Noname manuscript No.
(will be inserted by the editor)

On Recursion-free Horn Clauses and Craig Interpolation

Philipp Rümmer · Hossein Hojjat · Viktor Kuncak

Received: date / Accepted: date

Abstract One of the main challenges in software verification is efficient and precise analysis of programs with procedures and loops. Interpolation methods remain among the most promising techniques for such verification. To accommodate the demands of various programming language features, over the past years several extended forms of interpolation have been introduced. We give a precise ontology of such extended interpolation methods, and investigate the relationship between interpolation and fragments of constrained recursion-free Horn clauses. We also introduce a new notion of interpolation, disjunctive interpolation, which solves a more general class of problems in one step compared to previous notions of interpolants, such as tree interpolants or inductive sequences of interpolants. We present algorithms and complexity for construction of interpolants, as well as the corresponding decision problems for recursion-free Horn fragments. Finally, we give an extensive empirical evaluation using a solver for (recursive) Horn problems, in particular comparing the performance of tree interpolation and disjunctive interpolation for constraints modelling software verification tasks.

1 Introduction

Software model checking has greatly benefited from the combination of a number of seminal ideas: automated abstraction through theorem proving [15], exploration of finite-state abstractions, and counterexample-driven refinement [3]. Even

Philipp Rümmer
Uppsala University, Uppsala, Sweden
E-mail: philipp.ruemmer@it.uu.se

Hossein Hojjat
Department of Computer Science, 442 Gates Hall, Cornell University, Ithaca, NY 14853, USA
E-mail: hojjat@cornell.edu

Viktor Kuncak
EPFL IC IIF LARA, Station 14, CH-1015 Lauusanne, Switzerland
E-mail: viktor.kuncak@epfl.ch

though these techniques can be viewed independently, the effectiveness of verification has been consistently improving by providing more sophisticated communication between these steps. Often, carefully chosen search aspects are being pushed into a learning-enabled constraint solver, resulting in better overall verification performance. An essential advance was to use interpolants derived from unsatisfiability proofs to refine the abstraction [22]. In recent years, we have seen significant progress in interpolating methods for different logical constraints [8, 9, 34], and a wealth of more general forms of interpolation [1,21,34,37].

As a promising direction to extend the reach of automated verification methods to programs with procedures, and concurrent programs, among others, recently the use of Horn constraints as intermediate representation has been proposed [16,17,34]. This paper examines the relationship between various forms of Craig interpolation and syntactically defined fragments of recursion-free Horn clauses, observing in particular a natural correspondence between Craig interpolation and several natural fragments of Horn clauses. Extrapolating from this correspondence, we identify a new notion, *disjunctive interpolants*, which are more general than tree interpolants and inductive sequences of interpolants, but can still effectively be computed. Like tree interpolation [21,34], a disjunctive interpolation query is a tree-shaped constraint specifying the interpolants to be derived; however, in disjunctive interpolation, branching in the tree can represent both conjunctions and disjunctions. We present an algorithm for solving the interpolation problem, relating it to a subclass of recursion-free Horn clauses [17,35,36]. We then consider solving general recursion-free Horn clauses and show that this problem is solvable whenever the logic admits interpolation. We establish tight complexity bounds for solving recursion-free Horn clauses for propositional logic (PSPACE) and for integer linear arithmetic (co-NEXPTIME). In contrast, the disjunctive interpolation problem remains in coNP for these logics. We also show how to use solvers for recursion-free Horn clauses to verify recursive Horn clauses using counterexample-driven predicate abstraction. We present an algorithm and experimental results on publicly available benchmarks.

Organisation. Related work is surveyed in Sect. 1.1, following in Sect. 2 by an example of (recursive) Horn clauses. Sect. 3 formally introduces the concept of Horn clauses. Sect. 4 investigates the relationship between several Horn fragments and Craig interpolation; in more detail, Sect. 5 introduces tree interpolants, and Sect. 6 disjunctive interpolants. Sect. 7 considers the problem of solving general recursion-free sets of Horn clauses. Sect. 8 gives results about the computational complexity of solving recursion-free Horn clauses, and corresponding interpolation problems. Finally, Sect. 9 defines model checking algorithms on the basis of the various interpolation approaches, and Sect. 9.2 provides an empirical evaluation.

1.1 Related Work

There is a long line of research on Craig **interpolation** methods, and generalised forms of interpolation tailored to verification. For an overview of interpolation in the presence of theories, we refer the reader to [8, 9]. Binary Craig interpolation for implications $A \to C$ goes back to [10], was used on conjunctions $A \wedge B$ in [32], and generalised to inductive sequences of interpolants in [22, 33]. The concept of tree interpolation, strictly generalising inductive sequences of interpolants, is presented in the documentation of the interpolation engine iZ3 and in [34]; the computation of tree interpolants by computing a sequence of binary interpolants is also described in [21]. In this paper (extending work in [39,40]), we present a new form of interpolation, *disjunctive interpolation*, which is strictly more general than sequences of interpolants and tree interpolants. Our implementation supports Presburger arithmetic, including divisibility constraints [8], which is rarely supported by existing tools, yet helpful in practice [24].

A further generalisation of inductive sequences of interpolants are restricted DAG interpolants [1], which also include disjunctiveness in the sense that multiple paths through a program can be handled simultaneously. Disjunctive interpolants are incomparable in power to restricted DAG interpolants, since the former does not handle interpolation problems in the form of DAGs, while the latter does not subsume tree interpolation. A combination of the two kinds of interpolants ("disjunctive DAG interpolation") is strictly more powerful (and harder) than disjunctive interpolation, see Sect. 8 for a complexity-theoretic analysis. We discuss techniques and heuristics to practically handle shared sub-trees in disjunctive interpolation, extending the benefits of DAG interpolation to recursive programs.

Horn clauses have extensively been used in the context of (constraint) logic programming community, for the purpose of program analysis and related application (e.g., [4,13,20,36]). The use of Horn clauses as intermediate representation for model checking was proposed in [17], with the verification of concurrent programs as main application. The underlying procedure for solving sets of recursion-free Horn clauses, over the combined theory of linear rational arithmetic and uninterpreted functions, was presented in [18]. Our paper extends this direction by presenting general results about solvability and computational complexity, independent of any particular calculus. Our experiments are with linear *integer* arithmetic, arguably a more faithful model of discrete computation than rationals [24].

An algorithm to solve recursion-free systems of Horn constraints by repeated computation of binary interpolants was given in [43], for the purpose of type inference. Further techniques for solving Horn clauses were developed in [26]. Horn clauses are also used as a format for verification problems supported by the SMT solver Z3 [23], generalising the IC3 algorithm; several recent papers propose optimisations of the algorithm by integrating abstraction [29] and under-approximations [28].

In addition to methods for ordinary (constrained) Horn clauses, a range of extensions have been proposed in literature. This includes support for restricted forms of existential quantification [5], for the computation of solutions in presence of well-foundedness constraints [19], as well as for derivation of quantified predicates in solutions of Horn clauses [7]. Verification methods on the basis of Horn clauses, including inter-procedural model checking, were given in [16].

Inter-procedural **software model checking** with interpolants has been an active area of research. In the context of predicate abstraction, it has been discussed how well-scoped invariants can be inferred [22] in the presence of function calls. Based

```
(1) gcd(M,N,R) ← M = N ∧ R = M
(2) gcd(M,N,R) ← M > N ∧ M1 = M − N ∧ gcd(M1,N,R)
(3) gcd(M,N,R) ← M < N ∧ N1 = N − M ∧ gcd(M,N1,R)
(4) false ← M ≥ 0 ∧ M = N ∧ gcd(M,N,R) ∧ R > M
```

Fig. 1 Horn clauses computing the greatest common divisor of two numbers and an assertion on result. Variables are universally quantified in each clause.

```
(1) gcd(M,N,R) ← M = N ∧ R = M
(1') gcd1(M,N,R) ← M = N ∧ R = M
(2') gcd(M,N,R) ← M > N ∧ M1 = M − N ∧ gcd1(M1,N,R)
(3') gcd(M,N,R) ← M < N ∧ N1 = N − M ∧ gcd1(M,N1,R)
(4) false ← M ≥ 0 ∧ M = N ∧ gcd(M,N,R) ∧ R > M
```

Fig. 2 Extended recursion-free approximation of the Horn clauses in Fig. 1.

on the concept of Horn clauses, a predicate abstraction-based algorithm for bottom-up construction of function summaries was presented in [16]. Verification of programs with procedures is described in [21] (using nested word automata) as well as in [2]. Function summaries generated using interpolants have also been used in bounded model checking [41]. Researchers also showed how to lift these techniques to higher-order programs [25, 44].

2 Example: Verification of Recursive Predicates

We start by showing how our approach can verify programs encoded as Horn clauses, by means of predicate abstraction and a theorem prover for Presburger arithmetic. Fig. 1 shows an example of a system of Horn clauses that compute the greatest common divisor of its first and its second argument in its third argument. After invoking the gcd operation on the equal positive numbers M and N, we wish to check whether it is possible for the result R to be more than the M. In general, we encode error conditions as Horn clauses with *false* in their head, and refer to such clauses as error clauses, although such clauses do not have a special semantic status in our system. When executed with these clauses as input, our verification tool automatically identifies that the definition of gcd(M,N,R) as the predicate $(M = N) \rightarrow (M \geq R)$ gives a solution to these Horn clauses. In terms of safety (partial correctness), this means that the error condition cannot be reached.

Our approach uses counterexample-driven refinement to perform verification. In this example, the abstraction of Horn clauses starts with a trivial set of predicates, containing only the predicate *false*, which is assumed to be a valid approximation until proven otherwise. Upon examining a clause that has a concrete satisfiable formula on the right-hand side (e.g. $M = N \wedge R = M$), we rule out *false* as the approximation of gcd. In the absence of other candidate predicates, the approximation of gcd becomes the conjunction of an empty set of predicates, which is *true*. Using this approximation the error clause is no longer satisfied. At this point the algorithm checks whether a true error is reached by directly chaining the clauses involved in computing the approximation of predicates. This amounts to checking whether the following recursion-free subset of clauses has a solution:

```
(1) gcd(M,N,R) ← M = N ∧ R = M
(4) false   ← M ≥ 0 ∧ M = N ∧ gcd(M,N,R) ∧ R > M
```

The solution to above problem is any formula $I(M, N, R)$ such that

```
    I(M,N,R) ← M = N ∧ R = M
    false   ← M ≥ 0 ∧ M = N ∧ I(M,N,R) ∧ R > M
```

This is precisely an interpolant of $M = N \wedge R = M$ and $M \geq 0 \wedge M = N \wedge R > M$. A valid interpolant is $P_1(M, N, R) \equiv M \geq R$. Choosing this interpolant eliminates the current contradiction for Horn clauses and P_1 is added into a list of abstraction predicates for the relation gcd. Because the predicates approximating gcd are now updated, we consider the abstraction of the system in terms of these predicates.

