Available online at www.sciencedirect.com
SCIENCE DIRECT®
Theoretical Computer Science 345 (2005) 101–121
Theoretical Computer Science
www.elsevier.com/locate/tcs

An interpolating theorem prover

K.L. McMillan

Cadence Berkeley Labs, 1995 University Ave. Suite 460, Berkeley, CA 94704, USA

Communicated by D. Sannella

Abstract

We present a method of deriving Craig interpolants from proofs in the quantifier-free theory of linear inequality and uninterpreted function symbols, and an interpolating theorem prover based on this method. The prover has been used for predicate refinement in the BLAST software model checker, and can also be used directly for model checking infinite-state systems, using interpolation-based image approximation.

Keywords: Craig interpolation; Model checking; Decision procedures; Infinite-state systems

1. Introduction

A Craig interpolant [2] for an inconsistent pair of logical formulas (A, B) is a formula ϕ that is implied by A, inconsistent with B and refers only to uninterpreted symbols common to A and B. If A and B are propositional formulas, and we are given a refutation of $A \wedge B$ by resolution steps, we can derive an interpolant for (A, B) in linear time [5,12]. This fact has been exploited in a method of over-approximate image computation based on interpolation [7]. This provides a complete symbolic method of model checking finite-state systems with respect to linear temporal properties. The method is based entirely on a proof-generating Boolean satisfiability solver and does not rely on quantifier elimination or reduction to normal forms such as binary decision diagrams (BDDs) or conjunctive normal form. In practice it was found to be highly effective in proving localizable properties of large sequential circuits.

E-mail address: mcmillan@cadence.com.

0304-3975/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2005.07.003

Here we present a first step in expanding this approach from propositional to first-order logic, and from finite-state to infinite-state systems. We present an interpolating prover for a quantifier-free theory that includes linear inequalities and equality with uninterpreted function symbols. As in [3] the prover combines a Boolean satisfiability solver with a proof-generating decision procedure for ground clauses. After generating a refutation for $A \wedge B$, the prover derives from this refutation an interpolant ϕ for the pair (A, B). The main contribution of this work is to show how to derive quantifier-free interpolants from proofs in the combined theories of linear inequality and equality with uninterpreted function symbols (LIUF). This extends earlier work that handles only linear inequalities [12]. The combination of theories is useful, for example, for applications in software model checking.

It is important to note that we are deriving quantifier-free interpolants from quantifier-free formulas. As we will observe later, this is crucial for applications in formal verification, such as image approximation and predicate abstraction. In Craig's original work on interpolants [2], unwanted individual symbols were eliminated by simply quantifying them. Here, we must take a different approach, to avoid introducing quantifiers in the interpolants.

The interpolating prover has been applied in the BLAST software model checking system [4]. This system is based on predicate abstraction [13], and uses interpolants as a guide in generating new predicates for abstraction refinement. The approach resulted in a substantial reduction in abstract state space size relative to earlier methods. Further, using the method of [7], the prover can be used directly to verify some infinite-state systems, such as the Fischer and "bakery" mutual exclusion protocols. In principle, it can also be applied to the model checking phase of predicate abstraction.

This paper is organized as follows. In Section 2, we introduce a simple proof system for LIUF, and show how refutations in this system can be translated into interpolants. Section 3 discusses the practicalities of constructing an efficient interpolating prover using this system. Finally, Section 4 discusses actual and potential applications of the interpolating prover.

2. Interpolants from proofs

We now describe a system of rules that, given a refutation of a pair of clause sets (A, B), derive an interpolant ϕ for the pair. For the sake of simplicity, we begin with a quantifier-free logic with linear inequalities (LI). Then we treat a logic with equality and uninterpreted functions (EUF). Finally, we combine the two theories.

2.1. Linear inequalities

A *term* in this logic is a linear combination $c_0 + c_1 v_1 + \cdots c_n v_n$, where $v_1, \ldots, v_n$ are distinct individual variables, $c_0 \ldots c_n$ are rational constants, and further $c_1 \ldots c_n$ are non-zero. When we perform arithmetic on terms, we will assume they are reduced to this normal form. That is, if x is a term and c is a non-zero constant, we will write cx to denote the term obtained by distributing the coefficient c inside x. Similarly, if x and y are terms, we will write $x + y$ to denote the term obtained by summing like terms in x and y and dropping resulting terms with zero coefficients. Thus, for example, if x is the term $1 + a$ and y is the term $b - 2a$ then $2x + y$ would denote the term $2 + b$.

An *atomic predicate* in the logic is either a propositional variable or an inequality of the form $0 \leqslant x$, where x is a term. A *literal* is either an atomic predicate or its negation. A *clause* is a disjunction of literals. We will write the clause containing the set of literals Γ as $\langle \Gamma \rangle$. In particular, we will distinguish syntactically between a literal l and the clause $\langle l \rangle$ containing just l. The empty clause, equivalent to false, will be written $\langle\rangle$.

