) \cap \Omega^c \subseteq \mathsf{st}(T_A)$, respectively, t is T_B-*pure* if $\mathsf{st}(t) \cap \Omega^c \subseteq \mathsf{st}(T_B)$. We extend these notions from sets of terms T_A and T_B to clauses and sets of clauses, as expected.

Given a conjunction $A \wedge B$ of Σ^c-formulas A, B that is unsatisfiable in $\mathcal{T}$, a *Craig interpolant* for $A \wedge B$ is a Σ^c-formula I such that: (a) I is a consequence of A in $\mathcal{T}$: $A \models_{\mathcal{T}} I$, (b) the conjunction of I and B is unsatisfiable in $\mathcal{T}$: $I \wedge B \models_{\mathcal{T}} \bot$, and (c) all terms in $\mathsf{st}(I)$ are shared between A and B. We say that $\mathcal{T}$ *admits ground interpolation* if for all finite sets of Σ^c-ground clauses A and B with $A \cup B \models_{\mathcal{T}} \bot$, there exists a finite set of Σ^c-ground clauses I that is a Craig interpolant for $A \wedge B$.

4. Instantiation-Based Interpolation

We now present our framework for instantiation-based interpolation. In the following, when we refer to a theory extension $\mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$, we denote by Σ_0 the signature of $\mathcal{T}_0$ and by $\Sigma_1 = \Sigma_0 \cup \Sigma_e$ the signature of $\mathcal{T}_1$, where $\Sigma_e = (S_e, \Omega_e)$ are the extension symbols and sorts.

In the case of local theories, the instantiation-based reduction approach to interpolation works as follows [47]. Suppose we are given sets of ground clauses A and B over Σ_1, whose conjunction is unsatisfiable in $\mathcal{T}_1$. The goal is to compute a ground interpolant I for $A \wedge B$. Locality tells us that we can reduce the problem of checking (un)satisfiability of $A \cup B$ in $\mathcal{T}_1$ to checking (un)satisfiability of $\mathcal{K}[A \cup B] \cup A \cup B$ in $\mathcal{T}_0$. Here, $\mathcal{K}[A \cup B]$ is the (finite) set of instances of clauses in $\mathcal{K}$ that are obtained by replacing the free variables appearing below extension terms in $\mathcal{K}$ with ground subterms appearing in $\mathcal{K} \cup A \cup B$, such that all resulting ground extension terms in $\mathcal{K}[A \cup B]$ already appear in $A \cup B$. The instances $\mathcal{K}[A \cup B]$ can be partitioned into A-pure instances $\mathcal{K}_A$ (obtained by instantiating clauses in $\mathcal{K}$ with terms from A only), B-pure instances $\mathcal{K}_B$ (obtained by instantiating clauses in $\mathcal{K}$ with terms from B only), and mixed instances $\mathcal{K}_C$ (obtained by instantiating clauses in $\mathcal{K}$ with terms from both A and B). If it is possible to find a finite set of non-mixed terms that separates the mixed instances $\mathcal{K}_C$ into sets of A-pure instances $\mathcal{K}_{C,A}$ and B-pure instances $\mathcal{K}_{C,B}$, then we obtain an interpolation problem for the base theory $A_0 \cup B_0 \models_{\mathcal{T}_0} \bot$. Here, A_0 and B_0 are the results of applying Ackermann's expansion to eliminate the extension symbols from the sets of clauses $\mathcal{K}_A \cup \mathcal{K}_{C,A} \cup A$, respectively, $\mathcal{K}_B \cup \mathcal{K}_{C,B} \cup B$. From a ground interpolant I_0 for $A_0 \wedge B_0$ one can then easily reconstruct a ground interpolant I for $A \wedge B$.

The question is whether it is indeed possible to separate the instances $\mathcal{K}_C$ into A-pure and B-pure parts. The result in [47] identifies sufficient conditions on the theory extension to ensure this. Unfortunately, these restrictions are quite severe. In particular, the axioms in $\mathcal{K}$ are required to be Horn clauses of a specific form, which rules out many interesting applications. Instead of imposing such syntactic restrictions on the theory, we first identify a stronger completeness condition on the theory extension than just (Ψ-)locality and then relate this condition to a semantic condition on the models of the theory. By combining these two results, we obtain a framework of complete instantiation-based ground interpolation procedures for a more general class of theory extensions.

proc Interpolate
  input
    $\mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$: theory extension
    $\mathsf{Interpolate}_0$: ground interpolation procedure for $\mathcal{T}_0 \cup \mathcal{T}_{\mathsf{EUF}}$
    W : amalgamation closure for $\mathcal{T}_1$
    A, B : sets of ground Σ_1^c-clauses with $A \cup B \models_{\mathcal{T}_1} \bot$
  begin
    $A_0 := A \cup \mathcal{K}[W(A, B)]$
    $B_0 := B \cup \mathcal{K}[W(B, A)]$
    $I := \mathsf{Interpolate}_0(A_0, B_0)$
    return I
  end

Figure 4. Generic instantiation-based interpolation procedure.

4.1 W-Separable Theories

To formalize the set of terms that is required to separate the mixed instances of A and B, we introduce the notion of an *amalgamation closure*. An *amalgamation closure* for a theory extension $\mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$ is a function W associating with finite sets of ground terms T_A and T_B, a finite set $W(T_A, T_B)$ of ground terms such that (i) all ground subterms in $\mathcal{K}$ and T_A are in $W(T_A, T_B)$; (ii) W is monotone, i.e., for all $T_A \subseteq T'_A$, $T_B \subseteq T'_B$, $W(T_A, T_B) \subseteq W(T'_A, T'_B)$; (iii) W is a closure, i.e., $W(W(T_A, T_B), W(T_B, T_A)) \subseteq W(T_A, T_B)$; (iv) W is compatible with any map h between constants satisfying $h(c_1) \neq h(c_2)$, for all constants $c_1 \in \mathsf{st}(T_A)$, $c_2 \in \mathsf{st}(T_B)$ that are not shared between T_A and T_B, i.e., for any such h we require $W(h(T_A), h(T_B)) = h(W(T_A, T_B))$; and (v) $W(T_A, T_B)$ only contains T_A-pure terms. For sets of ground clauses A, B we write $W(A, B)$ as a shorthand for $W(\mathsf{st}(A), \mathsf{st}(B))$. For the remainder of this section W always refers to an amalgamation closure.

We next identify the cases where the instances of extension axioms can be separated.

Definition 1. *We say that a theory extension $\mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$ is* W-separable *if for all sets of ground clauses A and B, $\mathcal{T}_1 \cup A \cup B \models \bot$ iff $\mathcal{T}_0 \cup \mathcal{K}[W(A, B)] \cup A \cup \mathcal{K}[W(B, A)] \cup B \models \bot$.*

From this definition we directly obtain the following theorem.

Theorem 2. *If $\mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$ is W-separable, then it is Ψ-local where Ψ is defined by $\Psi(T) = W(T, T)$, for all sets of ground terms T.*

Our generic instantiation-based interpolation procedure is described in Figure 4. Procedure Interpolate reduces the given interpolation problem $A \wedge B$ for the theory extension $\mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$, to an interpolation problem $A_0 \wedge B_0$ in $\mathcal{T}_0 \cup \mathcal{T}_{\mathsf{EUF}}$, where $\mathcal{T}_{\mathsf{EUF}}$ is the theory of uninterpreted function symbols with equality. For W-separable theory extensions, procedure Interpolate is sound and complete, provided that a complete ground interpolation procedure $\mathsf{Interpolate}_0$ for $\mathcal{T}_0 \cup \mathcal{T}_{\mathsf{EUF}}$ exists:

Theorem 3. *Let $\mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$ be a theory extension such that:*

1. $\mathcal{T}_0 \cup \mathcal{T}_{\mathsf{EUF}}$ *has a ground interpolation procedure* $\mathsf{Interpolate}_0$,
2. *all free variables in* $\mathcal{K}$ *appear below extension symbols, and*
3. $\mathcal{T}_1 \supseteq \mathcal{T}_0$ *is* W*-separable.*

Then Interpolate *is a ground interpolation procedure for* $\mathcal{T}_1$.