The predicate P_1 is not a conjunct in a valid approximation for gcd in clause (2), so the following recursion-free unfolding is not solved by the approximation so far:

```
(1)    gcd(M,N,R) ← M = N ∧ R = M
(2')   gcd1(M,N,R) ← M > N ∧ M1 = M − N ∧ gcd(M1,N,R)
(4')   false   ← M ≥ 0 ∧ M = N ∧ gcd1(M,N,R) ∧ R > M
```

This particular problem could be reduced to solving an interpolation sequence, but it is more natural to think of it simply as a solution for recursion-free Horn clauses. A solution is an interpretation of the relations gcd and gcd1 as ternary relations on integers, such that the clauses are true. Note that this problem could also be viewed as the computation of tree interpolants, which are also a special case of solving recursion-free Horn clauses, as are DAG interpolants and a new notion of disjunctive tree interpolants that we introduce. In line with [16–18] we observe that recursion-free clauses are a perfect fit for counterexample-driven verification: they allow us to provide the theorem proving procedure with much more information that they can use to refine abstractions. In the limit, the original set of clauses or its recursive unfoldings are its own approximations, some of them exact, but the advantage of *recursion-free* Horn clauses is that their solvability is decidable under very general conditions. This provides us with a solid theorem proving building block to construct robust and predictable solvers for the undecidable recursive case. Our paper describes a new such building block: disjunctive interpolants, which correspond to a subclass of non-recursive Horn clauses.

To illustrate disjunctive interpolants, Fig. 2 provides another recursion-free approximations of the problem. In this approximation we can distinguish 3 different paths from the error clause (4) through the clauses (1'), (2') and (3') to ground formulae. The traditional refinement approach using e.g. tree interpolation typically removes the 3 instances of the spurious counter-examples using 3 interpolation calls. A novelty of disjunctive interpolation is removing the different choices of counter-examples altogether using a single call to the interpolating theorem prover. Eliminating more counter-examples at once can reduce the number of iterations and increase convergence.

3 Formulae and Horn Clauses

Constraint languages. Throughout this paper, we assume that a first-order vocabulary of *interpreted symbols* has been fixed, consisting of a set $\mathcal{F}$ of fixed-arity function symbols, and a set $\mathcal{P}$ of fixed-arity predicate symbols. Interpretation of $\mathcal{F}$ and $\mathcal{P}$ is determined by a class $\mathcal{S}$ of structures (U, I) consisting of non-empty universe U, and a mapping I that assigns to each function in $\mathcal{F}$ a set-theoretic function over U, and to each predicate in $\mathcal{P}$ a set-theoretic relation over U. As a convention, we assume the presence of an equation symbol "=" in $\mathcal{P}$, with the usual interpretation. Given a countably infinite set $\mathcal{X}$ of variables, a *constraint language* is a set *Constr* of first-order formulae over $\mathcal{F}, \mathcal{P}, \mathcal{X}$ For example, the language of quantifier-free Presburger arithmetic has $\mathcal{F} = \{+, -, 0, 1, 2, \ldots\}$ and $\mathcal{P} = \{=, \leq, |\}$).

A constraint is called *satisfiable* if it holds for some structure in $\mathcal{S}$ and some assignment of the variables $\mathcal{X}$, otherwise *unsatisfiable*. We say that a set $\Gamma \subseteq Constr$ of constraints *entails* a constraint $\phi \in Constr$ if every structure and variable assignment that satisfies all constraints in Γ also satisfies ϕ; this is denoted by $\Gamma \models \phi$.

$fv(\phi)$ denotes the set of free variables in constraint ϕ. We write $\phi[x_1, \ldots, x_n]$ to state that a constraint contains (only) the free variables $x_1, \ldots, x_n$, and $\phi[t_1, \ldots, t_n]$ for the result of substituting the terms $t_1, \ldots, t_n$ for $x_1, \ldots, x_n$. Given a constraint ϕ containing the free variables $x_1, \ldots, x_n$, we write $Cl_\forall(\phi)$ for the *universal closure* $\forall x_1, \ldots, x_n.\phi$.

Positions. We denote the set of *positions* in a constraint ϕ by *positions*(ϕ). For instance, the constraint $a \wedge \neg a$ has 4 positions, corresponding to the sub-formulae $a \wedge \neg a, \neg a$, and the two occurrences of a. The sub-formula of a formula ϕ underneath a position p is denoted by $\phi \downarrow p$, and we write $\phi[p/\psi]$ for the result of replacing the sub-formula $\phi \downarrow p$ with ψ. Further, we write $p \leq q$ if position p is above q (that is, q denotes a position within the sub-formula $\phi \downarrow p$), and $p < q$ if p is strictly above q.

Craig interpolation is the main technique used to construct and refine abstractions in software model checking. A binary interpolation problem is a conjunction $A \wedge B$ of constraints. A *Craig interpolant* is a constraint I such that $A \models I$ and $B \models \neg I$, and such that $fv(I) \subseteq fv(A) \cap fv(B)$. The existence of an interpolant implies that $A \wedge B$ is unsatisfiable. We say that a constraint language has the *interpolation property* if also the opposite holds: whenever $A \wedge B$ is unsatisfiable, there is an interpolant I.

3.1 Horn Clauses

To define the concept of Horn clauses, we fix a set $\mathcal{R}$ of uninterpreted fixed-arity *relation symbols,* disjoint from $\mathcal{P}$ and $\mathcal{F}$. A *Horn clause* is a formula $C \wedge B_1 \wedge \cdots \wedge B_n \rightarrow H$ where

- C is a constraint over $\mathcal{F}, \mathcal{P}, \mathcal{X}$;
- each B_i is an application $p(t_1, \ldots, t_k)$ of a relation symbol $p \in \mathcal{R}$ to first-order terms over $\mathcal{F}, \mathcal{X}$;
- H is similarly either an application $p(t_1, \ldots, t_k)$ of $p \in \mathcal{R}$ to first-order terms, or is the constraint *false*.

H is called the *head* of the clause, $C \wedge B_1 \wedge \cdots \wedge B_n$ the *body.* In case $C = true$, we usually leave out C and just write $B_1 \wedge \cdots \wedge B_n \rightarrow H$. First-order variables (from $\mathcal{X}$) in a clause are considered implicitly universally quantified; relation symbols represent set-theoretic relations over the universe U of a structure $(U, I) \in \mathcal{S}$. Notions like (un)satisfiability and entailment generalise straightforwardly to formulae with relation symbols.

A *relation symbol assignment* is a mapping $sol : \mathcal{R} \to Constr$ that maps each n-ary relation symbol $p \in \mathcal{R}$ to a constraint $sol(p) = C_p[x_1, \ldots, x_n]$ with n free variables. The *instantiation* $sol(h)$ of a Horn clause h is defined by:

$$sol(C \wedge p_1(\bar{t}_1) \wedge \cdots \wedge p_n(\bar{t}_n) \to p(\bar{t})) = C \wedge sol(p_1)[\bar{t}_1] \wedge \cdots \wedge sol(p_n)[\bar{t}_n] \to sol(p)[\bar{t}]$$
$$sol(C \wedge p_1(\bar{t}_1) \wedge \cdots \wedge p_n(\bar{t}_n) \to false) = C \wedge sol(p_1)[\bar{t}_1] \wedge \cdots \wedge sol(p_n)[\bar{t}_n] \to false$$

Definition 1 (Solvability) Let $\mathcal{HC}$ be a set of Horn clauses over relation symbols $\mathcal{R}$.

1. $\mathcal{HC}$ is called *semantically solvable* if for every structure $(U, I) \in \mathcal{S}$ there is an interpretation of the relation symbols $\mathcal{R}$ as set-theoretic relations over U such that the universal closure $Cl_\forall(h)$ of every clause $h \in \mathcal{HC}$ holds in (U, I).
2. $\mathcal{HC}$ is called *syntactically solvable* if there is a relation symbol assignment sol such that for every structure $(U, I) \in \mathcal{S}$ and every clause $h \in \mathcal{HC}$ it is the case that $Cl_\forall(sol(h))$ is satisfied.

Note that, in the special case when $\mathcal{S}$ contains only one structure, $\mathcal{S} = \{(U, I)\}$, semantic solvability reduces to the existence of relations interpreting $\mathcal{R}$ that extend the structure (U, I) in such a way to make all clauses true. In other words, Horn clauses are solvable in a structure if and only if the extension of the theory of (U, I) by relation symbols $\mathcal{R}$ in the vocabulary and by given Horn clauses as axioms is consistent.

A set $\mathcal{HC}$ of Horn clauses induces a *dependence relation* $\to_{\mathcal{HC}}$ on $\mathcal{R}$, defining $p \to_{\mathcal{HC}} q$ if there is a Horn clause in $\mathcal{HC}$ that contains p in its head, and q in the body. The set $\mathcal{HC}$ is called *recursion-free* if $\to_{\mathcal{HC}}$ is acyclic, and *recursive* otherwise. In the next sections we study the solvability problem for recursion-free Horn clauses; in particular, Theorem 3 below characterises the relationship between syntactic and semantic solvability for recursion-free Horn clauses. This case is relevant, since solvers for recursion-free Horn clauses form a main component of many general Horn-clause-based verification systems [16, 17].

3.2 Semantic and Syntactic Solvability

Clearly, if a set of Horn clauses is syntactically solvable, then it is also semantically solvable. The converse is not true in general, because the solution need not be expressible in the constraint language. Consider the following clause set $\mathcal{HC}$:

```
multA(X,Y,Z) ← X = 0 ∧ Z = 0
multA(X,Y,Z) ← multA(X1,Y,Z1) ∧ X1 = X − 1 ∧ Z = Z1 + Y
multB(X,Y,Z) ← X = 0 ∧ Z = 0
multB(X,Y,Z) ← multB(X1,Y,Z1) ∧ X1 = X − 1 ∧ Z = Z1 + Y
false  ← multA(X,Y,Z1) ∧ multB(X,Y,Z2) ∧ Z1 ≠ Z2
```

The clauses define two version of a multiplication and assert that the result is functionally determined by the first two arguments. Let $a, b \subseteq \mathbb{Z}^3$ denote the interpretations of multA and multB, respectively, in any solution that satisfies all Horn clauses.

Form of interpolation	**Fragment of Horn clauses**
Binary interpolation [10,32] $A \wedge B$	Pair of Horn clauses $A \to p(\bar{x}),\ B \wedge p(\bar{x}) \to \mathit{false}$ with $\{\bar{x}\} = \mathit{fv}(A) \cap \mathit{fv}(B)$
Inductive interpolant seq. [22,33] $T_1 \wedge T_2 \wedge \cdots \wedge T_n$	Linear tree-like Horn clauses $T_1 \to p_1(\bar{x}_1),\ \ p_1(\bar{x}_1) \wedge T_2 \to p_2(\bar{x}_2),\ \ \ldots$ with $\{\bar{x}_i\} = \mathit{fv}(T_1, \ldots, T_i) \cap \mathit{fv}(T_{i+1}, \ldots, T_n)$
Tree interpolants (Sect. 5)	Tree-like Horn clauses
(Restricted) DAG interpolants [1]	Linear Horn clauses
Disjunctive interpolants (Sect. 6)	Body disjoint Horn clauses

Table 1 Equivalence of interpolation problems and systems of Horn clauses.