A *sequent* is of the form $\Gamma \vdash \Delta$, where Γ and Δ are sets of formulas (in this case, either literals or clauses). The interpretation of $\Gamma \vdash \Delta$ is that the conjunction of the formulas in Γ entails the disjunction of the formulas in Δ. In what follows, lower case letters generally stand for formulas and upper case letters for sets of formulas. Further, a formula in a place where a set is expected should be taken as the singleton containing it, and a list of sets should be taken as their union. Thus, for example, the expression $\Gamma, \phi \vdash p, A$ should be taken as an abbreviation for $\Gamma \cup \{\phi\} \vdash \{p\} \cup A$.

Our theorem prover generates refutations for sets of clauses using the following proof rules: [1]

$$\textsc{Hyp}\ \frac{}{\Gamma \vdash \phi}\ \phi \in \Gamma, \qquad \textsc{Comb}\ \frac{\Gamma \vdash 0 \leqslant x \quad \Gamma \vdash 0 \leqslant y}{\Gamma \vdash 0 \leqslant c_1 x + c_2 y}\ c_{1,2} > 0,$$

$$\textsc{Contra}\ \frac{\neg p_1, \ldots, \neg p_n \vdash \ \bot}{\Gamma \vdash \langle p_1, \ldots, p_n \rangle}\ddagger, \qquad \textsc{Res}\ \frac{\Gamma \vdash \langle p, \Theta \rangle \quad \Gamma \vdash \langle \neg p, \Theta' \rangle}{\Gamma \vdash \langle \Theta, \Theta' \rangle}.$$

In the above, $\bot$ is a shorthand for $0 \leqslant -1$ (note this is semantically equivalent but not identical to $\langle\rangle$). Also, in the CONTRA rule, the symbol ‡ indicates that all atomic predicates occurring on the right hand side of the consequent must occur on the left. This requirement is not needed for soundness, but our interpolation rules will rely on it. In effect, it prevents us from introducing new atomic predicates in the proof, thus ensuring that proofs are cut-free. All Boolean reasoning is done by the resolution rule RES.

We will use the notation $\phi \preceq \Gamma$ to indicate that all variables and uninterpreted function symbols occurring in ϕ also occur in Γ. A term x is *local* with respect to a pair (A, B) if it contains a variable or uninterpreted function symbol not occurring in B (in other words $x \not\preceq B$) and *global* otherwise.

In order to represent the rules for deriving interpolants from proofs, we will define several classes of *interpolations*. These have the general syntactic form $(A, B) \vdash \psi\ [X]$, where the exact form of X varies. Intuitively, X is a representation of an "interpolant" associated with the deduction of ψ from A and B. In the case where ψ is the empty clause, X should in fact be an interpolant for (A, B). In general, X represents some fact that is derivable from A, and that together with B proves ψ.

For each class of interpolation, we will define a notion of validity. This definition consists of three conditions, corresponding to the three conditions for interpolants—the first ensures that A implies the interpolant, the second ensures that A and B together imply ψ, and the third ensures that the interpolant is over common variables. We will then introduce derivation rules that are sound, in the sense that they derive only valid interpolations

[1] Note, this system is not complete, since it has no rule to deal with negated inequalities. Later, after we introduce the equality operator, we will obtain a complete system for the rationals.

from valid interpolations. We will sketch a proof of soundness for each rule, except in trivial cases.

We begin with the derivation of inequalities. This is done by summing up inequalities drawn from A and B, using the COMB rule. As observed in [12], the contribution to this sum from A is effectively an interpolant. For example, suppose A contains $0 \leqslant w - x$ and $0 \leqslant x - y$, while B contains $0 \leqslant y - z$. Summing these, we obtain $0 \leqslant w - z$, which we will call ψ. The sum of the contributions from A is $0 \leqslant w - y$, which satisfies our conditions for an interpolant, since it is derivable from A, and along with B, gives us ψ. Moreover, notice that the coefficient of w is the same in the interpolant and in ψ. In general, the coefficients of any local variables in ψ and its interpolant must be equal, since these cannot be altered by adding inequalities from B. Thus, in particular, when we derive $0 \leqslant -1$, a contradiction, only variables common to A and B may appear (with non-zero coefficient) in the interpolant. This intuition is captured formally in the following definition:

Definition 1. An *inequality interpolation* has form $(A, B) \vdash 0 \leqslant x\ [x', \rho, \gamma]$, where A and B are sets of literals, x and x' are terms, and ρ and γ are formulas. It is said to be *valid* when
(1) $A, \rho \vDash 0 \leqslant x' \wedge \gamma$,
(2) $B \vDash \rho$ and $B, \gamma \vDash 0 \leqslant x - x'$, and
(3) $\rho, \gamma \preceq B$ and $x', \rho, \gamma \preceq A$ and $(x - x') \preceq B$.

For the current system, the formulas ρ and γ are always $\top$. They will play a role later, when we combine theories. The intuition behind this definition is that $0 \leqslant x$ is a linear combination of inequalities from A and B, where x' represents the contribution to x from A.