4.2 Identifying W-Separable Theories

We now present our semantic criterion to identify W-separable theories. Let us begin by recalling the model theoretic notion of amalgamation [32]. An *amalgam* for a theory $\mathcal{T}$ is a tuple (M_A, M_B, M_C) where M_A, M_B, M_C are models of $\mathcal{T}$ with $M_A \supseteq M_C \subseteq M_B$ and $[M_C] = [M_A] \cap [M_B]$. Theory $\mathcal{T}$ has the *amalgamation property* if for every amalgam (M_A, M_B, M_C) of $\mathcal{T}$, there exists a model M_D of $\mathcal{T}$, and embeddings $h_A : M_A \to M_D$ and $h_B : M_B \to M_D$ such that $h_A|_{[M_C]} = h_B|_{[M_C]}$. If in addition $h_A[M_A] \cap h_B[M_B] = h_A[M_C] = h_B[M_C]$ where for any sets $X \subseteq Y$ and function f with domain Y, $f\,X = \{\, f(x) \mid x \in X \,\}$, then $\mathcal{T}$ is said to have the *strong amalgamation property*. Note that $\mathcal{T}$ has the strong amalgamation property iff for all models M_A, M_B, M_C of $\mathcal{T}$ with $M_A \supseteq M_C \subseteq M_B$ and $[M_C] = [M_A] \cap [M_B]$ there exists a model M_D of $\mathcal{T}$ with $M_A \subseteq M_D \supseteq M_B$.

It is well-known that amalgamation and ground interpolation are strongly related:

Theorem 4 ([2])**.** *A theory* $\mathcal{T}$ *has ground interpolation iff* $\mathcal{T}$ *has the amalgamation property.*

Theorem 4 provides an effective tool to check whether a given theory admits ground interpolation. Unfortunately, the amalgamation property only tells us that ground interpolants exist, not how to compute them (other than by brute-force enumeration). To remedy this fact, we define a related notion of *partial* amalgamation that refers to partial instead of total models and weak embeddings instead of embeddings. This notion allows us to characterize W-separable theories. Together with Theorem 3, we then obtain a powerful model theoretic criterion that does not just allow us to prove the existence of ground interpolants, but also tells us how to generically construct the accompanying interpolation procedure by applying Theorem 3.

For a weak partial model M of a theory extension $\mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$, we denote by $T(\Omega_e, M)$ the set of terms $T(M) = \{\, f(a_1, \ldots, a_n) \mid a_i \in [M], f \in \Omega_e, M(f)(a_1, \ldots, a_n) \text{ defined} \,\}$ where we treat the elements of the support $[M]$ as constant symbols that are interpreted by themselves. Further, we denote by $\mathsf{PMod}(\mathcal{T}_1)$ the set of all weak partial Σ_1-models M of $\mathcal{T}_1$ in which all symbols in Σ_0 are totally defined and $T(M)$ is finite. Let W be an amalgamation closure for theory extension $\mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$. A *partial* W*-amalgam* for $\mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$ is a tuple (M_A, M_B, M_C) where (i) $M_A, M_B, M_C \in \mathsf{PMod}(\mathcal{T}_1)$; (ii) M_C is a substructure of both M_A and M_B; (iii) $[M_C] = [M_A] \cap [M_B]$; (vi) both $T(M_A)$ and $T(M_B)$ are closed under W, i.e., $W(T(M_A), T(M_B)) \subseteq T(M_A)$ and $W(T(M_B), T(M_A)) \subseteq T(M_B)$; and (v) $T(M_A) \cap T(M_B) \subseteq T(M_C)$.

Definition 5. *A theory extension* $\mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$ *is said to have the* partial amalgamation property *with respect to* W *if for all partial* W*-amalgams* (M_A, M_B, M_C) *there exists a model* M_D *of* $\mathcal{T}_1$*, and weak embeddings* $h_A : M_A \to M_D$ *and* $h_B : M_B \to M_D$ *such that* $h_A|_{[M_C]} = h_B|_{[M_C]}$.

To simplify matters, we assume that the extension axioms $\mathcal{K}$ are in a specific normal form: a clause C is called Σ_1*-flat* if no term that occurs in C below a predicate symbol or the symbol $=$ contains nested function symbols. A clause C is called Σ_1*-linear* if (i) whenever a variable occurs in two non-variable terms in C that do not start with a predicate or the equality symbol, the two terms are identical, and if (ii) no such term contains two occurrences of the same variable. Note that every set of extension axioms can be syntactically transformed into one that is Σ_1-flat and Σ_1-linear.

Intuitively, a weak partial model M of $A_0 \wedge B_0$ corresponds to a partial W-amalgam (M_A, M_B, M_C) where M_A is obtained from M by restricting M to the terms in A_0, M_B is obtained by restricting M to the terms in B_0, and M_C is obtained by restricting M to the common terms of A_0 and B_0. Partial amalgamation then tells us that we can always obtain a total model of $A_0 \wedge B_0$ from M. This is what the following theorem says:

Theorem 6. *Let* $\mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$ *be a theory extension with* $\mathcal{K}$ Σ_1*-linear and* Σ_1*-flat. If* $\mathcal{T}_1$ *has the partial amalgamation property with respect to* W*, then* $\mathcal{T}_1$ *is* W*-separable.*

Finally, in order to apply Theorem 3, we need to be able to identify the cases where a ground interpolation procedure $\mathsf{Interpolate}_0$ for the theory $\mathcal{T}_0 \cup \mathcal{T}_{\mathsf{EUF}}$ exists. One possibility is that we view $\mathcal{T}_0 \cup \mathcal{T}_{\mathsf{EUF}}$ as the disjoint combination of the theories $\mathcal{T}_0$ and $\mathcal{T}_{\mathsf{EUF}}$. In this case, we require that $\mathcal{T}_0$ is decidable, has the strong amalgamation property, and is stably infinite. Since $\mathcal{T}_{\mathsf{EUF}}$ satisfies the same properties, a ground interpolation procedure for the disjoint combination of the theories $\mathcal{T}_0$ and $\mathcal{T}_{\mathsf{EUF}}$ exists, as follows from [11, Corollary 1].

Corollary 7. *Let* $\mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$ *be a theory extension such that:*

1. *satisfiability of sets of ground clauses is decidable for* $\mathcal{T}_0$,
2. $\mathcal{T}_0$ *is stably infinite*
3. $\mathcal{T}_0$ *has the strong amalgamation property,*
4. *all free variables in* $\mathcal{K}$ *appear below extension symbols, and*
5. $\mathcal{T}_1 \supseteq \mathcal{T}_0$ *is* W*-separable.*

Then Interpolate *is a ground interpolation procedure for* $\mathcal{T}_1$ *where* $\mathsf{Interpolate}_0$ *is the ground interpolation procedure for the disjoint theory combination* $\mathcal{T}_0 \cup \mathcal{T}_{\mathsf{EUF}}$.

Alternatively, we can view $\mathcal{T}_0 \cup \mathcal{T}_{\mathsf{EUF}}$ as a local extension of $\mathcal{T}_0$ and use results from [47] to obtain the procedure $\mathsf{Interpolate}_0$. This yields different requirements on the base theory than the ones stated in Corollary 7. See the technical report [48] for further details.

5. Examples

Our framework of complete instantiation-based interpolation applies to many known local theory extensions, including those described in [47]. In the following, we discuss two non-trivial examples that go beyond the theories considered in [47].

5.1 Theory of Arrays with Difference Function

Our first example is the theory of arrays with difference function that has been recently investigated in [10]. We define this theory of arrays as a theory extension $\mathcal{T}_{\mathsf{arr}} = \mathcal{T}_0 \cup \mathcal{K}_{\mathsf{arr}}$ that is parametric in its base theory $\mathcal{T}_0$. For this purpose, we assume that the base theory $\mathcal{T}_0$ is over signature $\Sigma_0 = (S_0, \Omega_0)$ with sorts index and elem in S_0, and that $\mathcal{T}_0$ satisfies the assumptions of Corollary 7. Examples of appropriate base theories are the empty theory (in which case Ω_0 contains only equality predicates), the theory of uninterpreted function symbols with equality, and the theory of linear arithmetic, interpreting the sort index as integers.