We show that the only possibility is that $a = b = m$ where $m = \{(x, y, z) \in \mathbb{Z}^3 \mid z = xy\}$ is the multiplication relation. Indeed, by induction we can easily prove that $m \subseteq a$ and $m \subseteq b$, using the first four clauses. To show the converse, suppose on the contrary, that $(x, y, z) \in a$ where $z \neq xy$ (the case for $(x, y, z) \in b$ is symmetrical). Because $(x, y, xy) \in b$ and $x \neq z$, the last clause does not hold, a contradiction.

Therefore, the clauses have a unique solution $a = b = m$, but this solution is not definable in a Presburger arithmetic (e.g., by semilinearity of the solution sets, or by decidability of Presburger arithmetic vs. undecidability of its extension with multiplication). Therefore, the above clauses give an example of clauses that are semantically but not syntactically solvable in Presburger arithmetic.

Further such examples can be constructed by using Horn clauses to define other total computable functions that are not definable in Presburger arithmetic alone.

4 The Relationship between Craig Interpolation and Horn Clauses

It has become common to work with generalised forms of Craig interpolation, such as inductive sequences of interpolants, tree interpolants, and restricted DAG interpolants. We show that a variety of such interpolation approaches can be reduced to recursion-free Horn clauses. Recursion-free Horn clauses thus provide a general framework unifying and subsuming a number of earlier notions. As a side effect, we can formulate a general theorem about existence of the individual kinds of interpolants in Sect. 8, applicable to any constraint language with the (binary) interpolation property.

An overview of the relationship between specific forms of interpolation and specific fragments of recursions-free Horn clauses is given in Table 1, and will be explained in more detail in the rest of this section. Table 1 refers to the following fragments of recursion-free Horn clauses:

Definition 2 (Horn clause fragments) We say that a finite, recursion-free set $\mathcal{HC}$ of Horn clauses

1. is *linear* if the body of each Horn clause contains at most one relation symbol,

2. is *body-disjoint* if for each relation symbol p there is at most one clause containing p in its body; furthermore, every clause contains p at most once;
3. is *head-disjoint* if for each relation symbol p there is at most one clause containing p in its head;
4. is *tree-like* [18] if it is body-disjoint and head-disjoint.

Theorem 1 (Interpolation and Horn clauses) *For each line of Table 1 it holds that:*

1. *an interpolation problem of the stated form can be polynomially reduced to (syntactically) solving a set of Horn clauses, in the stated fragment;*
2. *solving a set of Horn clauses (syntactically) in the stated fragment can be polynomially reduced to solving a sequence of interpolation problems of the stated form.*

As an illustration, consider the case of a binary interpolation problem $A \wedge B$, in which a constraint I such that $A \models I$, $B \models \neg I$, and $\mathit{fv}(I) \subseteq \mathit{fv}(A) \cap \mathit{fv}(B)$ has to be determined. To encode a binary interpolation problem into Horn clauses, we first determine the set $\bar{x} = \mathit{fv}(A) \cap \mathit{fv}(B)$ of variables that can possibly occur in the interpolant. We then pick a relation symbol p of arity $|\bar{x}|$, and define two Horn clauses expressing that $p(\bar{x})$ is an interpolant:

$$A \to p(\bar{x}), \qquad B \wedge p(\bar{x}) \to \mathit{false}$$

It is clear that every syntactic solution for the two Horn clauses corresponds to an interpolant of $A \wedge B$. Vice versa, for every pair of complementary Horn clauses $C_1 \to p(\bar{t})$ and $C_2 \wedge p(\bar{s}) \to \mathit{false}$, we can determine solutions by first normalising the clauses to $C_1 \wedge \bar{t} = \bar{x} \to p(\bar{x})$ and $C_2 \wedge \bar{s} = \bar{x} \wedge p(\bar{x}) \to \mathit{false}$, and then finding solutions for the binary interpolation problem $(C_1 \wedge \bar{t} = \bar{x}) \wedge (C_2 \wedge \bar{s} = \bar{x})$.

Proofs for the most cases of Theorem 1, in particular for inductive sequences of interpolants and DAG interpolants, are given in [38]. On the next pages, we first give a proof for a particularly important form of interpolation, *tree interpolation*, and then present a generalisation to *disjunctive interpolation.*

5 Tree Interpolants

Tree interpolants [21, 31] strictly generalise inductive sequences of interpolants, and are designed with the application of inter-procedural verification in mind: in this context, the tree structure of the interpolation problem corresponds to (a part of) the call graph of a program. Tree interpolation problems correspond to recursion-free tree-like sets of Horn clauses.

Suppose (V, E) is a finite directed tree, writing $E(v, w)$ to express that the node w is a direct child of v. Further, suppose $\phi : V \to \mathit{Constr}$ is a function that labels each node v of the tree with a formula $\phi(v)$. A labelling function $I : V \to \mathit{Constr}$ is called a *tree interpolant* (for (V, E) and ϕ) if the following properties hold:

1. for the root node $v_0 \in V$, it is the case that $I(v_0) = \mathit{false}$,

2. for any node $v \in V$, the following entailment holds:

$$\phi(v) \wedge \bigwedge_{(v,w)\in E} I(w) \models I(v) ,$$

3. for any node $v \in V$, every non-logical symbol (in our case: variable) in $I(v)$ occurs both in some formula $\phi(w)$ for w such that $E^*(v, w)$, and in some formula $\phi(w')$ for some w' such that $\neg E^*(v, w')$. (E^* is the reflexive transitive closure of E).

Since the case of tree interpolants is instructive for solving recursion-free sets of Horn clauses in general, we give a result about the existence of tree interpolants. The proof of the lemma computes tree interpolants by repeated derivation of binary interpolants; however, as for inductive sequences of interpolants, there are solvers that can compute all formulae of a tree interpolant simultaneously [17, 18, 31].

Lemma 1 *Suppose the constraint language Constr that has the interpolation property. Then a tree (V, E) with labelling function $\phi : V \to Constr$ has a tree interpolant I if and only if $\bigwedge_{v\in V} \phi(v)$ is unsatisfiable.*

Proof "⇒" follows from the observation that every interpolant $I(v)$ is a consequence of the conjunction $\bigwedge_{(v,w)\in E^+} \phi(w)$.

"⇐": let $v_1, v_2, \ldots, v_n$ be an inverse topological ordering of the nodes in (V, E), i.e., an ordering such that $\forall i, j.\ (E(v_i, v_j) \Rightarrow i > j)$. We inductively construct a sequence of formulae $I_1, I_2, \ldots, I_n$, such that for every $i \in \{1, \ldots, n\}$ the following properties hold:

1. the following conjunction is unsatisfiable:

$$\bigwedge\{I_k \mid k \le i,\ \forall j.\ (E(v_j, v_k) \Rightarrow j > i)\} \ \wedge\ \big(\phi(v_{i+1}) \wedge \phi(v_{i+2}) \wedge \cdots \wedge \phi(v_n)\big) \quad (1)$$

2. the following entailment holds:

$$\phi(v_i) \wedge \bigwedge_{(v_i,v_j)\in E} I_j \models I_i$$

3. every non-logical symbol in I_i occurs both in a formula $\phi(w)$ with $E^*(v_i, w)$, and in a formula $\phi(w')$ with $\neg E^*(v_i, w')$.

Assume that the formulae $I_1, I_2, \ldots, I_i$ have been constructed, for $i \in \{0, \ldots, n-1\}$. We then derive the next interpolant I_{i+1} by solving the binary interpolation problem

$$\Big(\phi(v_{i+1}) \wedge \bigwedge_{E(v_{i+1},v_j)} I_j\Big) \ \wedge$$
$$\Big(\bigwedge\{I_k \mid k \le i,\ \forall j.\ (E(v_j, v_k) \Rightarrow j > i+1)\} \ \wedge\ \phi(v_{i+2}) \wedge \cdots \wedge \phi(v_n)\Big) \quad (2)$$

That is, we construct I_{i+1} so that the following entailments hold:

$$\phi(v_{i+1}) \wedge \bigwedge_{E(v_{i+1},v_j)} I_j \models I_{i+1},$$

$$\bigwedge\{I_k \mid k \le i,\ \forall j.\ (E(v_j, v_k) \Rightarrow j > i+1)\} \ \wedge\ \phi(v_{i+2}) \wedge \cdots \wedge \phi(v_n) \models \neg I_{i+1}$$

Furthermore, I_{i+1} only contains non-logical symbols that are common to the left and the right side of the conjunction.

Note that (2) is equivalent to (1), therefore unsatisfiable, and a well-formed interpolation problem. It is also easy to see that the properties 1–3 hold for I_{i+1}. Also, we can verify that the labelling function $I : v_i \mapsto I_i$ is a solution for the tree interpolation problem defined by (V, E) and ϕ. $\square$

Tree interpolation as Tree-like Horn clauses. In order to encode a tree interpolation problem as a tree-like set of Horn clauses, we first introduce a fresh relation symbol p_v for each node $v \in V$ of a tree interpolation problem $(V, E), \phi$, assuming that for each $v \in V$ the vector $\bar{x}_v = \bigcup_{E^*(v,w)} fv(\phi(w)) \cap \bigcup_{\neg E^*(v,w)} fv(\phi(w))$ represents the set of variables that can occur in the interpolant $I(v)$. The interpolation problem is then represented by the following clauses:

$$p_0(\bar{x}_0) \rightarrow \mathit{false}, \quad \Big\{ \phi(v) \wedge \bigwedge_{(v,w)\in E} p_w(\bar{x}_w) \rightarrow p_v(\bar{x}_v) \Big\}_{v \in V}$$

Tree-like Horn clauses as tree interpolation. Suppose $\mathcal{HC}$ is a finite, recursion-free, and tree-like set of Horn clauses. We can solve the system of Horn clauses by computing a tree interpolant for every connected component of the $\rightarrow_{\mathcal{HC}}$-graph. As before, we first normalise the Horn clauses by fixing, for every relation symbol p, a unique vector of variables $\bar{x}_p$, and rewriting $\mathcal{HC}$ such that p only occurs in the form $p(\bar{x}_p)$. We also ensure that every variable x that is not argument of a relation symbol occurs in at most one clause. The tree interpolation graph (V, E) is then defined by choosing the set $V = \mathcal{R} \cup \{\mathit{false}\}$ of relation symbols as nodes, and the child relation $E(p, q)$ to hold whenever p occurs as head, and q within the body of a clause. The labelling function ϕ is defined by $\phi(p) = C$ whenever there is a clause with head symbol p and constraint C, and $\phi(p) = \mathit{false}$ if p does not occur as head of any clause.