We now begin with the interpolation rules for introduction of hypotheses. Here, we distinguish two cases, depending on whether the hypothesis is from A or B:

$$\text{HYPLEQ-A}\frac{}{(A, B) \vdash 0 \leqslant x\ [x, \top, \top]}\ (0 \leqslant x) \in A,$$

$$\text{HYPLEQ-B}\frac{}{(A, B) \vdash 0 \leqslant x\ [0, \top, \top]}\ (0 \leqslant x) \in B.$$

The soundness of these rules (i.e., validity of their consequents, given the side conditions) is easily verified. The rule for combining inequalities is as follows:

$$\text{COMB}\frac{\begin{array}{c}(A, B) \vdash 0 \leqslant x\ [x', \rho, \gamma],\\ (A, B) \vdash 0 \leqslant y\ [y', \rho', \gamma']\end{array}}{(A, B) \vdash 0 \leqslant c_1 x + c_2 y\ [c_1 x' + c_2 y', \rho \wedge \rho', \gamma \wedge \gamma']} c_{1,2} > 0.$$

In effect, we derive the interpolant for a linear combination of inequalities by taking the same linear combination of the contributions from A. Again, the reader may wish to verify that the validity conditions for inequality interpolations are preserved by this rule.

Example 1. As an example, let us derive an interpolant for the case where A is $(0 \leqslant y - x)(0 \leqslant z - y)$ and B is $(0 \leqslant x - z - 1)$. For clarity, we will abbreviate $(A, B) \vdash \psi\ [x, \top, \top]$

to $\vdash \psi\ [x]$. We first use the HYPLEQ-A rule to introduce two hypotheses from A:

$$\text{HYPLEQ-A}\frac{}{\vdash 0 \leqslant y - x\ [y - x]}, \qquad \text{HYPLEQ-A}\frac{}{\vdash 0 \leqslant z - y\ [z - y]}.$$

Now, we sum these two inequalities using the COMB rule:

$$\text{COMB}\frac{\vdash 0 \leqslant y - x\ [y - x] \quad \vdash 0 \leqslant z - y\ [z - y]}{\vdash 0 \leqslant z - x\ [z - x]}.$$

Now, we introduce a hypothesis from B:

$$\text{HYPLEQ-B}\frac{}{\vdash 0 \leqslant x - z - 1\ [0]}.$$

Finally, we sum this with our previous result, to obtain $0 \leqslant -1$, which is false:

$$\text{COMB}\frac{\vdash 0 \leqslant z - x\ [z - x] \quad \vdash 0 \leqslant x - z - 1\ [0]}{\vdash 0 \leqslant -1\ [z - x]}.$$

You may want to check that all the interpolations derived are valid. Also notice that in the last step we have derived a contradiction, and that $0 \leqslant z - x$ is an interpolant for (A, B).

Now we turn to Boolean reasoning using the resolution rule. Constructions to produce linear-size interpolants from resolution proofs were first introduced in [5,12]. They differ slightly from the one used here, which derives from [7]. The basic idea is to reduce the resolution proof to a Boolean circuit in which each resolution step corresponds to a gate. In this circuit, resolutions on local predicates correspond to "or" gates, while resolutions on global predicates correspond to "and" gates.

The intuition behind this is as follows. A resolution step is a case split in the proof on some atomic predicate. If we split cases on a predicate unique to A, then A proves a disjunction of facts—one which holds in the positive case and the other in the negative. If we split cases on a predicate occurring in B, then B proves a disjunction of facts, both of which must be refuted by A, so A must prove a conjunction. As an example, suppose that A contains the clauses $\langle \neg a, b\rangle$, $\langle a, c\rangle$, while B contains $\langle \neg b\rangle$, $\langle \neg c\rangle$. To refute this pair, we might split cases on a. In the positive case, A implies b, which is refuted by B, while in the negative case A implies c, which is also refuted by B. Thus, $b \vee c$ is an interpolant. If we reverse the definitions of A and B, and again split cases on a (now a global proposition) we observe that B proves b in one case and c in the other, both of which are refuted by A. Thus A proves the conjunctive interpolant $\neg b \wedge \neg c$.

We now introduce an interpolation syntax for clauses. If Θ is a set of literals, we will denote by $\Theta \downarrow B$ the literals of Θ over atomic predicates occurring in B and by $\Theta \setminus B$ the literals of Θ over atomic predicates *not* occurring in B.

Definition 2. A *clause interpolation* has the form $(A, B) \vdash \langle \Theta\rangle\ [\phi]$, where A and B are clause sets, Θ is a literal set and ϕ is a formula. It is said to be *valid* when:
(1) $A \vDash \phi \vee \langle \Theta \setminus B\rangle$, and

(2) $B, \phi \vDash \langle \Theta \downarrow B \rangle$, and
(3) $\phi \preceq B$ and $\phi \preceq A$.

Notice that if Θ is empty, ϕ is an interpolant for (A, B). Notice also that the interpolant ϕ serves as a cut that localizes the proof of the clause $\langle \Theta \rangle$. If ϕ is false, then A proves $\langle \Theta \setminus B \rangle$, while if ϕ is true then B proves $\langle \Theta \downarrow B \rangle$.