The theory $\mathcal{T}_{\mathsf{arr}}$ extends $\mathcal{T}_0$ with a fresh sort array and extension symbols $rd : \mathsf{array} \times \mathsf{index} \to \mathsf{elem}$, $wr : \mathsf{array} \times \mathsf{index} \times \mathsf{elem} \to \mathsf{array}$, and $diff : \mathsf{array} \times \mathsf{array} \to \mathsf{index}$. The function symbols rd and wr stand for the usual array selection and update function whose meaning is given by McCarthy's read over write axioms [36]:

$$rd(wr(a, i, e), i) = e, \tag{4}$$

$$i \neq j \Rightarrow rd(wr(a, i, e), j) = rd(a, j) \tag{5}$$

The function *diff* is defined as follows: for any two distinct arrays a and b, the term $\mathit{diff}(a, b)$ denotes an index at which a and b differ. This is formalized by the following axiom:

$$a \neq b \Rightarrow rd(a, \mathit{diff}(a, b)) \neq rd(b, \mathit{diff}(a, b)) \tag{6}$$

Note that this axiom is obtained by skolemizing the extensionality axiom for arrays

$$\forall ab.\, a \neq b \Rightarrow \exists i.\, rd(a, i) \neq rd(b, i)$$

where *diff* is the introduced Skolem function for the existentially quantified variable i. The set of extension axioms $\mathcal{K}_{\mathsf{arr}}$ of our theory of arrays consists of the flattened and linearized versions of the axioms (4), (5), and (6) where a, b, i, j and e are implicitly universally quantified variables. For instance, the linearized and flattened version of axiom (4) is

$$b = wr(a, i, e) \wedge i = j \Rightarrow rd(b, j) = e$$

It is well-known that the standard theory of arrays (i.e., the one given by axioms (4) and (5)) does not admit ground interpolation. We illustrate this through an example due to Ranjit Jhala: consider the ground formulas $A \equiv b = wr(a, i, e)$ and $B \equiv j \neq k \wedge rd(a, j) \neq rd(b, j) \wedge rd(a, k) \neq rd(b, k)$ whose conjunction is unsatisfiable. There exists no ground interpolant for (A, B) that only contains the shared constants a, b and the theory symbols wr and rd. However, as has been observed in [10], such a ground interpolant can be constructed if one includes the difference function *diff* in the theory. An appropriate ground interpolant for (A, B) in the extended theory is given by $b = wr(a, \mathit{diff}(a, b), rd(b, \mathit{diff}(a, b)))$. In fact, the authors of [10] have shown that including the *diff* function is sufficient for ground interpolation. We now give an alternative proof of this result by showing that $\mathcal{T}_{\mathsf{arr}}$ has the partial amalgamation property. This leads to an alternative interpolation procedure for theory $\mathcal{T}_{\mathsf{arr}}$ that can be easily implemented on top of an existing interpolation procedure for the base theory.

In order to define an appropriate amalgamation closure W_{arr}, we need to generalize the example above. That is, we have to define W_{arr} in such a way that there exists no partial W_{arr}-amalgams (M_A, M_B, M_C) with arrays a and b that are shared between M_A and M_B, and M_A, M_B disagree on the number of indices at which a and b differ. To this end, inductively define for any terms a and b of sort array and $k \geq 0$ the term $a \overset{k}{\rightsquigarrow} b$ as follows:

$$a \overset{k}{\rightsquigarrow} b = a \text{ if } k = 0 \text{ and}$$
$$a \overset{k}{\rightsquigarrow} b = wr(a \overset{k-1}{\rightsquigarrow} b, \mathit{diff}(a \overset{k-1}{\rightsquigarrow} b, b), rd(b, \mathit{diff}(a \overset{k-1}{\rightsquigarrow} b, b)))$$

for all $k > 0$. Note that if in some $\mathcal{T}_{\mathsf{arr}}$-model two arrays a and b differ in exactly k positions, then $b = a \overset{k}{\rightsquigarrow} b$. Now define W_{arr} as follows:

$$\begin{aligned} &W_{\mathsf{arr}}(T_A, T_B) = \\ &\quad \text{let } T_0 = \mathsf{st}(T_A \cup \{\, a \overset{k}{\rightsquigarrow} b \mid a, b \in \mathsf{st}(T_A \cap T_B) \,\}) \text{ in} \\ &\quad \text{let } T_1 = \mathsf{st}(T_0 \cup \{\, rd(a, \mathit{diff}(a, b)) \mid a, b \in \mathsf{st}(T_A) \,\}) \text{ in} \\ &\quad T_1 \cup \{\, rd(a, i) \mid a, i \in \mathsf{st}(T_1), rd(b, i) \in T_1 \cup \mathsf{st}(T_B) \,\} \end{aligned}$$

where k is the number of non-shared terms of the form $wr(a, i, e)$ in $T_A \cup T_B$. Note that $W_{\mathsf{arr}}(T_A, T_B)$ can be represented in space that is polynomial in the size of $T_A \cup T_B$. Hence, also $\mathcal{K}_{\mathsf{arr}}[W(A, B)] \cup \mathcal{K}_{\mathsf{arr}}[W(B, A)]$ is polynomial in A, B for finite sets of ground clauses A, B.

Clearly W_{arr} satisfies properties (i), (ii), (iv), and (v) of amalgamation closures. To see that it also satisfies (iii), note that W_{arr} does not increase the number of non-shared terms of the form $wr(a, i, e)$.

Lemma 8. *W_{arr} is an amalgamation closure.*

Theorem 9. *The theory $\mathcal{T}_{\mathsf{arr}} = \mathcal{T}_0 \cup \mathcal{K}_{\mathsf{arr}}$ has the partial amalgamation property with respect to W_{arr}.*

5.2 Theory of Linked Lists with Reachability

Our second example is an extension of Nelson's theory of linked lists with reachability [41], which is also at the core of the LISBQ logic studied in [35]. This theory is useful for reasoning about the correctness of programs that manipulate list-like heap-allocated data structures. We show that neither Nelson's original theory, nor its variation in [35] admit ground interpolation. Using counterexamples to the partial amalgamation property for Nelson's theory, we then systematically develop an extension of the theory, which admits ground interpolation. As a result, we obtain the first complete ground interpolation procedure for a non-trivial theory of linked data structures.

As in the previous example, we define our theory of lists with reachability as a theory extension $\mathcal{T}_{\mathsf{llr}} = \mathcal{T}_0 \cup \mathcal{K}_{\mathsf{llr}}$ that is parametric in its base theory $\mathcal{T}_0$. We require that the base theory is over the signature $\Sigma_0 = (S_0, \Omega_0)$ with a dedicated sort addr in S_0 and that $\mathcal{T}_0$ satisfies the assumptions of Corollary 7. Theory $\mathcal{T}_{\mathsf{llr}}$ extends the base theory with an additional sort field and extension symbols rd, wr, df, jp, lb, and R. The associated sorts are as follows:

$$\begin{aligned} rd &: \mathsf{field} \times \mathsf{addr} \to \mathsf{addr} \\ wr &: \mathsf{field} \times \mathsf{addr} \times \mathsf{addr} \to \mathsf{field} \\ df &: \mathsf{field} \times \mathsf{field} \to \mathsf{addr} \\ jp, lb &: \mathsf{field} \times \mathsf{addr} \times \mathsf{addr} \to \mathsf{addr} \\ R &: \mathsf{field} \times \mathsf{addr} \times \mathsf{addr} \times \mathsf{addr} \to \mathsf{bool} \end{aligned}$$

Before we present the axioms of the theory extension, we define the meaning of the extension symbols in terms of a set of structures $\mathcal{M}_{\mathsf{llr}}$ in which the interpretation of extension symbols is determined by the interpretation of the sorts addr and field. We call the structures in $\mathcal{M}_{\mathsf{llr}}$ *heap models*. In a heap model $M \in \mathcal{M}_{\mathsf{llr}}$, the sort addr represents a set of memory addresses and the sort field a set of address-valued pointer fields. We use a Bornat/Burstall-style memory model [7], i.e., each field is represented as a function from addresses to addresses. The base theory may, e.g., interpret the sort addr as integers to model pointer arithmetic, or it may leave this sort uninterpreted to obtain a more abstract memory model.