Example 1 We consider the following recursion-free set of Horn clauses:

```
r1(X, R)      ←  true
r2(X', R)     ←  r1(X, R) ∧ X' ≥ 0
r3(N, C, T)   ←  true
r4(N, C, T)   ←  r3(N, C, T) ∧ N ≤ 0
r5(N, C', T)  ←  r4(N, C, T) ∧ C' = 1
r6(N, C)      ←  r5(N, C, T)
r7(X, R')     ←  r2(X, R) ∧ r6(X, R')
false         ←  r7(X, R) ∧ R ≠ X + 1
```

Note that this recursion-free subset of the clauses is body-disjoint and head-disjoint, and thus tree-like. In order to compute a (syntactic) solution of the clauses, we set up the corresponding tree interpolation problem. Fig. 3 (left) shows the tree with the labelling ϕ to be interpolated (in grey), as well as the head literals of the clauses generating the nodes of the tree; the labelling ϕ contains additional equations needed to match up the arguments of the various relation symbols. A tree interpolant solving the interpolation problem is given in Fig. 3 (right). The tree interpolant can straightforwardly be mapped to a solution of the original tree-like Horn, for instance we set $r_4(n_4, c_4, t_4) = (n_4 \leq 0)$ and $r_5(n_5, c_5, t_5) = (n_5 \leq -1 \vee (c_5 = 1 \wedge n_5 = 0))$.

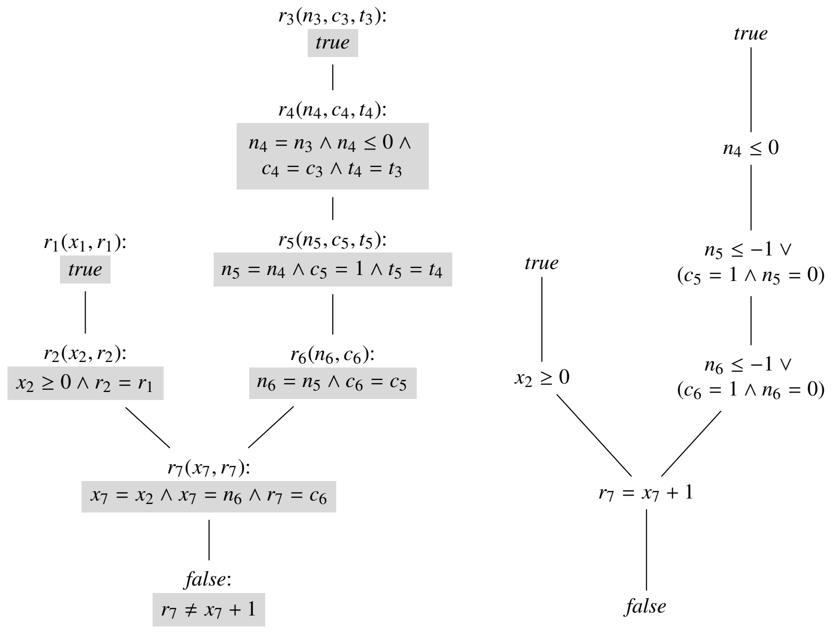

$r_3(n_3, c_3, t_3)$:
true
$r_4(n_4, c_4, t_4)$:
$n_4 = n_3 \wedge n_4 \leq 0 \wedge c_4 = c_3 \wedge t_4 = t_3$
$r_1(x_1, r_1)$:
true
$r_5(n_5, c_5, t_5)$:
$n_5 = n_4 \wedge c_5 = 1 \wedge t_5 = t_4$
$r_2(x_2, r_2)$:
$x_2 \geq 0 \wedge r_2 = r_1$
$r_6(n_6, c_6)$:
$n_6 = n_5 \wedge c_6 = c_5$
$r_7(x_7, r_7)$:
$x_7 = x_2 \wedge x_7 = n_6 \wedge r_7 = c_6$
false:
$r_7 \neq x_7 + 1$
true
$n_4 \leq 0$
true
$n_5 \leq -1 \vee (c_5 = 1 \wedge n_5 = 0)$
$x_2 \geq 0$
$n_6 \leq -1 \vee (c_6 = 1 \wedge n_6 = 0)$
$r_7 = x_7 + 1$
false

Fig. 3 Tree interpolation problem for the clauses in Example 1 (left), and a tree interpolant solving the interpolation problem (right).

6 Disjunctive Interpolants and Body-Disjoint Horn Clauses

We now present our notion of disjunctive interpolants, and the corresponding class of Horn clauses. Our inspiration are generalized forms of Craig interpolation, such as inductive sequences of interpolants [22,33] or tree interpolants [21,34]. We introduce disjunctive interpolation as a new form of interpolation that is tailored to the refinement of abstractions in Horn clause verification, strictly generalising both inductive sequences of interpolants and tree interpolation. Disjunctive interpolation problems can specify both conjunctive and disjunctive relationships between interpolants, and are thus applicable for simultaneous analysis of multiple paths in a program, but also tailored to inter-procedural analysis or verification of concurrent programs [16].

Disjunctive interpolation problems correspond to a specific fragment of recursion-free Horn clauses, namely recursion-free body-disjoint Horn clauses (see Sect. 6.1). The definition of disjunctive interpolation is chosen deliberately to be as general as possible, while still avoiding the high computational complexity of solving general systems of recursion-free Horn clauses. Computational complexity is discussed in Sect. 8.

We introduce disjunctive interpolants as a form of *sub-formula abstraction*. For example, given an unsatisfiable constraint $\phi[\alpha]$ containing α as a sub-formula in a positive position, the goal is to find an abstraction α' such that $\alpha \models \alpha'$ and $\alpha[\alpha'] \models$ *false*, and such that α' only contains variables common to α and $\phi[true]$. Generalizing this to any number of subformulas, we obtain the following.

Definition 3 (Disjunctive interpolant) Let ϕ be a constraint, and $pos \subseteq positions(\phi)$ a set of positions in ϕ that are only underneath the connectives $\wedge$ and $\vee$. A *disjunctive interpolant* is a map $I : pos \rightarrow Constr$ from positions to constraints such that:

1. For each position $p \in pos$, with direct children $\{q_1, \ldots, q_n\} = \{q \in pos \mid p < q \text{ and } \neg\exists r \in pos.\ p < r < q\}$ we have
$$(\phi[q_1/I(q_1), \ldots, q_n/I(q_n)]) \downarrow p \models I(p)\,,$$
2. For the topmost positions $\{q_1, \ldots, q_n\} = \{q \in pos \mid \neg\exists r \in pos.\ r < q\}$ we have
$$\phi[q_1/I(q_1), \ldots, q_n/I(q_n)] \models false\,,$$
3. For each position $p \in pos$, we have $fv(I(p)) \subseteq fv(\phi \downarrow p) \cap fv(\phi[p/true])$.

Example 2 Consider $A_p \wedge B$, with position p pointing to the sub-formula A, and $pos = \{p\}$. The disjunctive interpolants for $A \wedge B$ and pos coincide with the ordinary binary interpolants for $A \wedge B$.

Example 3 Consider the formula $\phi = (\cdots(((T_1)_{p_1} \wedge T_2)_{p_2} \wedge T_3)_{p_3} \wedge \cdots)_{p_{n-1}} \wedge T_n$ and positions $pos = \{p_1, \ldots, p_{n-1}\}$. Disjunctive interpolants for ϕ and pos correspond to inductive sequences of interpolants [22, 33]. Note that we have the entailments $T_1 \models I(p_1),\ I(p_1) \wedge T_2 \models I(p_2),\ \ldots,\ I(p_{n-1}) \wedge T_n \models false$.

Example 4 Tree interpolation (Sect. 5) corresponds to disjunctive interpolation with a set pos of positions that are only underneath $\wedge$ (and never underneath $\vee$).

Example 5 We consider the example given in Fig. 2, Sect. 2. To compute a solution for the Horn clauses, we first *expand* the Horn clauses into a constraint, by means of exhaustive inlining/resolution (see Sect. 7), obtaining a disjunctive interpolation problem:

$$\begin{aligned}
false &\leadsto M \geq 0 \wedge M = N \wedge \mathsf{gcd}(M, N, R) \wedge R > M \\
&\leadsto \begin{pmatrix} M \geq 0 \\ \wedge\ M = N \\ \wedge\ R > M \end{pmatrix} \wedge \begin{pmatrix} M = N \wedge R = M \\ \vee \\ M > N \wedge M_1 = M - N \wedge \mathsf{gcd1}(M_1, N, R) \\ \vee \\ M < N \wedge N_1 = N - M \wedge \mathsf{gcd1}(M, N_1, R) \end{pmatrix} \\
&\leadsto \begin{pmatrix} M \geq 0 \\ \wedge\ M = N \\ \wedge\ R > M \end{pmatrix} \wedge \begin{pmatrix} M = N \wedge R = M \\ \vee \\ M > N \wedge M_1 = M - N \wedge (M_1 = N \wedge R = M_1)_q \\ \vee \\ M < N \wedge N_1 = N - M \wedge (M = N_1 \wedge R = M)_r \end{pmatrix}_p
\end{aligned}$$

In the last formula, the positions p, q, r corresponding to the relation symbol gcd and the two occurrences of $\mathsf{gcd1}$ are marked. It can be observed that the last formula is unsatisfiable, and that $I = \{p \mapsto ((M = N) \rightarrow (M \geq R)),\ q \mapsto true,\ r \mapsto true\}$ is a disjunctive interpolant. A solution for the Horn clauses can be derived from the interpolant by conjoining the constraints derived for the two occurrences of $\mathsf{gcd1}$:

$$\mathsf{gcd}(M, N, R) = ((M = N) \rightarrow (M \geq R)), \quad \mathsf{gcd1}(M, N, R) = true$$

Theorem 2 *Suppose ϕ is a constraint, and suppose $pos \subseteq positions(\phi)$ is a set of positions in ϕ that are only underneath the connectives $\wedge$ and $\vee$. If Constr is a constraint language that has the interpolation property, then a disjunctive interpolant I exists for ϕ and pos if and only if ϕ is unsatisfiable.*

Proof "$\Rightarrow$" By means of simple induction, we can derive that $\phi \downarrow p \models I(p)$ holds for every disjunctive interpolant I for ϕ and pos, and for every $p \in pos$. From Def. 3, it then follows that ϕ is unsatisfiable.