Two rules are needed for introduction of clauses as hypotheses:

$$\textsc{HypC-A}\frac{}{(A, B) \vdash \langle \Theta \rangle\ [\langle \Theta \downarrow B \rangle]}\ \langle \Theta \rangle \in A,$$

$$\textsc{HypC-B}\frac{}{(A, B) \vdash \langle \Theta \rangle\ [\top]}\ \langle \Theta \rangle \in B.$$

Note that the derived interpolations are trivially valid, given the side conditions. Now, we introduce two interpolation rules for resolution of clauses. The first is for resolution on an atomic predicate *not* occurring in B:

$$\textsc{Res-A}\frac{\begin{array}{c}(A, B) \vdash \langle p, \Theta \rangle\ [\phi], \\ (A, B) \vdash \langle \neg p, \Theta' \rangle\ [\phi']\end{array}}{(A, B) \vdash \langle \Theta, \Theta' \rangle\ [\phi \vee \phi']}\ p \text{ not occurs in } B.$$

Soundness. For the first condition, we know that A implies $\phi \vee p \vee \langle \Theta \setminus B \rangle$ and $\phi' \vee \neg p \vee \langle \Theta' \setminus B \rangle$. By resolution on p we have A implies $(\phi \vee \phi') \vee \langle (\Theta, \Theta') \setminus B \rangle$. For the second condition, given B, we know that $\phi \Longrightarrow \langle \Theta \downarrow B \rangle$ and $\phi' \Longrightarrow \langle \Theta' \downarrow B \rangle$. Thus, $\phi \vee \phi'$ implies $\langle (\Theta, \Theta') \downarrow B \rangle$. The third condition is trivial.

The second rule is for resolution on an atomic predicate occurring in B:

$$\textsc{Res-B}\frac{\begin{array}{c}(A, B) \vdash \langle p, \Theta \rangle\ [\phi], \\ (A, B) \vdash \langle \neg p, \Theta' \rangle\ [\phi']\end{array}}{(A, B) \vdash \langle \Theta, \Theta' \rangle\ [\phi \wedge \phi']}\ p \text{ occurs in } B.$$

Soundness. For the first validity condition, we know that A implies $\phi \vee \langle \Theta \setminus B \rangle$ and $\phi' \vee \langle \Theta' \setminus B \rangle$. These in turn imply $(\phi \wedge \phi') \vee \langle (\Theta, \Theta') \setminus B \rangle$. For the second condition, given B, we know that $\phi \Longrightarrow p \vee \langle \Theta \downarrow B \rangle$ while $\phi' \Longrightarrow \neg p \vee \langle \Theta' \downarrow B \rangle$. By resolution, we have that $\phi \wedge \phi'$ implies $\langle (\Theta, \Theta') \downarrow B \rangle$. The third condition is trivial.

Example 2. As an example, we derive an interpolant for (A, B), where A is $\langle b \rangle$, $\langle \neg b \vee c \rangle$ and B is $\langle \neg c \rangle$. First, using the HypC-A rule, we introduce the two clauses from A as hypotheses:

$$\textsc{HypC-A}\frac{}{\vdash \langle b \rangle\ [\bot]} \quad \textsc{HypC-A}\frac{}{\vdash \langle \neg b, c \rangle\ [c]}.$$

We now resolve these two clauses on b.

$$\textsc{Res-A}\frac{\vdash \langle b \rangle\ [\bot] \qquad \vdash \langle \neg b, c \rangle\ [c]}{\vdash \langle c \rangle\ [\bot \vee c]}.$$

We then use the HYP-B rule to introduce the clause from B.

$$\text{HYP-B}\frac{}{\vdash \langle \neg c\rangle\ [\top]}.$$

Finally, we resolve the last two clauses on c. We use the RES-B rule, since c occurs in B.

$$\text{RES-B}\frac{\vdash \langle c\rangle\ [c] \quad \vdash \langle \neg c\rangle\ [\top]}{\vdash \langle\rangle\ [c \wedge \top]}.$$

Thus c is an interpolant for (A, B).

Finally, we introduce a rule to connect inequality reasoning to Boolean reasoning. In effect, we prove a tautology clause $\langle\Theta\rangle$ by deriving a contradiction from the set of the negations of its literals (which we will abbreviate as $\neg\Theta$). To obtain a clause interpolation, we first partition these literals into two subsets, $\neg\Theta \setminus B$ and $\neg\Theta \downarrow B$, which will take role of A and B, respectively in deriving the contradiction. The interpolant we obtain for this pair serves as the interpolant for the derivation of $A, B \vdash \langle\Theta\rangle$. Note that $\langle\Theta\rangle$ itself is a tautology and hence its proof does not depend on A or B. However, the interpolant we obtain depends on A and B, since these determine the partition of the literals in $\neg\Theta$. The interpolation rule is as follows:

$$\text{CONTRA}\frac{(\neg\Theta \setminus B, \neg\Theta \downarrow B) \vdash\ \bot\ [x', \rho, \gamma]}{(A, B) \vdash \langle\Theta\rangle\ \ [\rho \Longrightarrow (0 \leqslant x' \wedge \gamma)]}\ \ddagger,$$

where ‡ indicates that all atomic predicates occurring Θ must occur in A or B.