In a heap model, the extension symbols rd and wr are interpreted as function application and function update, respectively. For notational convenience, we write $x.f$ and $f[x := y]$ in formulas for terms of the form $rd(f, x)$, respectively, $wr(f, x, y)$. An atom $R(f, a, b, c)$ holds in a heap model M, if there exists a path in the function graph spanned by field f that connects addresses a and b, and the shortest such path does not visit address c. In formulas, we write $x \xrightarrow{f/u} y$ instead of $R(f, x, y, u)$ and we write $x \xrightarrow{f} y$ as a short-hand for $x \xrightarrow{f/y} y$. Note that $a \xrightarrow{f} b$ holds in a heap model M iff a and b are related by the reflexive transitive closure of f. The function symbol jp is interpreted such that $jp(f, a, b)$ denotes the *join point* of addresses a and b, i.e., $jp(f, a, b)$ is the first address on the f-path starting in a that is also on the f-path starting in b, unless these paths are disjoint. In the latter case, we define $jp(f, a, b) = a$. Note that even if the f-paths starting in a and b are not disjoint, we might still have $jp(f, a, b) \neq jp(f, b, a)$ if the two paths form a cycle. In formulas, we write $x \overset{f}{\curlyvee} y$ as a shorthand for $jp(f, x, y)$. The function symbol lb is interpreted such that if b is reachable from a via f, then $lb(f, a, b)$ is the last address before b on the shortest f-path from a to b. The function symbol df is interpreted as in the theory of arrays, i.e., $df(f, g)$ denotes an address for which f and g take different values in case f and g are not the same functions.

$$
\begin{aligned}
&M(rd)(f,a) = f(a)\\
&M(wr)(f,a,b) = f[a \mapsto b]\\
&M(R)(f,a,b,c) = 1 \text{ iff}\\
&\quad (a,b) \in \{\, (d, f(d)) \mid d \in M(\mathsf{addr}) \wedge d \neq c \,\}^*\\
&M(df)(f,g) \in \{\, a \in M(\mathsf{addr}) \mid f(a) \neq g(a) \,\} \text{ if } f \neq g\\
&M(jp)(f,a,b) = c \text{ iff}\\
&\quad (a,c) \in \{\, (d, f(d)) \mid (b,d) \notin f^* \,\}^* \wedge ((b,c) \in f^* \vee a = c)\\
&M(lb)(f,a,f(b)) = b \text{ if } M(R)(f,a,b,f(b)) = 1
\end{aligned}
$$

Figure 5. Restrictions on the interpretation of extension symbols in a heap model M

Formally, for a binary relation P over a set X (respectively, a unary function $P : X \to X$), we denote by P^* the reflexive transitive closure of P. The set of heap models $\mathcal{M}_{\mathsf{llr}}$ is then defined as the set of all structures M such that (i) $M|_{\Sigma_0}$ is a model of $\mathcal{T}_0$, (ii) $M(\mathsf{field})$ is the set of all functions $M(\mathsf{addr}) \to M(\mathsf{addr})$, (iii) the interpretation of the extension symbols in M satisfies the restrictions specified in Figure 5, and (iv) for every $a \in M(\mathsf{addr}), f \in M(\mathsf{field})$, the set $\{\, b \in M(\mathsf{addr}) \mid a \xrightarrow{f} b \vee b \xrightarrow{f} a \,\}$ is finite. Condition (iv) is not strictly necessary, but it provides a more precise characterization of the models that we obtain from partial amalgams of $\mathcal{T}_{\mathsf{llr}}$.

We make the following simplifying assumption, which restricts the set of input formulas that we consider.

Assumption 10. *The set of uninterpreted constants Ω^c contains at most one constant of sort* field.

Assumption 10 means that we will only consider input formulas $A \wedge B$ in which all terms of sort field are related by a finite sequence of field writes. That is, there will be no models of such formulas in which the interpretation of two terms of sort field appearing in $A \wedge B$ differ at more than n addresses, where n is the number of field writes in $A \wedge B$.

The extension axioms $\mathcal{K}_{\mathsf{llr}}$ are the set of clauses that is obtained by computing the conjunctive normal form of the axioms given in Figure 6, and linearizing and flattening the resulting set of clauses. The following lemma states that the resulting theory extension $\mathcal{T}_{\mathsf{llr}}$ is a sound axiomatization of heap models.

Lemma 11. *All heap models are models of $\mathcal{T}_{\mathsf{llr}}$.*

As we shall see later, the extension axioms are also sufficient to fully characterize heap models, i.e., every ground formula that is satisfiable modulo $\mathcal{T}_{\mathsf{llr}}$ is also satisfiable in some heap model. However, let us first explain why the theory without the function jp does not have ground interpolation. To this end, consider the situation illustrated in Figure 7. The graphs M_A, M_B, and M_C depict (partial) heap models where the dashed edges denote binary reachability $x \xrightarrow{f} y$. Transitive and reflexive edges are omitted for readability. The structures M_A and M_B are almost identical. They only differ in the order in which the list segments starting in c_1 and c_2 join the list segment starting in c_0. In M_A the segment of c_1 joins before the one of c_2 and in M_B it is the other way around. We express M_A and M_B in terms of formulas A and B as follows:

$$
\begin{aligned}
A \equiv\ & c_0 \xrightarrow{f/c_1} a_0 \wedge c_1 \xrightarrow{f/c_0} a_0 \wedge a_0 \xrightarrow{f} a_1 \wedge c_2 \xrightarrow{f/a_0} a_1 \wedge\\
& a_1 \xrightarrow{f} c_3 \wedge a_0 \neq a_1\\
B \equiv\ & c_0 \xrightarrow{f/c_2} b_0 \wedge c_2 \xrightarrow{f/c_0} b_0 \wedge b_0 \xrightarrow{f} b_1 \wedge c_1 \xrightarrow{f/b_0} b_1 \wedge\\
& b_1 \xrightarrow{f} c_3 \wedge b_0 \neq b_1
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{Reflexive}\ & x \xrightarrow{f/u} x\\
\mathsf{Step}\ & x \xrightarrow{f/u} x.f \vee x = u\\
\mathsf{SelfLoop}\ & x.f = x \wedge x \xrightarrow{f} y \Rightarrow x = y\\
\mathsf{Sandwich}\ & x \xrightarrow{f/x} y \Rightarrow x = y\\
\mathsf{Reach}\ & x \xrightarrow{f/u} y \Rightarrow x \xrightarrow{f} y\\
\mathsf{Linear1}\ & x \xrightarrow{f} y \Rightarrow x \xrightarrow{f/y} u \vee x \xrightarrow{f/u} y\\
\mathsf{Linear2}\ & x \xrightarrow{f/u} y \wedge x \xrightarrow{f/v} z \Rightarrow\\
& \quad x \xrightarrow{f/u} z \wedge z \xrightarrow{f/u} y \vee x \xrightarrow{f/v} y \wedge y \xrightarrow{f/v} z\\
\mathsf{Transitive1}\ & x \xrightarrow{f/u} y \wedge y \xrightarrow{f/u} z \Rightarrow x \xrightarrow{f/u} z\\
\mathsf{Transitive2}\ & x \xrightarrow{f/z} y \wedge y \xrightarrow{f/z} u \wedge y \xrightarrow{f} z \Rightarrow x \xrightarrow{f/u} y\\
\mathsf{Join1}\ & x \xrightarrow{f} (x \overset{f}{\curlyvee} y)\\
\mathsf{Join2}\ & x \xrightarrow{f} z \wedge y \xrightarrow{f} z \Rightarrow y \xrightarrow{f} (x \overset{f}{\curlyvee} y)\\
\mathsf{Join3}\ & x \xrightarrow{f} z \wedge y \xrightarrow{f} z \Rightarrow x \xrightarrow{f/z} (x \overset{f}{\curlyvee} y)\\
\mathsf{Join4}\ & y \xrightarrow{f} (x \overset{f}{\curlyvee} y) \vee (x \overset{f}{\curlyvee} y) = x\\
\mathsf{LastBefore}\ & x \xrightarrow{f/y.f} y \Rightarrow lb(f,x,y.f) = y\\
\mathsf{ReachWrite}\ & x \xrightarrow{f[u:=v]/w} y \Leftrightarrow\\
& \quad x \xrightarrow{f/w} y \wedge x \xrightarrow{f/u} y \vee\\
& \quad x \xrightarrow{f/w} u \wedge v \xrightarrow{f/w} y \wedge v \xrightarrow{f/u} y \wedge u \neq w\\
\mathsf{ReadWrite1}\ & x.(f[x := y]) = y\\
\mathsf{ReadWrite2}\ & x \neq y \Rightarrow y.(f[x := z]) = y.f\\
\mathsf{Diff}\ & f \neq g \Rightarrow df(f,g).f \neq df(f,g).g
\end{aligned}
$$

Figure 6. Axioms of theory extension $\mathcal{T}_{\mathsf{llr}}$

The conjunction $A \wedge B$ is unsatisfiable because A and B do not agree on the order of the join points of the list segments. An appropriate ground interpolant for $A \wedge B$ is given by

$$\neg(c_0 \overset{f}{\curlyvee} c_2) \xrightarrow{f} (c_0 \overset{f}{\curlyvee} c_1)$$

All other ground interpolants for $A \wedge B$ also rely on the join function. Hence, if we drop the join function from the theory $\mathcal{T}_{\mathsf{llr}}$, we lose ground interpolation. This is also reflected in a violation of the amalgamation property. If we drop joins, the models M_A and M_B have a common substructure M_C, which we also depict in Figure 7. There is no model M_D in which both M_A and M_B can be embedded while preserving the common substructure M_C.