"$\Leftarrow$" Suppose ϕ is unsatisfiable. We encode the disjunctive interpolation problem into a (conjunctive) tree interpolation problem by adding auxiliary Boolean variables.[1] Wlog, we assume that pos contains the root position $root$ of ϕ. The graph of the tree interpolation problem is (pos, E), with the edge relation $E = \{(p, q) \mid p < q \text{ and } \neg\exists r. p < r < q\}$. For every $p \in pos$, let a_p be a fresh Boolean variable. We label the nodes of the tree using the function $\phi_L : pos \to Constr$. For each position $p \in pos$, with direct children $\{q_1, \ldots, q_n\} = \{q \in pos \mid E(p, q)\}$ we define

$$\phi_L(p) = \begin{cases} \phi[q_1/a_{q_1}, \ldots, q_n/a_{q_n}] & \text{if } p = root \\ \neg a_p \vee (\phi[q_1/a_{q_1}, \ldots, q_n/a_{q_n}]) \downarrow p & \text{otherwise} \end{cases}$$

Observe that $\bigwedge_{p \in pos} \phi_L(p)$ is unsatisfiable. According to [37], a tree interpolant I_T exists for this labelling function. By construction, for non-root positions $p \in pos \setminus \{root\}$ the interpolant labelling is equivalent to $I_T(p) \equiv \neg a_p \vee I_p$, where I_p does not contain any further auxiliary Boolean variables. We can then construct a disjunctive interpolant I for the original problem as

$$I(p) = \begin{cases} \mathit{false} & \text{if } p = root \\ I_p & \text{otherwise} \end{cases}$$

To see that I is a disjunctive interpolant, observe that for each position $p \in pos$ with direct children $\{q_1, \ldots, q_n\} = \{q \in pos \mid E(p, q)\}$ the following entailment holds (since I_T is a tree interpolant): $\phi_L(p) \wedge (\neg a_{q_1} \vee I_{q_1}) \wedge \cdots \wedge (\neg a_{q_n} \vee I_{q_n}) \models I_T(p)$ Via Boolean reasoning this implies: $(\phi[q_1/I_{q_1}, \ldots, q_n/I_{q_n}]) \downarrow p \models I(p)$. □

The proof provides a constructive method to solve disjunctive interpolation problems, by means of transformation to a tree interpolation problem. This is also the algorithm that we used in our experiments in Sect. 9.2; practical aspects of this approach are discussed in the beginning of Sect. 9.

6.1 Solvability of Body-Disjoint Horn Clauses

Disjunctive interpolation corresponds to a specific class of recursion-free clauses, namely Horn clauses that are *body disjoint* (Def. 2); in contrast, the Horn clauses do not have to be *head disjoint,*, i.e., it is possible to consider multiple clauses with the same head symbol. Syntactic solutions of a set $\mathcal{HC}$ of body-disjoint Horn clauses

[1] The concept of auxiliary Boolean variables to represent interpolation problems has also been used in [41] and [2], for the purpose of extracting function summaries in model checking.

can be computed by solving a disjunctive interpolation problem; vice versa, every disjunctive interpolation problem can be translated into an equivalent set of body-disjoint clauses.

In order to extract an interpolation problem from $\mathcal{HC}$, we first normalise the clauses: for every relation symbol $p \in \mathcal{R}$, we fix a unique vector of variables $\bar{x}_p$, and rewrite $\mathcal{HC}$ such that p only occurs in the form $p(\bar{x}_p)$. This is possible due to the fact that $\mathcal{HC}$ is body disjoint. The translation from Horn clauses to a disjunctive interpolation problem is done recursively, similar in spirit to inlining of function invocations in a program; thanks to body-disjointness, the encoding is polynomial.

$$\begin{aligned} \mathit{enc}(\mathcal{HC}) &= \bigvee_{(C \wedge B_1 \wedge \cdots \wedge B_n \to \mathit{false}) \in \mathcal{HC}} C \wedge \mathit{enc}'(B_1) \wedge \cdots \wedge \mathit{enc}'(B_n) \\ \mathit{enc}'(p(\bar{x}_p)) &= \left(\bigvee_{(C \wedge B_1 \wedge \cdots \wedge B_n \to p(\bar{x}_p)) \in \mathcal{HC}} C \wedge \mathit{enc}'(B_1) \wedge \cdots \wedge \mathit{enc}'(B_n) \right)_{l_p} \end{aligned}$$

Note that the resulting formula $\mathit{enc}(\mathcal{HC})$ contains a unique position l_p at which the definition of a relation symbol p is inlined; in the second equation, this position is marked with l_p. Any disjunctive interpolant I for this set of positions represents a syntactic solution of $\mathcal{HC}$, and vice versa.

7 Solvability of Recursion-free Horn Clauses

The previous section discussed how the class of recursion-free body-disjoint Horn clauses can be solved by reduction to disjunctive interpolation. We next show that this construction can be generalised to arbitrary systems of recursion-free Horn clauses. In absence of the body-disjointness condition, however, the encoding of Horn clauses as interpolation problems can incur a potentially exponential blowup. We give a complexity-theoretic argument justifying that this blowup cannot be avoided in general. This puts disjunctive interpolation (and, equivalently, body-disjoint Horn clauses) at a sweet spot: preserving the relatively low complexity of ordinary binary Craig interpolation, while carrying much of the flexibility of the Horn clause framework.

We first introduce the exhaustive *expansion* $\mathit{exp}(\mathcal{HC})$ of a set $\mathcal{HC}$ of Horn clauses, which generalises the Horn clause encoding from the previous section. We write $C' \wedge B'_1 \wedge \cdots \wedge B'_n \to H'$ for a fresh variant of a Horn clause $C \wedge B_1 \wedge \cdots \wedge B_n \to H$, i.e., the clause obtained by replacing all free first-order variables with fresh variables. Expansion is then defined by the following recursive functions:

$$\begin{aligned} \mathit{exp}(\mathcal{HC}) &= \bigvee_{(C \wedge B_1 \wedge \cdots \wedge B_n \to \mathit{false}) \in \mathcal{HC}} C' \wedge \mathit{exp}'(B'_1) \wedge \cdots \wedge \mathit{exp}'(B'_n) \\ \mathit{exp}'(p(\bar{t})) &= \bigvee_{(C \wedge B_1 \wedge \cdots \wedge B_n \to p(\bar{s})) \in \mathcal{HC}} C' \wedge \mathit{exp}'(B'_1) \wedge \cdots \wedge \mathit{exp}'(B'_n) \wedge \bar{t} = \bar{s}' \end{aligned}$$

Note that exp is only well-defined for finite and recursion-free sets of Horn clauses, since the expansion might not terminate otherwise.

Theorem 3 (Solvability of recursion-free Horn clauses) *Let $\mathcal{HC}$ be a finite and recursion-free set of Horn clauses. If the underlying constraint language has the interpolation property, then the following statements are equivalent:*

1. *$\mathcal{HC}$ is semantically solvable;*
2. *$\mathcal{HC}$ is syntactically solvable;*
3. *$exp(\mathcal{HC})$ is unsatisfiable.*

Proof $2 \Rightarrow 1$ holds because a syntactic solution gives rise to a semantic solution by interpreting the solution constraints. $\neg 3 \Rightarrow \neg 1$ holds because a model of $exp(\mathcal{HC})$ witnesses domain elements that every semantic solution of $\mathcal{HC}$ has to contain, but which violate at least one clause of the form $C \wedge B_1 \wedge \cdots \wedge B_n \rightarrow \mathit{false}$, implying that no semantic solution can exist.

$3 \Rightarrow 2$ is shown by encoding $\mathcal{HC}$ into a disjunctive interpolation problem (Sect. 6), which can solved with the help of Theorem 2. To this end, clauses are first duplicated to obtain a problem that is body disjoint, and subsequently normalised as described in Sect. 6.1. More details are given in Appendix A of [40]. □

8 The Complexity of Recursion-free Horn Clauses

Theorem 3 gives rise to a general algorithm for (syntactically) solving recursion-free sets $\mathcal{HC}$ of Horn clauses, over constraint languages for which interpolation procedures are available. The general algorithm requires, however, to generate and solve the expansion $exp(\mathcal{HC})$ of the Horn clauses, which can be exponentially bigger than $\mathcal{HC}$ (in case $\mathcal{HC}$ is not body disjoint), and might therefore require exponential time. This leads to the question whether more efficient algorithms are possible for solving Horn clauses.

8.1 Complexity for Boolean Constraints

We give a number of complexity results about (semantic) Horn clause solvability. Most importantly, we can observe that solvability is PSPACE-hard, for every non-trivial constraint language *Constr*. The authors of [30] conjecture a similar complexity result for the case of programs with procedures. In [14] a similar result is proved for non-recursive Boolean programs.

Lemma 2 *Suppose a constraint language can distinguish at least two values, i.e., there are two ground terms t_0 and t_1 such that $t_0 \neq t_1$ is satisfiable. Then the semantic solvability problem for recursion-free Horn clauses is PSPACE-hard.*

Proof We reduce the unsatisfiability problem of quantified Boolean formulae (QBF, known to be PSPACE-hard) to solvability of recursion-free Horn clauses. Assume an arbitrary QBF of the shape $\phi = Q_1 x_1 . Q_2 x_2 .... Q_n x_n . F$, where $Q_i \in \exists, \forall$ are quantifiers, x_i are all variables occurring in the formula, and F is a quantifier-free Boolean formula in CNF.

We translate ϕ into a recursion-free set of Horn clauses:

```
function ShowExpSat($\mathcal{HC}$ : set of Horn clauses)
    nondet. choose clause $(C \wedge p_1(\bar{t}_1) \wedge \cdots \wedge p_n(\bar{t}_n) \rightarrow \mathit{false}) \in \mathcal{HC}$
    for $i = 1, \ldots, n$ do
        nondet. choose Boolean values $\bar{a}_i \in \mathbb{B}^{\alpha(p_i)}$
        ShowLiteral($p_i$, $\bar{a}_i$, $\mathcal{HC}$)
    end for
    Assume $C \wedge \bigwedge_{i=1}^n \bar{t}_i = \bar{a}_i$ is satisfiable
    return $\mathcal{HC}$ is satisfiable
end function

procedure ShowLiteral($p : \mathcal{R}$, $\bar{a} : \mathbb{B}^{\alpha(p)}$, $\mathcal{HC}$ : set of Horn clauses)
    nondet. choose clause $(C \wedge p_1(\bar{t}_1) \wedge \cdots \wedge p_n(\bar{t}_n) \rightarrow p(\bar{t})) \in \mathcal{HC}$
    for $i = 1, \ldots, n$ do
        nondet. choose Boolean values $\bar{a}_i \in \mathbb{B}^{\alpha(p_i)}$
        ShowLiteral($p_i$, $\bar{a}_i$, $\mathcal{HC}$)
    end for
    Assume $C \wedge \bar{t} = \bar{a} \wedge \bigwedge_{i=1}^n \bar{t}_i = \bar{a}_i$ is satisfiable
end procedure
```

Fig. 4 Algorithm for proving satisfiability of a formula $\mathit{exp}(\mathcal{HC})$ with polynomial space.