Soundness. Let ϕ be $\rho \Longrightarrow (0 \leqslant x' \wedge \gamma)$. By the first condition of Definition 1, $\neg\Theta \setminus B \vDash \phi$. Thus, by DeMorgan's laws, we have $\vDash \phi \vee \langle\Theta \setminus B\rangle$, satisfying the first validity condition. From the second condition of Definition 1, we know that $\neg\Theta \downarrow B \vDash \rho$, and $\neg\Theta \downarrow B, \gamma \vDash 0 \leqslant -1 - x'$. Thus, summing inequalities, we have $\neg\Theta \downarrow B, \phi \vDash 0 \leqslant -1$, so by DeMorgan's laws $\phi \vDash \langle\Theta \downarrow B\rangle$ holds, satisfying the second validity condition. Finally, the third validity condition is guaranteed by the third condition of Definition 1 and the side condition.

2.2. Equality and uninterpreted functions

In our logic of equality and uninterpreted functions, a term is either an individual variable or a function application $f(x_1, \ldots, x_n)$, where f is a n-ary function symbol and $x_1 \ldots x_n$ are terms. An atomic predicate is a propositional variable or an equality of the form $x = y$, where x and y are terms. In the sequel, we will use the notation $x \simeq y$ for syntactic equality of two meta-variables x and y, to distinguish this notion from the atomic predicate $x = y$.

Refutations in this theory are generated using the following proof rules (in addition to the HYP rule):

$$\text{REFL}\frac{}{\Gamma \vdash x = x}\ \dagger, \qquad \text{SYMM}\frac{\Gamma \vdash x = y}{\Gamma \vdash y = x},$$

$$\text{TRANS}\frac{\Gamma \vdash x = y \quad \Gamma \vdash y = z}{\Gamma \vdash x = z}, \qquad \text{CONG}\frac{\Gamma \vdash x_1 = y_1 \quad \ldots \quad \Gamma \vdash x_n = y_n}{\Gamma \vdash f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)}\ \dagger,$$

$$\text{EQNEQ}\,\frac{\Gamma \vdash x = y}{\Gamma \vdash \bot}\ \neg(x = y) \in \Gamma,$$

where † indicates that the terms equated in the consequent must occur in Γ. This requirement is not needed for soundness, but our interpolation rules will rely on it. Boolean reasoning can be added to the system by adding the CONTRA and RES rules of the previous system.

Now let us consider the problem of deriving interpolants from proofs using the transitivity rule. To derive $x = y$, we effectively build up a chain of equalities $\sigma \simeq (x = t_1)(t_1 = t_2)\cdots(t_n = y)$. Now suppose that these equalities are drawn from two sets, A and B, and suppose for the moment that at least one global term occurs in σ. We can make several observations. First, let ${}^\bullet\sigma$ stand for the leftmost global term in σ, and let $\sigma^\bullet$ stand for the rightmost global term in σ (with respect to (A, B)). We observe that A implies $x = {}^\bullet\sigma$ and $y = \sigma^\bullet$, since all the equalities to the left of ${}^\bullet\sigma$ and to the right of $\sigma^\bullet$ must come from A. Thus, A gives us solutions for x and y as global terms.

Moreover, consider the segment of σ between ${}^\bullet\sigma$ and $\sigma^\bullet$. The endpoints of this segment are by definition global terms. We can divide the segment into maximal subchains, consisting of only equalities from A, or only equalities from B. Each such subchain $(t_i = \cdots = t_j)$ can be summarized by the single equality $t_i = t_j$. Note that t_i and t_j must be global terms, since they are either ${}^\bullet\sigma$ or $\sigma^\bullet$, or are common between an A and a B subchain. Thus, if the subchain is derived from A, then t_i and t_j must be common to A and B. We will use γ to denote the conjunction of the summaries of the A subchains. We observe that γ is implied by A, and that B with γ implies ${}^\bullet\sigma = \sigma^\bullet$, and that γ contains only common symbols. Thus, we can say that γ is an interpolant for the derivation of $x = y$, under the global solutions we obtain for x and y.

We have not yet considered the case when σ contains no global terms. We will call this the *degenerate* case, and will say that by definition ${}^\bullet\sigma = y$ and $\sigma^\bullet = x$. In the degenerate case, our interpolant γ is just $\top$, and our solutions yield exactly $x = y$.

We are now ready to define an interpolation syntax for equalities, as follows:

Definition 3. An *equality interpolation* has form $(A, B) \vdash x = y\ [x', y', \rho, \gamma]$, where A and B are sets of literals, x, y, x', y' are terms, and ρ and γ are formulas. It is said to be *valid* when:

(1) $A, \rho \vDash x = x' \wedge y = y' \wedge \gamma$,
(2) $B \vDash \rho$, and
  (a) $x' \simeq y$ and $y' \simeq x$ (the *degenerate* case), or
  (b) $x', y' \preceq B$ and $B, \gamma \vDash x' = y'$,
(3) $\rho, \gamma \preceq B$ and $\rho, \gamma \preceq A$, and if $x \preceq B$ then $x' \simeq x$, else $x' \preceq A$, and similarly for y, y'.

Here, x' and y' take the roles of ${}^\bullet\sigma$ and $\sigma^\bullet$, respectively. For the case of transitivity proofs, ρ is always $\top$. The first condition says that A gives the solutions $x = {}^\bullet\sigma$ and $y = \sigma^\bullet$. The second says, in effect, that B along with the A subchains γ guarantees ${}^\bullet\sigma = \sigma^\bullet$ (except in the degenerate case). In the degenerate case, A entails $x = y$ by itself. The third condition contains some invariants that are necessary for soundness of the transitivity rule, as we shall observe shortly.