The function lb plays a similar role than jp in that it is also needed for the completeness of ground interpolation. A corresponding counterexample to partial amalgamation can be found in the extended version of this paper [48]. We omit further details here because the function lb appears to be of less practical importance than the join function.

Although the full theory $\mathcal{T}_{\mathsf{llr}}$ admits ground interpolation, it is still difficult to devise an amalgamation closure W_{llr} that allows us to prove partial amalgamation for this theory. We illustrate this using the unsatisfiable formula (3) in Section 2.2. One ground interpolant for $A \wedge B$ is $c.f \xrightarrow{f} c$, which expresses that c lies on

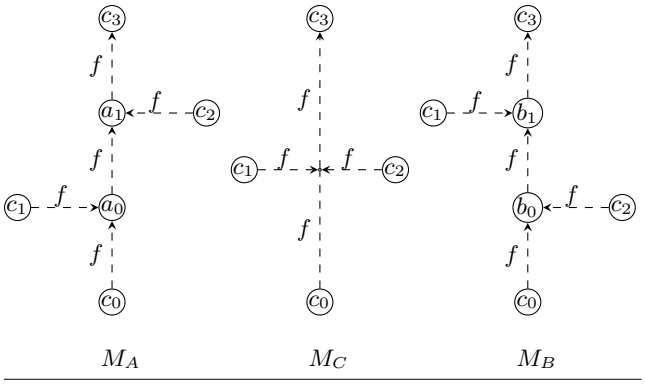

Figure 7. An amalgam (M_A, M_C, M_B) of the theory of linked lists without join that witnesses a violation of amalgamation

a cycle. In order to prove the partial amalgamation property for theory $\mathcal{T}_{\text{llr}}$, we have to ensure that the shared substructure of every partial amalgam already contains all information about which of the shared terms c lie on a cycle. Since we can express this using the formula $c.f \xrightarrow{f} c$ we may attempt to define the amalgamation closure W_{llr} such that for every shared field f and every shared address c defined in the common substructure, also $c.f$ is defined. However, since $c.f$ is again shared, such an operator W_{llr} would inevitably fail to satisfy condition (iii) of amalgamation closures.

We can avoid this problem by further extending $\mathcal{T}_{\text{llr}}$ with an additional predicate symbol $Cy : \mathsf{field} \times \mathsf{addr} \to \mathsf{bool}$ such that $Cy(f, c)$ holds iff $c.f \xrightarrow{f} c$. The corresponding extension axioms defining Cy are as follows

$$\mathsf{Cycle1}\ x \xrightarrow{f} y \wedge y \xrightarrow{f} x \Rightarrow Cy(f, x) \vee x = y$$

$$\mathsf{Cycle2}\ Cy(f, x) \wedge x \xrightarrow{f} y \Rightarrow y \xrightarrow{f} x$$

Note that in these axioms we do not use $c.f$ to define $Cy(f, c)$, which avoids the problem with the definition of the amalgamation closure mentioned above. We denote by $\mathcal{K}_{\text{llrc}}$ the set of extension axioms obtained by adding these axioms to $\mathcal{K}_{\text{llr}}$ and we denote by $\mathcal{T}_{\text{llrc}}$ the resulting theory extension $\mathcal{T}_{\text{llrc}} = \mathcal{T}_0 \cup \mathcal{K}_{\text{llrc}}$.

To define the amalgamation closure for $\mathcal{T}_{\text{llrc}}$, we first define the terms $f \overset{k}{\rightsquigarrow} g$ for all terms f, g of sort field and $k \geq 0$, as in the case of the array theory, except that we replace array reads by field reads and array writes by field writes. The amalgamation closure for $\mathcal{T}_{\text{llrc}}$ is then defined as follows:

$$\begin{aligned}
&W_{\text{llrc}}(T_A, T_B) = \\
&\quad \text{let } T_0 = \mathsf{st}(T_A \cup \{\, f \overset{k}{\rightsquigarrow} g \mid f, g \in \mathsf{st}(T_A \cap T_B) \,\}) \text{ in} \\
&\quad \text{let } T_1 = \mathsf{st}(T_0 \cup \{\, df(f, g).f \mid f, g \in \mathsf{st}(T_A) \,\}) \text{ in} \\
&\quad \text{let } T_2 = T_1 \cup \{\, a.f \mid f, a \in \mathsf{st}(T_1), a.g \in T_1 \cup \mathsf{st}(T_B) \,\} \text{ in} \\
&\quad \text{let } T_3 = T_2 \cup \{\, lb^1(f, a, b.f) \mid a, b.f \in T_2 \,\} \text{ in} \\
&\quad \text{let } T_4 = T_3 \cup \{\, (a \overset{f}{\curlyvee} b)^1 \mid a, b, f \in T_3 \text{ shared with } T_B \,\} \text{ in} \\
&\quad \text{let } T_5 = T_4 \cup \{\, a \xrightarrow{f/c} b \mid a, b, c, f \in T_4 \,\} \text{ in} \\
&\quad \text{let } T_6 = T_5 \cup \{\, Cy(f, a) \mid f, a \in T_5 \text{ shared with } T_B \,\} \text{ in} \\
&\quad T_6
\end{aligned}$$

where k is the number of non-shared terms of the form $f[a := b]$ in $T_A \cup T_B$, $lb^1(f, a, b.f)$ denotes $lb(f, a, b.f)$ if lb does not appear in a or else it denotes a, and similarly $(a \overset{f}{\curlyvee} b)^1$ denotes $a \overset{f}{\curlyvee} b$ if no join appears in a and b, or else a. The intermediate set of terms T_2 in the definition of $W_{\text{llrc}}(T_A, T_B)$ is similar to the set of terms computed by W_{arr} for the theory of arrays. The set of terms T_4 ensures that the joins of all shared terms are defined. Note that in models of $\mathcal{T}_{\text{llrc}}$ we have for all a, b, c and f, $(a \overset{f}{\curlyvee} b) \overset{f}{\curlyvee} c = a \overset{f}{\curlyvee} c$ or $(a \overset{f}{\curlyvee} b) \overset{f}{\curlyvee} c = b \overset{f}{\curlyvee} c$, and similarly $a \overset{f}{\curlyvee} (b \overset{f}{\curlyvee} c) = a \overset{f}{\curlyvee} b$ or $a \overset{f}{\curlyvee} (b \overset{f}{\curlyvee} c) = a \overset{f}{\curlyvee} c$. Because of this property and Assumption 10, we can avoid the construction of terms with nested occurrences of joins. The set T_5 ensures that the reachability predicate is fully defined in all partial models. Finally, T_6 ensures that the predicate Cy is defined for all shared terms.

Lemma 12. *W_{llrc} is an amalgamation closure.*

We can now prove that theory $\mathcal{T}_{\text{llrc}}$ has the partial amalgamation property. In fact, we can prove a slightly stronger statement:

Lemma 13. *For every partial W_{llrc}-amalgam (M_A, M_B, M_C) of $\mathcal{T}_{\text{llrc}}$ there exists a total model M_D of $\mathcal{T}_{\text{llrc}}$, and weak embeddings $h_A : M_A \to M_D$ and $h_B : M_B \to M_D$ such that $h_A|_{[M_C]} = h_B|_{[M_C]}$. Moreover, $M_D|_{\Sigma_{\text{llr}}}$ is isomorphic to a heap model.*

Theorem 14. *The theory $\mathcal{T}_{\text{llrc}} = \mathcal{T}_0 \cup \mathcal{K}_{\text{llrc}}$ has the partial amalgamation property with respect to W_{llrc}.*

Theorem 14 does not just give us a complete ground interpolation procedure for $\mathcal{T}_{\text{llrc}}$, but also for $\mathcal{T}_{\text{llr}}$. This is because we can always rewrite atoms of the form $Cy(f, c)$ in interpolants for theory $\mathcal{T}_{\text{llrc}}$ into atoms $c.f \xrightarrow{f} c$, obtaining an interpolant for $\mathcal{T}_{\text{llr}}$.