- a literal x_i of a clause C_j in F becomes a Horn clause
  $x_i = t_1 \rightarrow C_{i,j}(x_1, x_2, \ldots, x_{i-1}, t_1, x_{i+1}, \ldots, x_n)$
- a literal $\neg x_i$ of a clause C_j in F becomes a Horn clause
  $x_i = t_0 \rightarrow C_{i,j}(x_1, x_2, \ldots, x_{i-1}, t_0, x_{i+1}, \ldots, x_n)$
- a clause C_j in F becomes a set of Horn clauses
  $C_{1,j}(x_1, \ldots) \rightarrow C_j(x_1, \ldots), \quad C_{2,j}(x_1, \ldots) \rightarrow C_j(x_1, \ldots), \quad \ldots$
- the body F becomes the Horn clause
  $C_1(x_1, \ldots) \wedge C_2(x_1, \ldots) \wedge \cdots \rightarrow F_n(x_1, \ldots)$
- a quantifier $Q_i = \exists$ is translated as the two clauses
  $F_{i+1}(x_1, \ldots, x_{i-1}, 0) \rightarrow F_i(x_1, \ldots, x_{i-1}), \quad F_{i+1}(x_1, \ldots, x_{i-1}, 1) \rightarrow F_i(x_1, \ldots, x_{i-1})$
- a quantifier $Q_i = \forall$ is translated as the clause
  $F_{i+1}(x_1, \ldots, x_{i-1}, 0) \wedge F_{i+1}(x_1, \ldots, x_{i-1}, 1) \rightarrow F_i(x_1, \ldots, x_{i-1})$
- finally, we add the clause $F_1() \wedge t_0 \neq t_1 \rightarrow \mathit{false}$.

It is now easy to see that the expansion $\mathit{exp}(\mathcal{HC})$ of the Horn clauses coincides with the result of expanding all quantifiers in ϕ. By Theorem 3, unsatisfiability of the expansion is equivalent to solvability of the set of Horn clauses. □

Looking for upper bounds, it is easy to see that solvability of Horn clauses is in co-NEXPTIME for any constraint language with satisfiability problem in NP (for instance, quantifier-free Presburger arithmetic). This is because the size of the expansion $\mathit{exp}(\mathcal{HC})$ is at most exponential in the size of $\mathcal{HC}$. Individual constraint languages admit more efficient solvability checks:

Theorem 4 *Semantic solvability of recursion-free Horn clauses over the constraint language of Booleans is PSPACE-complete.*

Proof In combination with Lemma 2, it suffices to show that solvability of a set $\mathcal{HC}$ of recursion-free Horn clauses is in PSPACE. This can be done by constructing an algorithm ShowExpSat($\mathcal{HC}$) for checking satisfiability of *exp*($\mathcal{HC}$), and proving that the algorithm runs in polynomial space. Importantly, this satisfiability check can be done without explicit construction of the (worst-case exponentially big) formula *exp*($\mathcal{HC}$).

The algorithm is shown in Fig. 4, and is kept non-deterministic for reasons of presentation; it could easily be made deterministic by turning non-deterministic choices into explicit loops. $\alpha(p)$ in the algorithm denotes the arity of a relation symbol p.

To see that the algorithm runs in polynomial space, first observe that the recursion depth of the procedures is bounded by the number of available relation symbols, and thus by the size of $\mathcal{HC}$. At each recursive call, only data linear in the size of one clause in $\mathcal{HC}$ has to be stored, so that the overall memory consumption is linear in the size of $\mathcal{HC}$. □

8.2 Complexity for Presburger Arithmetic Constraints

Constraint languages that are more expressive than the Booleans lead to a significant increase in the complexity of solving Horn clauses. The lower bound in the following theorem can be shown by simulating time-bounded non-deterministic Turing machines.

Theorem 5 *Semantic solvability of head-disjoint recursion-free Horn clauses over the constraint language of quantifier-free Presburger arithmetic is co-NEXPTIME-complete.*

Proof It has already been observed that solvability is in co-NEXPTIME, so we proceed to show hardness by direct reduction of exponential-time-bounded Turing machines (possibly non-deterministic, with binary tape) to head-disjoint Horn clauses over quantifier-free PA. A Turing machine $M = (Q, \delta, q_0, F)$ is defined by

- a finite non-empty set Q of states,
- an initial state $q_0 \in Q$,
- a final state $f \in Q$,
- a transition relation $\delta \subseteq ((Q \setminus \{f\}) \times \{0, 1\}) \times (Q \times \{0, 1\} \times \{L, R\})$.

Wlog, we assume that $Q = \{0, 1, \ldots, f\} \subseteq \mathbb{Z}$ and $q_0 = 0$.

We define a relation symbol $\mathit{step}(q, l, r, q', l', r')$ to represent single execution steps of the machine. The parameters l, r, l', r' represent the tape, which is encoded as non-negative integers; the bits in the binary representation of the integers are the contents of the tape cells. l is the tape left of the head, r the tape right of the head. The least-significant bit of r is the tape cell at the head position. l', r' are the corresponding post-state variables after one execution step.

A tuple $(q, b, q', b', L) \in \delta$ (moving the tape to the left) is represented by a clause

$$\mathit{step}(q,\ x,\ b + 2y,\ q',\ b' + 2x,\ y) \tag{3}$$

where x, y are implicitly universally quantified variables of the clause, and q, b, q', b' concrete numeric constants. Similarly, a tuple $(q, b, q', b', R) \in \delta$ is encoded as

$$0 \leq x \leq 1 \rightarrow step(q,\ x + 2y,\ b + 2z,\ q',\ y,\ x + 2b' + 4z) \tag{4}$$

Finally, to represent termination, we add a clause

$$step(f, x, y, f, x, y) \tag{5}$$

implying that the machine will stay in the final state f forever. Note that clauses (3), (4), (5) can be merged to a single clause to establish head-disjointness.

We then introduce n further clauses to model an execution sequence of length 2^n:

$$\begin{aligned} step(x, y, z, x', y', z') \wedge step(x', y', z', x'', y'', z'') &\rightarrow step^1(x, y, z, x'', y'', z'') \\ step^1(x, y, z, x', y', z') \wedge step^1(x', y', z', x'', y'', z'') &\rightarrow step^2(x, y, z, x'', y'', z'') \\ &\dots \\ step^{n-1}(x, y, z, x', y', z') \wedge step^{n-1}(x', y', z', x'', y'', z'') &\rightarrow step^n(x, y, z, x'', y'', z'') \end{aligned}$$

The final clauses expresses that the Turing machine does not terminate within 2^n steps, when started with the initial tape t: $step^n(0, 0, t, f, x, y) \rightarrow false$.

Clearly, the expansion $exp(\mathcal{HC})$ of the resulting set $\mathcal{HC}$ of Horn clauses is unsatisfiable (i.e., $\mathcal{HC}$ can be solved) if and only if no execution of the Turing machine, starting with the initial tape t, terminates within 2^n steps. □

The lower bounds in Lemma 2 and Theorem 5 hinge on the fact that sets of Horn clauses can contain shared relation symbols in bodies. Neither result holds if we restrict ourselves to body-disjoint Horn clauses, which correspond to disjunctive interpolation as introduced in Sect. 6. Since the expansion $exp(\mathcal{HC})$ of body-disjoint Horn clauses is linear in the size of the set of Horn clauses, also solvability can be checked efficiently:

Theorem 6 *Semantic solvability of a set of body-disjoint Horn clauses, and equivalently the existence of a solution for a disjunctive interpolation problem, is in co-NP when working over the constraint languages of Booleans and quantifier-free Presburger arithmetic.*

Body-disjoint Horn clauses are still expressive: they can directly encode acyclic control-flow graphs, as well as acyclic unfolding of many simple recursion patterns.

A similar complexity result holds for linear Horn clauses:

Lemma 3 *Semantic solvability of recursion-free linear Horn clauses is in co-NP when working over the constraint languages of Booleans and quantifier-free Presburger arithmetic..*

Proof A set $\mathcal{HC}$ of recursion-free linear Horn clauses is solvable if and only if the expansion $exp(\mathcal{HC})$ is unsatisfiable. For linear clauses, $exp(\mathcal{HC})$ is a disjunction of (possibly) exponentially many formulae, each of which is linear in the size of $exp(\mathcal{HC})$. Consequently, satisfiability of $exp(\mathcal{HC})$ is in NP, and unsatisfiability in co-NP. □

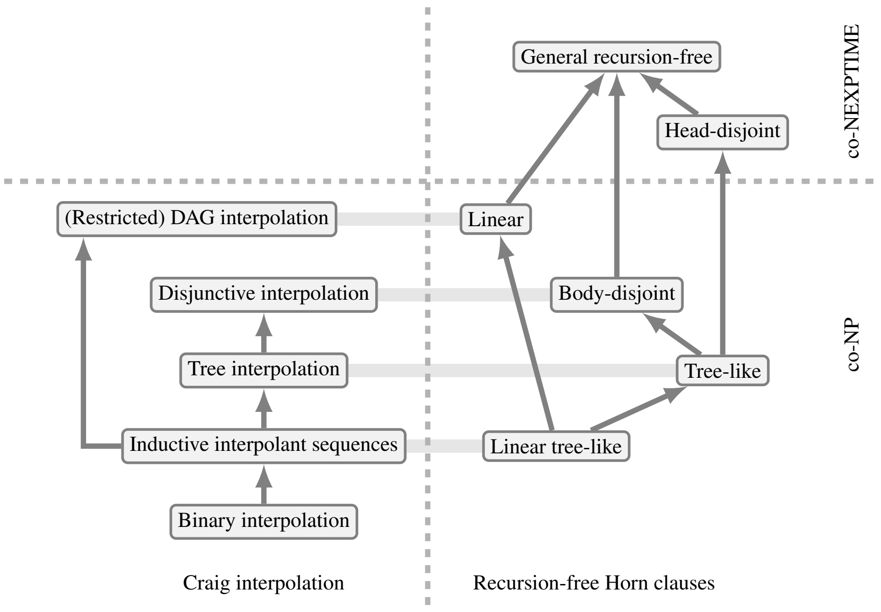


Fig. 5 Relationship between different forms of Craig interpolation, and different fragments of recursion-free Horn clauses. An arrow from A to B expresses that problem A is (strictly) subsumed by B. The complexity classes "co-NP" and "co-NEXPTIME" refer to the problem of checking solvability of Horn clauses over quantifier-free Presburger arithmetic.

We conclude by giving an overview of the various fragments of recursion-free Horn clauses, and the corresponding interpolation problem, in Fig. 5. The diagram also shows the complexity of deciding (semantic or syntactic) solvability of a set of Horn clauses, for Horn clauses over the constraint language of quantifier-free Presburger arithmetic.

9 Model Checking with Recursive Horn Clauses

Where *recursion-free* Horn clauses generalise the concept of Craig interpolation, solving *recursive* Horn clauses corresponds to the verification of general programs with loops, recursion, or concurrency features [16]. Procedures to solve recursion-free Horn clauses can serve as a building block within model checking algorithms for recursive Horn clauses [16], and are used to construct or refine abstractions by analysing spurious counterexamples. In particular, our disjunctive interpolation can be used for this purpose, and offers a high degree of flexibility due to the possibility to analyse counterexamples combining multiple execution traces. We illustrate the use of disjunctive interpolation within a predicate abstraction-based algorithm for solving Horn clauses. Our model checking algorithm is similar in spirit to the procedure in [16], and is explained in Sect. 9.1.