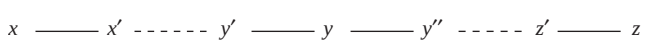

Fig. 1. Transitivity rule for non-degenerate antecedents.

In order to introduce a hypothesis $x = y$ from A, we need extract from $x = y$ the leftmost and rightmost global terms. For this purpose, we will use ${}^\bullet(x, y)$ as a shorthand for x if $x \preceq B$, else y and similarly $(x, y)^\bullet$ as a shorthand for y if $y \preceq B$, else x. Further, if x and y are both global, we introduce an A subchain into γ. Thus, letting $p|_B$ denote p if $p \preceq B$ else $\top$, we have

$$\textsc{HypEq-A}\frac{}{(A, B) \vdash x = y\ [{}^\bullet(x, y), (x, y)^\bullet, \top, (x = y)|_B]}\ (x = y) \in A.$$

The consequent of the above rule is easily shown to be valid, according to Definition 3, by splitting cases on whether $x \preceq B$ and $y \preceq B$. Introducing a hypothesis from B is handled as follows:

$$\textsc{HypEq-B}\frac{}{(A, B) \vdash x = y\ [x, y, \top, \top]}\ (x = y) \in B.$$

Soundness is straightforward. The interpolation rules for reflexivity and symmetry are as follows:

$$\textsc{Refl}\frac{}{(A, B) \vdash x = x\ [x, x, \top, \top]}\dagger, \qquad \textsc{Symm}\frac{(A, B) \vdash x = y\ [x', y', \rho, \gamma]}{(A, B) \vdash y = x\ [y', x', \rho, \gamma]}.$$

Here and in the sequel, † indicates that the terms equated in the consequent must occur in A or B. Note that for REFL, condition 3 holds because the side condition ensures $x \preceq B$ or $x \preceq A$. The other soundness conditions are straightforward.

Now we consider the transitivity rule. From antecedents $x = y$ and $y = z$, we derive $x = z$. Fig. 1 depicts the case when neither antecedent is degenerate. In the figure, solids lines represent equalities implied by A, and dotted lines represent equalities implied by B,γ. Notice that x' and z' are solutions for x and z. Moreover, the two center equalities can be combined to obtain an equality over global terms, $y' = y''$. If y is local, then we know $y', y'' \preceq A$. Adding this equality to γ, we have B, γ implies $x' = z'$, while γ is still over common symbols. Thus, γ is now an interpolant for $x = z$ under the solutions $x = x', z = z'$. On the other hand, if y is not local, then we know $y' \simeq y''$. Thus, γ serves as an interpolant unchanged. This gives us the following interpolation rule:

$$\textsc{Trans}\frac{\begin{array}{c}(A, B) \vdash x = y\ [x', y', \rho, \gamma],\\ (A, B) \vdash y = z\ [y'', z', \rho', \gamma']\end{array}}{(A, B) \vdash x = z\ [x', z', \rho \wedge \rho', \gamma \wedge \gamma' \wedge y' \doteq y'']}\ x' \not\simeq y, z' \not\simeq y,$$

where $x \doteq y$ denotes the formula $\top$ if $x \simeq y$ else the formula $x = y$.

Soundness. The first condition of Definition 3 holds trivially by validity of the antecedents. The side condition of the rule ensures that the antecedents are not degenerate. Now suppose

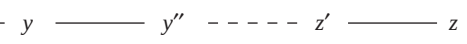

Fig. 2. Transitivity rule for one degenerate antecedent.

B, γ, γ' and $y' = y''$ hold. By validity of the antecedents, we know that $x' = y'$ and $y'' = z'$ hold. Thus, we have $B, \gamma \wedge \gamma' \wedge y' \doteq y'' \vDash x' = z'$. Moreover, since $x', z' \preceq B$ by validity of the antecedents, condition 2 is satisfied. Finally, condition 3 holds by validity of the antecedents. In particular, note that if $y \preceq B$, then $y \simeq y' \simeq y''$, so $y' \doteq y''$ is $\top$. Otherwise, we know that $y', y'' \preceq A$. Either way, $(y' \doteq y'') \preceq A$.

Now suppose that one of the antecedents is degenerate. Fig. 2 depicts the case where the antecedent $x = y$ is degenerate. Note here that y'' is a solution for x and z' is a solution for z. Moreover, B, γ give us $y'' = z'$. Thus γ gives us an interpolant for $x = z$ under the solutions $x = y'', z = z'$. On the other hand, if both antecedents are degenerate, then the consequent is also degenerate. Thus, letting $x(y/z)$ denote y if $x \simeq z$ else x, we have

$$\textsc{Trans}' \frac{(A, B) \vdash x = y\ [x', y', \rho, \gamma],\quad (A, B) \vdash y = z\ [y'', z', \rho', \gamma']}{(A, B) \vdash x = z\ [x'(y''/y), z'(y'/y), \rho \wedge \rho', \gamma \wedge \gamma']}\quad x' \simeq y \text{ or } z' \simeq y.$$