Corollary 15. *Theory $\mathcal{T}_{\text{llr}}$ admits ground interpolation.*

Finally, we can show that satisfiability of ground formulas modulo $\mathcal{T}_{\text{llr}}$ is equivalent to satisfiability of ground formulas modulo heap models. This is a consequence of Lemma 13 and Lemma 11.

Theorem 16. *Let G be a finite set of ground Σ^c_{llr}-clauses. Then $\mathcal{T}_{\text{llr}} \cup G \models \bot$ iff $M \not\models G$ for all $M \in \mathcal{M}_{\text{llr}}$.*

Note that the number of terms in $W_{\text{llrc}}(T_A, T_B)$ is polynomial in the size of $T_A \cup T_B$ and, hence, so is the number of generated instances of extension axioms. Together with Theorem 14 this implies that satisfiability of ground formulas modulo $\mathcal{T}_{\text{llr}}$ is decidable in NP, provided $\mathcal{T}_0 \cup \mathcal{T}_{\mathsf{EUF}}$ is also decidable in NP.

6. Combining Interpolation and Abstraction

We now turn towards more practical concerns. A common problem in interpolation-based algorithms is that interpolants are not unique. Often the interpolation procedure produces interpolants that are not useful for a specific application. This may cause, e.g., that the refinement loop of a software model checker diverges because the generated interpolants do not sufficiently abstract from the infeasible error traces.

To illustrate this problem, consider the program shown in Figure 8. The function concat takes two lists x and y as input and concatenates them by first traversing x and then swinging the pointer of the last node of x to y. A second while loop then traverses x again to check whether y is reachable after the concatenation. Suppose we want to use interpolation-based verification to check that the assert statement in concat never fails. The right-hand side of Figure 8 shows the trace formula for a spurious error trace of concat, which is obtained by unrolling both loops twice. The trace formula is unsatisfiable, hence we can compute an interpolant for the indicated choice of $A \wedge B$. One valid interpolant is as follows:

$$\mathsf{x.n_1.n_1} \neq \mathsf{null} \vee \mathsf{y} = \mathsf{null}$$

This interpolant is rather useless for obtaining an inductive invariant that allows us to prove our verification goal. The interpolant only rules out the one given error trace and fails to abstract from its specifics, i.e., the length of the traversed list. We would rather like to obtain the alternative interpolant $\mathsf{x} \xrightarrow{\mathsf{n_1}} \mathsf{y}$, which is the inductive loop invariant we are seeking. But how can we ensure that the

```
void concat (List x, List y) {
  List curr, prev;
  prev = null;
  curr = x;
  while (curr != null) {
    prev = curr;
    curr = curr.n;
  }
  if (prev == null) x = y;
  else prev.n = y;

  curr = x;
  while (curr != null &&
         curr != y) {
    curr = curr.n
  }
  assert (curr == y);
}
```

$$
\begin{array}{ll}
A\ \{ & \mathsf{prev}_0 = \mathsf{null} \ \wedge \\
& \mathsf{curr}_0 = \mathsf{x} \ \wedge \\
& \mathsf{curr}_0 \neq \mathsf{null} \ \wedge \\
& \mathsf{prev}_1 = curr_0 \ \wedge \\
& \mathsf{curr}_1 = \mathsf{curr}_0.\mathsf{n}_0 \ \wedge \\
& \mathsf{curr}_1 = \mathsf{null} \ \wedge \\
& \mathsf{prev}_2 = \mathsf{curr}_1 \ \wedge \\
& \mathsf{curr}_2 = \mathsf{curr}_1.\mathsf{n}_0 \ \wedge \\
& \mathsf{curr}_2 = \mathsf{null} \ \wedge \\
& \mathsf{prev}_2 \neq \mathsf{null} \ \wedge \\
& \mathsf{n}_1 = \mathsf{n}_0[\mathsf{prev}_2 := y] \ \wedge \\
B\ \{ & \mathsf{curr}_3 = \mathsf{x} \ \wedge \\
& \mathsf{curr}_3 \neq \mathsf{null} \ \wedge \\
& \mathsf{curr}_3 \neq \mathsf{y} \ \wedge \\
& \mathsf{curr}_4 = \mathsf{curr}_3.\mathsf{n}_1 \ \wedge \\
& \mathsf{curr}_4 \neq \mathsf{null} \ \wedge \\
& \mathsf{curr}_4 \neq \mathsf{y} \ \wedge \\
& \mathsf{curr}_5 = \mathsf{curr}_4.\mathsf{n}_1 \ \wedge \\
& (\mathsf{curr}_5 = \mathsf{null} \vee \\
& \ \mathsf{curr}_5 = \mathsf{y}) \ \wedge \\
& \mathsf{curr}_5 \neq \mathsf{y}
\end{array}
$$

Figure 8. C code for concatenation of two lists. The second while loop checks whether y is reachable from x after the concatenation. The right-hand side shows the trace formula for an infeasible error trace that is obtained by unfolding both while loops twice.

proc InterpolateWithAbstraction
  input
    $\mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$: theory extension
    $\mathsf{Interpolate}_0$: ground interpolation procedure for $\mathcal{T}_0 \cup \mathcal{T}_{\mathsf{EUF}}$
    $\mathsf{GetModel}_0$: model generation procedure for $\mathcal{T}_0 \cup \mathcal{T}_{\mathsf{EUF}}$
    W : amalgamation closure for $\mathcal{T}_1$
    α : abstraction function for weak partial models $\mathsf{PMod}(\mathcal{T}_1)$
    A, B : sets of ground Σ_1^c-clauses with $A \cup B \models_{\mathcal{T}_1} \bot$
  begin
    $I := \bot$
    while $\neg I \wedge A \wedge \mathcal{K}[W(A,B)] \not\models_{\mathcal{T}_0 \cup \mathcal{T}_{\mathsf{EUF}}} \bot$ **do**
      $M_A := \mathsf{GetModel}_0(\neg I \wedge A \wedge \mathcal{K}[W(A,B)])$
      $A_0 := \alpha(M_A) \cup \mathcal{K}[W(\alpha(M_A), B)]$
      $B_0 := B \cup \mathcal{K}[W(B, \alpha(M_A))]$
      if $A_0 \cup B_0 \not\models_{\mathcal{T}_0 \cup \mathcal{T}_{\mathsf{EUF}}} \bot$ **then**
        $A_0 := M_A \cup \mathcal{K}[W(M_A, B)]$
        $B_0 := B \cup \mathcal{K}[W(B, M_A)]$
      $I := I \vee \mathsf{Interpolate}_0(A_0, B_0)$
    return I
  end

Figure 9. Instantiation-based interpolation procedure with user-defined abstraction of partial models.

interpolation procedure finds the right interpolant? What accounts for a "good" interpolant often depends on the concrete application. It is therefore difficult to devise generic strategies that can be hard-wired into the interpolation procedure.

Our approach enables the user of the interpolation procedure to inject domain-specific knowledge that helps to guide the proof search and improves the quality of the produced interpolants. This can be done as follows. Note that a partial model M can be thought of as a symbolic representation of a set of total models, namely the set of all models into which M weakly embeds. In fact, for many local theory extensions we can represent a ground formula A as a finite set of partial models each of which is represented as a finite set of ground unit clauses. We can then guide the interpolation procedure as follows: instead of interpolating $A \wedge B$ directly, we enumerate partial models M_A of A, which we interpolate one by one with B. However, before we compute the interpolant of $M_A \wedge B$ for a given partial model M_A, we first apply a user-defined abstraction function α that *widens* M_A by dropping certain clauses. For instance, in the case of the theory of linked-lists we may want to drop all clauses containing the function symbol for field dereference from partial models and only keep information about reachability and joins.