And/or trees of clauses. For sake of presentation, in our algorithm we represent counterexamples (i.e., recursion-free sets of Horn clauses) in the form of and/or trees labelled with clauses. Such trees are defined by the following grammar:

$$\mathit{AOTree} \ ::= \ \mathit{And}(h, \mathit{AOTree}, \ldots, \mathit{AOTree}) \ \mid \ \mathit{Or}(\mathit{AOTree}, \ldots, \mathit{AOTree})$$

where h ranges over (possibly recursive) Horn clauses. We only consider well-formed trees, in which the children of every *And*-node have head symbols that are consistent with the body literals of the clause stored in the node, and the sub-trees of an *Or*-node all have the same head symbol. And/or trees are turned into body-disjoint recursion-free sets of clauses by renaming relation symbols appropriately.

Example 6 The clauses in Fig. 2 can be represented by the following and/or tree (referring to clauses in Fig. 1).

$$\mathit{And}\Big((4),\ \mathit{Or}(\mathit{And}((1)),\ \mathit{And}((2),\ \mathit{And}((1))),\ \mathit{And}((3), \mathit{And}((1))))\Big)$$

Solving and/or dags. Counterexamples extracted from model checking problems often assume the form of and/or *dags*, rather than and/or *trees*. Since and/or-dags correspond to Horn clauses that are not body-disjoint, the complexity-theoretic results of the last section imply that it is in general impossible to avoid the expansion of and/or-dags to and/or-trees; there are, however, various effective techniques to speed-up handling of and/or-dags (related to the techniques in [30]). We highlight two of the techniques we use in our interpolation engine Princess [8], which we used in our experimental evaluation of the next section:

1) counterexample-guided expansion iteratively considers only parts of the fully expanded and/or dag, until an unsatisfiable fragment of the fully expanded tree has been found; such a fragment is sufficient to compute a solution. Counterexamples are useful in two ways: they can determine which or-branch of an and/or-dag is still satisfiable and has to be expanded further, but also whether it is necessary to create further copies of a shared subtree.

2) and/or dag restructuring factors out some common sub-dag s underneath an *Or*-node, making the and/or-dag more tree-like. This transformation corresponds to the following rewriting of and/or-dags, for a suitable fresh relation symbol q, and can significantly reduce the interpolation effort:

$$\begin{aligned}
&\mathit{Or}\Big(\mathit{And}(p(\bar{x}) \wedge B \to r(\bar{y}),\ s,\ \bar{t}),\ \mathit{And}(p(\bar{x}) \wedge B' \to r(\bar{y}),\ s,\ \bar{t}')\Big) \\
\rightsquigarrow\ \ &\mathit{And}\Big(p(\bar{x}) \wedge q(\ldots) \to r(\bar{y}),\ s,\ \mathit{Or}(\mathit{And}(B \to q(\ldots), \bar{t}),\ \mathit{And}(B' \to q(\ldots), \bar{t}'))\Big)
\end{aligned}$$

9.1 A Predicate Abstraction-based Model Checking Algorithm

Our model checking algorithm is in Fig. 6, and similar in spirit as the procedure in [16]; it has been implemented in the model checker Eldarica.[2] Solutions for Horn

[2] `http://lara.epfl.ch/w/eldarica`

clauses are constructed in disjunctive normal form by building an abstract reachability graph over a set of given predicates. When a counterexample is detected (a clause with consistent body literals and head *false*), a theorem prover is used to verify that the counterexample is genuine; spurious counterexamples are eliminated by generating additional predicates by means of disjunctive interpolation.

In Fig. 6, $\Pi : \mathcal{R} \to \mathcal{P}_{\text{fin}}(\mathit{Constr})$ denotes a mapping from relation symbols to the current (finite) set of predicates used to approximate the relation symbol. Given a (possibly recursive) set $\mathcal{HC}$ of Horn clauses, we define an *abstract reachability graph* (ARG) as a hyper-graph (S, E), where

– $S \subseteq \{(p, Q) \mid p \in \mathcal{R}, Q \subseteq \Pi(p)\}$ is the set of nodes, each of which is a pair consisting of a relation symbol and a set of predicates.
– $E \subseteq S^* \times \mathcal{HC} \times S$ is the hyper-edge relation, with each edge being labelled with a clause. An edge $E(\langle s_1, \ldots, s_n\rangle, h, s)$, with $h = (C \wedge B_1 \wedge \cdots \wedge B_n \to H) \in \mathcal{HC}$, implies that
  – $s_i = (p_i, Q_i)$ and $B_i = p_i(\bar{t}_i)$ for all $i = 1, \ldots, n$, and
  – $s = (p, Q)$, $H = p(\bar{t})$, and $Q = \{\phi \in \Pi(p) \mid C \wedge Q_1[\bar{t}_1] \wedge \cdots \wedge Q_n[\bar{t}_n] \models \phi[\bar{t}]\}$, where we write $Q_i[\bar{t}_i]$ for the conjunction of the predicates Q_i instantiated for the argument terms t_i.

An ARG (S, E) is called *closed* if the edge relation represents all Horn clauses in $\mathcal{HC}$. This means, for every clause $h = (C \wedge p_1(\bar{t}_1) \wedge \cdots \wedge p_n(\bar{t}_n) \to H) \in \mathcal{HC}$ and every sequence $(p_1, Q_1), \ldots, (p_n, Q_n) \in S$ of nodes one of the following properties holds:

– $C \wedge Q_1[\bar{t}_1] \wedge \cdots \wedge Q_n[\bar{t}_n] \models \mathit{false}$, or
– there is an edge $E(\langle(p_1, Q_1), \ldots, (p_n, Q_n)\rangle, C, s)$ such that $s = (p, Q)$, $H = p(\bar{t})$, and $Q = \{\phi \in \Pi(p) \mid C \wedge Q_1[\bar{t}_1] \wedge \cdots \wedge Q_n[\bar{t}_n] \models \phi[\bar{t}]\}$.

Lemma 4 *A set $\mathcal{HC}$ of Horn clauses has a closed ARG (S, E) if and only if $\mathcal{HC}$ is syntactically solvable.*

Proof "⇒": Define each relation symbol p as the disjunction $\bigvee_{(p,Q) \in S} \bigwedge Q$. Since S is closed under the edge relation, this yields a solution for the set $\mathcal{HC}$ of Horn clauses.

"⇐": Suppose $\mathcal{HC}$ is syntactically solvable, with each relation symbol p being mapped to the constraint C_p. We define the predicate abstraction $\Pi(p) = \{C_p\}$, and construct the ARG with nodes $S = \{(p, C_p)\}$, and the maximum edge relation E, which is closed. □

The function ExtractCEX extracts an and/or-tree representing a set of counterexamples, which can be turned into a recursion-free body-disjoint set of Horn clauses, and solved as described in Sect. 6.1. In general, the tree contains both conjunctions (from clauses with multiple body literals) and disjunctions, generated when following multiple hyper-edges (the case $|T| > 1$). Disjunctions make it possible to eliminate multiple counterexamples simultaneously. The algorithm is parametric in the precise strategy used to compute counterexamples (represented as non-deterministic choice in the pseudo code). The strategies we evaluated in the experiments (shown in the next section) are:

```
$S := \emptyset,\ E := \emptyset,\ \Pi := \{p \mapsto \emptyset \mid p \in \mathcal{R}\}$        ▷ Empty graph, no predicates
function ConstructARG
    while true do
        nondet. choose clause $h = (C \wedge p_1(\bar{t}_1) \wedge \cdots \wedge p_n(\bar{t}_n) \to H) \in \mathcal{HC}$
            and nodes $(p_1, Q_1), \ldots, (p_n, Q_n) \in S$
            such that $\neg\exists s.\ (\langle (p_1, Q_1), \ldots, (p_n, Q_n)\rangle, h, s) \in E$
            and $C \wedge Q_1[\bar{t}_1] \wedge \cdots \wedge Q_n[\bar{t}_n] \not\models false$
        if no such clauses and nodes exist then return $\mathcal{HC}$ is solvable
        end if
        if $H = false$ then        ▷ Refinement needed
            $tree := And(h, \text{ExtractCEX}(p_1, Q_1), \ldots, \text{ExtractCEX}(p_n, Q_n)$
            if $tree$ is unsatisfiable then
                extract disjunctive interpolant from $tree$, add predicates to $\Pi$
                delete part of $(S, E)$ used to construct $tree$
            else return $\mathcal{HC}$ is unsolvable, with counterexample trace $tree$
            end if
        else        ▷ Add edge to ARG
            then $H = p(\bar{t})$
            $Q := \{\phi \in \Pi(p) \mid \{C\} \cup Q_1 \cup \ldots \cup Q_n \models \phi\}$
            $e := (\langle (p_1, Q_1), \ldots, (p_n, Q_n)\rangle, h, (p, Q))$
            $S := S \cup \{(p, Q)\},\ E := E \cup \{e\}$
        end if
    end while
end function

function ExtractCEX($root : S$)        ▷ Extract disjunctive interpolation problem
    nondet. choose $\emptyset \neq T \subseteq E$ with $\forall e \in T.\ e = (\_, \_, root)$
    return $Or\{\ And(h, \text{ExtractCEX}(s_1), \ldots, \text{ExtractCEX}(s_n)) \mid$
            $(\langle s_1, \ldots, s_n\rangle, h, root) \in T\ \}$
end function
```

Fig. 6 Algorithm for construction of abstract reachability graphs.

TI extraction of a single counterexamples with minimal depth
(which means that disjunctive interpolation reduces to **T**ree **I**nterpolation), and

DI simultaneous extraction of all counterexamples with minimal depth
(so that genuine **D**isjunctive **I**nterpolation is used).

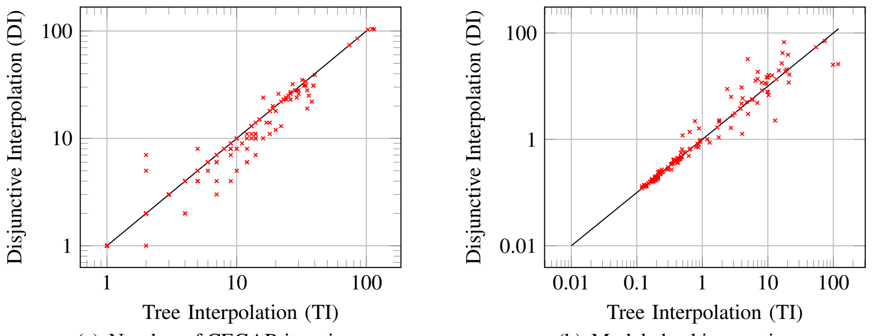
Example 7 We consider the Horn clauses given in Fig. 1, Sect. 2. Starting with an empty predicate map Π, the function ConstructARG will construct the reachability graph shown on the right (edges are labelled with the clauses from Fig. 1). Since *false* is reachable, function ExtractCEX will be called to extract a counterexample; possible results of executing ExtractCEX include:

$$\begin{tikzcd}
\arrow[d, "(1)"] \\
(\mathsf{gcd}, \emptyset) \arrow[loop left, "(2)"] \arrow[loop right, "(3)"] \arrow[d, "(4)"] \\
false
\end{tikzcd}$$

$$tree_1 \;=\; And((4),\ And((1))),$$

# Benchmarks	# Solved TI	DI	Z3
(a) Loop Invariants Using Abduction [11]			
46	30	31	35
(b) Control Flow and Integer Variables [6]			
13	9	13	13
(c) Benchmarks from HSF [16]			
15	10	11	11
(d) Benchmarks from [27]			
56	56	56	53
(e) Benchmarks from Reve [12]			
72	46	48	24

Fig. 7 Number of solved benchmarks for TI = Tree Interpolation, DI = Disjunctive Interpolation and Z3.