Soundness. Suppose that A, ρ and ρ' hold. Then we know $x = x'$ and $y = y''$ hold, thus $x = x'(y''/y)$ holds (and similarly $z = z'(y'/y)$ holds) thus condition 1 is satisfied. Now suppose that B, γ and γ' hold. If $x' \simeq y$ and $z' \not\simeq y$ then by validity of the antecedent we know that $z' = y''$ holds, hence $x'(y''/y) = z'(y'/y)$ holds (and a symmetric argument holds for the case $x' \not\simeq y$ and $z' \simeq y$). On the other hand, if $x' \simeq y$ and $z' \simeq y$, then either $y \not\preceq B$, in which case the consequent is degenerate, or $y \preceq B$, in which case $y \simeq y' \simeq y''$, thus trivially, $x'(y''/y) = z'(y'/y)$. In any case, condition 2 holds. Now suppose $x \preceq B$. Then $x' \simeq x$ and $x'(y''/y) = x$ holds. On the other hand, suppose $x \not\preceq B$. Then $x' \preceq A$. Thus if $x' \not\simeq y$ then $x'(y''/y) \preceq A$, however if $x' \simeq y$ then either $y \preceq B$ and $y'' \simeq y$ or $y \not\preceq B$ and $y'' \preceq A$. In either case, $x'(y''/y) \preceq A$. Arguing symmetrically for $z'(y'/y)$, we have condition 3.

Now we consider the CONG rule for uninterpreted functions symbols. Suppose that from $x = y$ we deduce $f(x) = f(y)$ by the CONG rule. To produce an interpolation, we must obtain solutions for $f(x)$ and $f(y)$ in terms of variables occurring in B (except in the degenerate case). We can easily obtain these solutions by simply applying f to the solutions for x and y. However, we must also take care in the case when the function symbol f does not occur in B, since in this case we cannot use f in the solutions. In the simple case, when either $f(x)$ or $f(y)$ occurs in B, we have the following rule (for unary functions):

$$\textsc{Cong}_1 \frac{(A, B) \vdash x = y\ [x', y', \rho, \gamma]}{(A, B) \vdash f(x) = f(y)\ [f(x'), f(y'), \rho, \gamma]}\ \dagger\quad f(x) \preceq B \text{ or } f(y) \preceq B.$$

Soundness. Since $A, \rho \vDash x = x' \wedge y = y' \wedge \gamma$, we know that $A, \rho \vDash f(x) = f(x') \wedge f(y) = f(y') \wedge \gamma$, satisfying condition 1. By the side condition, we have $x \preceq B$ or $y \preceq B$, so, since the antecedent satisfies condition 3, we know that either $x' \simeq x$ or $y' \simeq y$. Thus, either x and y are identical or the antecedent is non-degenerate. In either event, we have $x', y' \preceq B$ and

$B, \gamma \vDash x' = y'$. Since we know by the side condition that the function symbol f occurs in B, we have $f(x'), f(y') \preceq B$, and by congruence we have $B, \gamma \vDash f(x') = f(y')$, satisfying condition 2. For condition 3, if $f(x) \preceq B$, then $x \preceq B$, hence $x' \simeq x$ (since the antecedent satisfies condition 3), hence $f(x') \simeq f(x)$. If $f(x) \not\preceq B$ then $f(x)$ must occur in A, hence $x \preceq A$, hence $x' \preceq A$, hence $f(x') \preceq A$ (since we know f occurs in A). Thus (arguing symmetrically for $f(y)$, $f(y')$) condition 3 is satisfied.

Example 3. Suppose A is $x = y$ and B is $y = z$ and we wish to derive an interpolation for $f(x) = f(z)$. After introducing our two hypotheses, we use the TRANS′ rule to get $x = z$:

$$\text{TRANS}'\frac{\vdash x = y\ [y, y, \top, \top]\ \vdash y = z\ [y, z, \top, \top]}{\vdash x = z\ [y, z, \top, \top]}.$$

We then apply the CONG rule to obtain $f(x) = f(z)$:

$$\text{CONG}_1\frac{\vdash x = z\ [y, z, \top, \top]}{\vdash f(x) = f(z)\ [f(y), f(z), \top, \top]}.$$

The more complicated case is when neither $f(x)$ nor $f(y)$ occurs in B. Here, we cannot in general use f in the interpolant, since it may not be a common symbol. However, we can make use of the side condition that $f(x)$ and $f(y)$ must occur in A or B (i.e., the proof cannot introduce new terms). From this we know that $f(x)$ and $f(y)$ must occur in A. This allows us to produce a degenerate interpolation for the consequent. We let A prove $f(x) = f(y)$, but under a condition ρ proved by B. That is, A proves $f(x) = f(y)$ if B proves $\gamma \implies x' = y'$. Of course, we need this condition only if the antecedent is non-degenerate. Otherwise, A proves $f(x) = f(y)$ directly. Thus, the following rule applies, where $p|_B$ denotes p if $p \preceq B$ else $\top$:

$$\text{CONG}'_1\frac{(A, B) \vdash x = y\ [x', y', \rho, \gamma]}{\begin{array}{c}(A, B) \vdash f(x) = f(y) \\ [f(y), f(x), \rho \wedge (\gamma \implies (x' = y')|_B), \gamma].\end{array}}\ \dagger\ \ f(x) \not\preceq B \text{ and } f(y) \not\preceq B,$$