This idea of combining interpolation and abstraction is realized in procedure InterpolateWithAbstraction given in Figure 9, which refines our earlier procedure in Figure 4. Procedure InterpolateWithAbstraction takes as additional arguments the user-defined abstraction function α and a procedure $\mathsf{GetModel}_0$ that is able to generate partial models for satisfiable formulas in the theory $\mathcal{T}_0 \cup \mathcal{T}_{\mathsf{EUF}}$. The while loop enumerates the partial models of A. In each iteration, we only consider partial models that are not yet subsumed by the already computed interpolants I. This ensures that only few partial models have to be considered in practice. The conditional statement in the body of the while loop guarantees that the interpolation procedure falls back to the full partial model M_A in the cases where the abstraction function is too coarse and does not preserve unsatisfiability of $M_A \wedge B$.

Procedure InterpolateWithAbstraction enables us to easily incorporate domain-specific abstraction and widening techniques into the interpolation procedure while still treating the underlying interpolation procedure for the base theory as a black box. The ability of modern SMT solvers to generate models for satisfiable formulas makes it easy to implement this approach.

7. Implementation and Evaluation

We have implemented our framework in a prototype tool and instantiated it for the theory of linked lists with reachability that we presented in Section 5.2. The prototype is written in OCaml. It implements a variation of the algorithm presented in Figure 9. We use the SMT solver Z3 v.4.0 [14] for the generation of partial models and MathSAT 5 [22] for interpolation of formulas in the base theory. Communication with both provers is done via their SMT-LIB 2 [4] interfaces. The difference between the implementation and the vanilla algorithm in Figure 9 is that we use the incremental solving capability of the SMT solver to get a more fine-grained abstraction of partial models. Instead of falling immediately back to the full partial model M_A whenever the abstract partial model is too weak to prove unsatisfiability, we sort the clauses of M_A in a particular order and then push them one by one on the assertion stack of the solver. Each time we push a new clause onto the stack, we check whether the conjunction of the stack with B is still satisfiable. Once it becomes unsatisfiable, we compute the interpolant. The order in which the clauses of the partial model are pushed is determined by a weight associated with each clause, where lighter clauses are considered first. The weight function encodes the domain-specific knowledge that guides the proof search of the solver. In our implementation we have chosen the weight function such that highest weight is given to clauses that contain dereferences of pointer fields. This makes it more likely that the computed interpolants abstract from the length of the lists.

The number of generated instances of extension axioms is polynomial in the size of the input formulas, which means that the underlying decision problem remains in NP. However, in practice the eager instantiation approach can still be a performance bottle neck with thousands of clauses generated even for small examples. We therefore implemented several optimizations to reduce the number

benchmark	#unroll.	$\#M_A$	#inst.	time (s)
reverse	3	1	5,003	0.4
concat	2-2	1	9,953	0.6
delete	2	2	27,934	2.3
insertBefore	2	2	28,176	2.3
splice	2	4	135,446	12.1

Table 1. Summary of experiments. The columns list the benchmark name, the number of loop unrollings in the error trace, the number of generated partial models for A, the number of generated instances of extension axioms, and the total computation time.

of instances of axioms that we generate. First, we compute the congruence closure for the terms appearing in the original input formula and then only instantiate axioms by selecting one representative term per congruence class. We have found this to work particularly well for trace formulas, which typically contain many equalities. Second, we use a more lazy instantiation approach where we first compute a subset of the terms in the amalgamation closure that is likely to be sufficient for proving unsatisfiability. For example, in our experiments we never needed to generate instances with terms that contain the df function. Apart from these obvious optimizations, our instance generation procedure is still rather naive.

To evaluate the feasibility of our approach, we have used our implementation to automatically infer loop invariants for verifying properties of simple list-manipulating programs. In particular, we checked functional correctness properties and whether certain shape invariants are preserved. For this purpose, we manually generated spurious error traces from the considered programs by unrolling their loops a few times. Our prototype accepts such error traces as input and converts them into trace formulas which are then interpolated. Our experiments where conducted on a Linux laptop with a dual-core processor and 4GB RAM. Table 1 shows the summary of our experiments. In all cases, the obtained interpolant was an inductive loop invariant of the program and strong enough to prove the program correct. Roughly 40% of the running time is spent on I/O with the provers. However, the main bottle neck of the implementation is the eager instantiation of extension axioms. We believe that the running times can be significantly improved by using more sophisticated model-driven instantiation approaches such as [20, 29], which instantiate axioms incrementally.

8. Related Work

Our notion of partial amalgamation is closely related to the (strong) amalgamation property [32], whose role in ground interpolation for disjoint theory combinations has been recently studied [11]. Our use of amalgamation properties is orthogonal to [11], as we consider (non-disjoint) theory extensions rather than disjoint theory combinations. In a sense, partial amalgamation is the adaptation of the weak embedability condition in [46] to the case of interpolation. Our approach can thus be thought of as the symbioses of the two orthogonal approaches described in [46, 47] and [11]. Note that neither of the interpolation techniques presented in [47] and [11] can be applied directly to the theory of lists considered in this paper. The approach in [47] is restricted to extension axioms of a very specific syntactic form: Horn clauses in which all predicate symbols are binary and where additional guard constraints on the quantified variables apply. All three restrictions are violated by the axioms of the list theory. The approach in [11] could be used, in principle, to obtain an interpolation procedure for the combination of a theory of lists with uninterpreted heap nodes and, e.g., the theory of linear integer arithmetic (for interpreting heap nodes as addresses). However, the technique in [11] assumes that interpolation procedures for the component theories already exist. There is no interpolation procedure for the list theory component to start with. Hence, the combination technique of [11] cannot be applied. Other reduction-based approaches to interpolation that are less closely related include [33], which is based on quantifier elimination.

Ground interpolation procedures for specific theories have been developed, e.g., for linear arithmetic over reals [23, 38] and integers [8, 9], uninterpreted functions with equality [19, 38, 50], functional lists [50], as well as, combinations of these theories [11, 21, 38, 50]. These are the procedures that our approach builds on. We discussed two specific theories for which ground interpolation reduces to these existing procedures: the theory of arrays with difference functions [10], and the theory of linked-lists with reachability [35, 41] extended with join. We believe that our approach applies to many other theory extensions that are of importance in program verification, such as our theory of imperative trees [49].

Interpolation approaches that use resolution-based automated theorem provers have been studied, e.g., in [26, 40]. Unlike our approach, these methods target undecidable fragments of first-order logic and infer quantified interpolants. Sometimes, such quantified interpolants are needed to obtain inductive invariants. We can use our approach to infer quantified interpolants by applying techniques explored in [1]. One interesting observation is that these quantified interpolants themselves often constitute local theory extensions and can therefore be treated systematically by our framework, if they become part of subsequent interpolation problems. To out knowledge, McMillan's [40] interpolating version of the theorem prover SPASS, is the only other interpolation-based system that has been used to infer shape invariants of heap-allocated data structures. Unlike our theory of linked-lists, McMillan's axiomatization of reachability predicates is incomplete.

Recent works have explored techniques to influence the quality of computed interpolants, e.g., by reducing the size of unsatisfiability proofs from which interpolants are generated [27], restricting the language in which interpolants can be expressed [1, 30], or by controlling the interpolant strength [16]. Our technique of guiding the proof search of the interpolation procedure through user-defined abstraction functions is orthogonal to these approaches. In spirit, it is most closely related to [30]. The key difference is that we do not need to modify the underlying interpolation procedure, which would contradict our modular approach to interpolation. The idea of using the ability of SMT solvers to generate models for abstraction has been previously explored, e.g., in [43]. Whether the approach of combining interpolation and abstraction can be explained more concisely in terms of abstract interpretation, e.g., in the spirit of [12], remains a question for future research.

9. Conclusion

We have presented a new instantiation-based interpolation framework that enables the modular construction of ground interpolation procedures for application-specific theories. We introduced the semantic notion of partial amalgamation to systematically identify and construct theories for which our framework yields complete interpolation procedures. We gave examples of both new and existing theories to which our framework applies. Using a prototype implementation we demonstrated that our framework enables new applications of interpolation-based algorithms in program verification. Therefore, we see this work as a starting point for a new line of research that studies efficient instantiation-based interpolation procedures for applications in program verification.

Acknowledgments

This work was supported in part by the European Research Council (ERC) Advanced Investigator Grant QUAREM and by the Austrian Science Fund (FWF) project S11402-N23.