(a) Number of CEGAR iterations (b) Model checking runtime

Fig. 8 Comparison of the number of required refinement steps, and the runtime (in seconds), for the case of single counterexamples (**TI**) and simultaneous extraction of all minimal-depth counterexamples (**DI**). All experiments were done on an AMD Opteron$^{\mathrm{TM}}$Processor 6220 16-core machine with 3GHz and 32Gb of memory. The timeout is 2 minutes.

$$tree_2 \;=\; And((4),\; Or(And((1)),\; And((2),And((1))),\; And((3),And((1)))))$$

The counterexample $tree_2$ corresponds to the clauses shown in Fig. 2. Elimination of this counterexample with the help of disjunctive interpolation yields the predicates discussed in Example 5, which are sufficient to construct a closed ARG.

9.2 Experimental Evaluation

We have evaluated our algorithm on a library of more than 200 benchmarks in integer linear arithmetic from 5 different sources: benchmarks that are used in inductive loop invariant generation using abduction [11], control flow and integer variables programs of the International Competition on Software Verification (SVCOMP) [6], the library of benchmarks from the HSF tool [16], benchmarks for consistency analysis of decision-making programs [27], and automatic regression benchmarks [12].

Table 7 gives the total number of benchmarks that the approaches of tree interpolation and disjunctive interpolation can solve in each category. It also gives the number of successfully handled benchmarks for Z3[3]. Scatter plots comparing the results for the **T**ree **I**nterpolation and **D**isjunctive **I**nterpolation runs are given in Fig. 8. These plots compare the execution time and the number of iterations for the benchmarks for which both **TI** and **DI** succeed.

The total number of solved problems within the time bounds in Table 7 reveals that in 4 out of 5 categories **DI** has solved more benchmarks comparing to **TI** and in one category both approaches could solve all the benchmarks. This means that in general **DI** is able to converge faster with a smaller number of abstraction refinement steps. Studying the results more closely for the benchmarks that both approaches succeed, we observed that **DI** consistently led to a smaller number of abstraction refinement steps as the scatter plot in Fig. 8 shows. **DI** is indeed able to eliminate multiple counterexamples simultaneously, and to rapidly generate predicates that are useful for abstraction. The experiments also showed that there is a trade-off between the time spent generating predicates, and the quality of the predicates. In some benchmarks **DI** was slower than **TI**, despite fewer refinement steps. This may change as we make further improvements to our prototype implementation of disjunctive interpolation.

10 Conclusions

We have given a systematic study of different forms of Craig interpolation, the connection between Craig interpolation and fragments of recursion-free Horn clauses, as well the complexity for solving constraints. In addition, we introduced disjunctive interpolation as a new form of Craig interpolation tailored to model checkers based on Horn clauses. Disjunctive interpolation can be identified as solving body-disjoint systems of recursion-free Horn clauses, and subsumes a number of previous forms of interpolation, including tree interpolation. We believe that the flexibility of disjunctive interpolation is highly beneficial for building interpolation-based model checkers.

Acknowledgements We would like to thank Shaz Qadeer for discussions about the complexity of checking bounded recursive programs.

References

1. Albarghouthi, A., Gurfinkel, A., Chechik, M.: Craig interpretation. In: SAS (2012)
2. Albarghouthi, A., Gurfinkel, A., Chechik, M.: Whale: An interpolation-based algorithm for inter-procedural verification. In: VMCAI, pp. 39–55 (2012)
3. Ball, T., Podelski, A., Rajamani, S.K.: Relative completeness of abstraction refinement for software model checking. In: TACAS'02, *LNCS*, vol. 2280, p. 158 (2002)
4. Banda, G., Gallagher, J.P.: Analysis of linear hybrid systems in clp. In: Hanus [20], pp. 55–70. DOI 10.1007/978-3-642-00515-2_5
5. Beyene, T.A., Popeea, C., Rybalchenko, A.: Solving existentially quantified Horn clauses. In: CAV (2013)
6. Beyer, D.: Status report on software verification - (competition summary sv-comp 2014). In: TACAS, pp. 373–388 (2014)

[3] `http://z3.codeplex.com`, version 4.3.2

7. Bjørner, N., McMillan, K.L., Rybalchenko, A.: On solving universally quantified Horn clauses. In: SAS (2013)
8. Brillout, A., Kroening, D., Rümmer, P., Wahl, T.: An interpolating sequent calculus for quantifier-free Presburger arithmetic. Journal of Automated Reasoning **47**, 341–367 (2011). URL `http://dx.doi.org/10.1007/s10817-011-9237-y`
9. Cimatti, A., Griggio, A., Sebastiani, R.: Efficient generation of Craig interpolants in satisfiability modulo theories. ACM Trans. Comput. Log. **12**(1), 7 (2010)
10. Craig, W.: Linear reasoning. A new form of the Herbrand-Gentzen theorem. The Journal of Symbolic Logic **22**(3), 250–268 (1957)
11. Dillig, I., Dillig, T., Li, B., McMillan, K.L.: Inductive invariant generation via abductive inference. In: OOPSLA, pp. 443–456 (2013)
12. Felsing, D., Grebing, S., Klebanov, V., Rümmer, P., Ulbrich, M.: Automating regression verification. In: I. Crnkovic, M. Chechik, P. Grünbacher (eds.) ACM/IEEE International Conference on Automated Software Engineering, ASE, pp. 349–360. ACM (2014)
13. Fioravanti, F., Pettorossi, A., Proietti, M., Senni, V.: Generalization strategies for the verification of infinite state systems. TPLP **13**(2), 175–199 (2013). DOI 10.1017/S1471068411000627
14. Godefroid, P., Yannakakis, M.: Analysis of boolean programs. In: TACAS, pp. 214–229 (2013)
15. Graf, S., Saidi, H.: Construction of abstract state graphs with PVS. In: CAV (1997)
16. Grebenshchikov, S., Lopes, N.P., Popeea, C., Rybalchenko, A.: Synthesizing software verifiers from proof rules. In: PLDI (2012)
17. Gupta, A., Popeea, C., Rybalchenko, A.: Predicate abstraction and refinement for verifying multi-threaded programs. In: POPL (2011)
18. Gupta, A., Popeea, C., Rybalchenko, A.: Solving recursion-free Horn clauses over LI+UIF. In: APLAS, pp. 188–203 (2011)
19. Gupta, A., Popeea, C., Rybalchenko, A.: Generalised interpolation by solving recursion-free Horn clauses. CoRR **abs/1303.7378** (2013)
20. Hanus, M. (ed.): Logic-Based Program Synthesis and Transformation, 18th International Symposium, LOPSTR 2008, Valencia, Spain, July 17-18, 2008, Revised Selected Papers, *Lecture Notes in Computer Science*, vol. 5438. Springer (2009)
21. Heizmann, M., Hoenicke, J., Podelski, A.: Nested interpolants. In: POPL (2010)
22. Henzinger, T.A., Jhala, R., Majumdar, R., McMillan, K.L.: Abstractions from proofs. In: POPL, pp. 232–244. ACM (2004)
23. Hoder, K., Bjørner, N.: Generalized property directed reachability. In: SAT (2012)
24. Hojjat, H., Iosif, R., Konečný, F., Kuncak, V., Rümmer, P.: Accelerating interpolants. In: Automated Technology for Verification and Analysis (ATVA) (2012)
25. Jhala, R., Majumdar, R., Rybalchenko, A.: HMC: Verifying functional programs using abstract interpreters. In: CAV (2011)
26. Kafle, B., Gallagher, J.P.: Convex polyhedral abstractions, specialisation and property-based predicate splitting in Horn clause verification. In: Workshop on Horn Clauses for Verification and Synthesis (2014)
27. Kincaid, Z.: LIA Horn benchmarks.
`https://svn.sosy-lab.org/software/sv-benchmarks/trunk/clauses/LIA/Zachary/`
28. Komuravelli, A., Gurfinkel, A., Chaki, S.: SMT-based model checking for recursive programs. In: A. Biere, R. Bloem (eds.) CAV, *Lecture Notes in Computer Science*, vol. 8559, pp. 17–34. Springer (2014)
29. Komuravelli, A., Gurfinkel, A., Chaki, S., Clarke, E.M.: Automatic abstraction in SMT-based unbounded software model checking. In: Sharygina and Veith [42], pp. 846–862
30. Lal, A., Qadeer, S., Lahiri, S.K.: Corral: A solver for reachability modulo theories. In: CAV (2012)
31. McMillan, K.L.: iZ3 documentation.
`http://research.microsoft.com/en-us/um/redmond/projects/z3/iz3documentation.html`
32. McMillan, K.L.: Interpolation and SAT-based model checking. In: CAV (2003)
33. McMillan, K.L.: Lazy abstraction with interpolants. In: CAV (2006)
34. McMillan, K.L., Rybalchenko, A.: Solving constrained Horn clauses using interpolation. Tech. Rep. MSR-TR-2013-6, Microsoft Research (2013)
35. Méndez-Lojo, M., Navas, J.A., Hermenegildo, M.V.: A flexible, (C)LP-based approach to the analysis of object-oriented programs. In: LOPSTR, pp. 154–168 (2007)
36. Peralta, J.C., Gallagher, J.P., Saglam, H.: Analysis of imperative programs through analysis of constraint logic programs. In: SAS (1998)

37. Rümmer, P., Hojjat, H., Kuncak, V.: Classifying and solving Horn clauses for verification. In: VSTTE (2013)
38. Rümmer, P., Hojjat, H., Kuncak, V.: Classifying and solving Horn clauses for verification. In: E. Cohen, A. Rybalchenko (eds.) VSTTE, *Lecture Notes in Computer Science*, vol. 8164, pp. 1–21. Springer (2013)
39. Rümmer, P., Hojjat, H., Kuncak, V.: Disjunctive interpolants for Horn-clause verification. In: Sharygina and Veith [42], pp. 347–363
40. Rümmer, P., Hojjat, H., Kuncak, V.: Disjunctive Interpolants for Horn-Clause Verification (Extended Technical Report). ArXiv e-prints (2013). `http://arxiv.org/abs/1301.4973`
41. Sery, O., Fedyukovich, G., Sharygina, N.: Interpolation-based function summaries in bounded model checking. In: Haifa Verification Conference, pp. 160–175 (2011)
42. Sharygina, N., Veith, H. (eds.): Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings, *Lecture Notes in Computer Science*, vol. 8044. Springer (2013)
43. Terauchi, T.: Dependent types from counterexamples. In: POPL, pp. 119–130 (2010)
44. Unno, H., Terauchi, T., Kobayashi, N.: Automating relatively complete verification of higher-order functional programs. In: POPL (2013)