References

[1] F. Alberti, R. Bruttomesso, S. Ghilardi, S. Ranise, and N. Sharygina. Lazy abstraction with interpolants for arrays. In *LPAR*, volume 7180 of *LNCS*, pages 46–61. Springer, 2012.

[2] P. Bacsich. Amalgamation properties and interpolation theorems for equational theories. *Algebra Universalis*, 5:45–55, 1975.

[3] M. Barnett and K. R. M. Leino. To goto where no statement has gone before. In *VSTTE*, volume 6217 of *LNCS*, pages 157–168, 2010.

[4] C. Barrett, A. Stump, and C. Tinelli. The SMT-LIB Standard: Version 2.0, 2010.

[5] D. Beyer, T. A. Henzinger, and G. Théoduloz. Lazy shape analysis. In *CAV*, volume 4144 of *LNCS*, pages 532–546. Springer, 2006.

[6] D. Beyer, D. Zufferey, and R. Majumdar. CSIsat: Interpolation for LA+EUF. In *CAV*, volume 5123 of *LNCS*, pages 304–308, 2008.

[7] R. Bornat. Proving Pointer Programs in Hoare Logic. In *MPC*, volume 1837 of *LNCS*, pages 102–126. Springer, 2000.

[8] A. Brillout, D. Kroening, P. Rümmer, and T. Wahl. An Interpolating Sequent Calculus for Quantifier-Free Presburger Arithmetic. *J. Autom. Reasoning*, 47(4):341–367, 2011.

[9] A. Brillout, D. Kroening, P. Rümmer, and T. Wahl. Beyond Quantifier-Free Interpolation in Extensions of Presburger Arithmetic. In *VMCAI*, volume 6538 of *LNCS*, pages 88–102. Springer, 2011.

[10] R. Bruttomesso, S. Ghilardi, and S. Ranise. Rewriting-based quantifier-free interpolation for a theory of arrays. In *RTA*, volume 10 of *LIPIcs*, pages 171–186, 2011.

[11] R. Bruttomesso, S. Ghilardi, and S. Ranise. From strong amalgamability to modularity of quantifier-free interpolation. In *IJCAR*, volume 7364 of *LNCS*, pages 118–133. Springer, 2012.

[12] P. Cousot, R. Cousot, and L. Mauborgne. The reduced product of abstract domains and the combination of decision procedures. In *FOSSACS*, volume 6604 of *LNCS*, pages 456–472. Springer, 2011.

[13] W. Craig. Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. *The Journal of Symbolic Logic*, 22(3):269–285, 1957.

[14] L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *TACAS*, pages 337–340, 2008.

[15] K. Dräger, A. Kupriyanov, B. Finkbeiner, and H. Wehrheim. SLAB: A Certifying Model Checker for Infinite-State Concurrent Systems. In *TACAS*, volume 6015 of *LNCS*, pages 271–274. Springer, 2010.

[16] V. D'Silva, D. Kroening, M. Purandare, and G. Weissenbacher. Interpolant strength. In *VMCAI*, volume 5944 of *LNCS*, pages 129–145. Springer, 2010.

[17] E. Ermis, M. Schäf, and T. Wies. Error invariants. In *FM*, volume 7436 of *LNCS*, pages 187–201. Springer, 2012.

[18] J.-C. Filliâtre and C. Marché. The Why/Krakatoa/Caduceus Platform for Deductive Program Verification. In *CAV*, volume 4590 of *LNCS*, pages 173–177. Springer, 2007.

[19] A. Fuchs, A. Goel, J. Grundy, S. Krstic, and C. Tinelli. Ground interpolation for the theory of equality. In *TACAS*, volume 5505 of *LNCS*, pages 413–427. Springer, 2009.

[20] Y. Ge and L. M. de Moura. Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In *CAV*, volume 5643 of *LNCS*, pages 306–320. Springer, 2009.

[21] A. Goel, S. Krstic, and C. Tinelli. Ground interpolation for combined theories. In *CADE*, volume 5663 of *Lecture Notes in Computer Science*, pages 183–198. Springer, 2009.

[22] A. Griggio. A Practical Approach to Satisfiability Modulo Linear Integer Arithmetic. *JSAT*, 8:1–27, January 2012.

[23] A. Griggio, T. T. H. Le, and R. Sebastiani. Efficient interpolant generation in satisfiability modulo linear integer arithmetic. In *TACAS*, volume 6605 of *LNCS*, pages 143–157. Springer, 2011.

[24] M. Heizmann, J. Hoenicke, and A. Podelski. Nested interpolants. In *POPL*, pages 471–482. ACM, 2010.

[25] T. A. Henzinger, R. Jhala, R. Majumdar, and K. L. McMillan. Abstractions from proofs. In *31st POPL*, 2004.

[26] K. Hoder, L. Kovács, and A. Voronkov. Interpolation and symbol elimination in vampire. In *IJCAR*, volume 6173 of *LNCS*, pages 188–195. Springer, 2010.

[27] K. Hoder, L. Kovács, and A. Voronkov. Playing in the grey area of proofs. In *POPL*, pages 259–272. ACM, 2012.

[28] C. Ihlemann, S. Jacobs, and V. Sofronie-Stokkermans. On local reasoning in verification. In *TACAS*, pages 265–281, 2008.

[29] S. Jacobs. Incremental instance generation in local reasoning. In *CAV*, volume 5643 of *LNCS*, pages 368–382. Springer, 2009.

[30] R. Jhala and K. L. McMillan. A practical and complete approach to predicate refinement. In *TACAS*, volume 3920 of *LNCS*, pages 459–473. Springer, 2006.

[31] R. Jhala and K. L. McMillan. Interpolant-based transition relation approximation. *Logical Methods in Computer Science*, 3(4), 2007.

[32] B. Jónsson. Universal relational systems. *Math. Scand.*, 4:193–208, 1956.

[33] D. Kapur, R. Majumdar, and C. G. Zarba. Interpolation for data structures. In *SIGSOFT FSE*, pages 105–116. ACM, 2006.

[34] D. Kroening and G. Weissenbacher. Interpolation-Based Software Verification with Wolverine. In *CAV*, volume 6806 of *LNCS*, pages 573–578. Springer, 2011.

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

[36] J. McCarthy. Towards a mathematical science of computation. In *IFIP Congress*, pages 21–28, 1962.

[37] K. L. McMillan. Interpolation and SAT-Based Model Checking. In *CAV*, volume 2725 of *LNCS*, pages 1–13. Springer, 2003.

[38] K. L. McMillan. An interpolating theorem prover. *Theor. Comput. Sci.*, 345(1):101–121, 2005.

[39] K. L. McMillan. Lazy abstraction with interpolants. In *CAV*, volume 4144 of *LNCS*, pages 123–136. Springer, 2006.

[40] K. L. McMillan. Quantified invariant generation using an interpolating saturation prover. In *TACAS*, volume 4963 of *LNCS*, pages 413–427. Springer, 2008.

[41] G. Nelson. Verifying reachability invariants of linked structures. In *POPL*, pages 38–47. ACM, 1983.

[42] A. Podelski and T. Wies. Counterexample-guided focus. In *POPL*, pages 249–260. ACM, 2010.

[43] T. W. Reps, S. Sagiv, and G. Yorsh. Symbolic implementation of the best transformer. In *VMCAI*, volume 2937 of *LNCS*, pages 252–266. Springer, 2004.

[44] A. Rybalchenko and V. Sofronie-Stokkermans. Constraint solving for interpolation. In *VMCAI*, volume 4349 of *LNCS*, pages 346–362. Springer, 2007.

[45] M. Sagiv, T. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. *ACM TOPLAS*, 24(3):217–298, 2002.

[46] V. Sofronie-Stokkermans. Hierarchic reasoning in local theory extensions. In *CADE*, pages 219–234, 2005.

[47] V. Sofronie-Stokkermans. Interpolation in local theory extensions. *Logical Methods in Computer Science*, 4(4), 2008.

[48] N. Totla and T. Wies. Complete instantiation-based interpolation. Technical Report TR2012-950, New York University, 2012.

[49] T. Wies, M. Muñiz, and V. Kuncak. An efficient decision procedure for imperative tree data structures. In *CADE*, volume 6803 of *LNCS*, pages 476–491. Springer, 2011.

[50] G. Yorsh and M. Musuvathi. A combination method for generating interpolants. In *CADE*, volume 3632 of *LNCS*, pages 353–368, 2005.