Soundness. Suppose the antecedent is degenerate, that is, $x' \simeq y$ and $y' \simeq x$. Then we have $A, \rho \vDash x = y \wedge y = x \wedge \gamma$. If it is not degenerate, then $x', y' \preceq B$, thus $(x' = y')|_B \simeq (x' = y')$. Since $A, \rho \vDash x = x' \wedge y = y' \wedge \gamma$, it follows that $A, \rho \wedge (\gamma \implies (x' = y')|_B) \vDash x = y \wedge y = x$. In either case, by congruence we have $A, \rho \wedge (\gamma \implies (x' = y')|_B) \vDash f(x) = f(y) \wedge f(y) = f(x) \wedge \gamma$ satisfying condition 1. If the antecedent is degenerate, and if x and y are not identical, we know that $x, y \not\preceq B$ (because the antecedent satisfies condition 3), thus $(x' = y')|_B \simeq \top$, thus $B \vDash \rho \wedge (\gamma \implies (x' = y')|_B)$. If the antecedent is not degenerate, then, by validity of the antecedent, $B, \gamma \vDash x' = y'$, thus we also have $B \vDash \rho \wedge (\gamma \implies (x' = y')|_B)$. Moreover, since the consequent is always degenerate, condition 2 is satisfied. Finally, since by the side condition, $f(x)$, $f(y)$ cannot occur in B, we know they must occur in A, satisfying condition 3.

The above two rules generalize in a natural way to n-ary function symbols. Using the notation $\bar{x}$ as an abbreviation for $x_1 \ldots x_n$, we have

$$\textsc{Cong}\ \frac{\begin{array}{c}(A, B) \vdash x_1 = y_1\ [x_1', y_1', \rho_1, \gamma_1],\\ \cdots\\ (A, B) \vdash x_n = y_n\ [x_n', y_n', \rho_n, \gamma_n]\end{array}}{\begin{array}{l}(A, B) \vdash f(\bar{x}) = f(\bar{y})\\ \quad [f(\bar{x}'), f(\bar{y}'), \wedge_{i=1}^n \rho_i, \wedge_{i=1}^n \gamma_i].\end{array}} \dagger \quad f(\bar{x}) \preceq B \text{ or } f(\bar{y}) \preceq B,$$

Soundness. Since, for all i, $A, \rho_i \vDash x_i = x_i' \wedge y_i = y_i' \wedge \gamma_i$, we know that $A, \wedge_{i=1}^n \rho_i \vDash f(\bar{x}) = f(\bar{x}') \wedge f(\bar{y}) = f(\bar{y}') \wedge (\wedge_{i=1}^n \gamma_i)$, satisfying condition 1. By the side condition, we have for all i, $x_i \preceq B$ or for all i, $y_i \preceq B$, so, since the antecedents satisfy condition 3, we know that for all i, either $x_i' \simeq x_i$ or $y_i' \simeq y_i$. Thus, either x_i and y_i are identical or the ith antecedent is non-degenerate. In either event, we have $x_i', y_i' \preceq B$ and $B, \gamma_i \vDash x_i' = y_i'$. Since we know by the side condition that the function symbol f occurs in B, we have $f(\bar{x}'), f(\bar{y}') \preceq B$, and by congruence we have $B, \wedge_{i=1}^n \gamma_i \vDash f(\bar{x}') = f(\bar{y}')$, satisfying condition 2. For condition 3, if $f(\bar{x}) \preceq B$, then for all i, $x_i \preceq B$, hence $x_i' \simeq x_i$ (since the antecedents satisfy condition 3), hence $f(\bar{x}') \simeq f(\bar{x})$. If $f(\bar{x}) \not\preceq B$ then $f(\bar{x})$ must occur in A, hence for all i, $x_i \preceq A$, hence $x_i' \preceq A$, hence $f(\bar{x}') \preceq A$ (since we know f occurs in A). Thus (arguing symmetrically for $f(\bar{y}), f(\bar{y}')$) condition 3 is satisfied.

For the case when neither $f(x)$ nor $f(y)$ occurs in B, we have

$$\textsc{Cong}'\ \frac{\begin{array}{c}(A, B) \vdash x_1 = y_1\ [x_1', y_1', \rho_1, \gamma_1],\\ \cdots\\ (A, B) \vdash x_n = y_n\ [x_n', y_n', \rho_n, \gamma_n]\end{array}}{\begin{array}{l}(A, B) \vdash f(\bar{x}) = f(\bar{y})\ [f(\bar{y}), f(\bar{x}),\\ \wedge_{i=1}^n (\rho_i \wedge (\gamma_i \Longrightarrow (x_i' = y_i')|_B)), \wedge_{i=1}^n \gamma_i].\end{array}} \dagger \quad f(\bar{x}) \not\preceq B,\ f(\bar{y}) \not\preceq B,$$

Soundness. Suppose the ith antecedent is degenerate, that is $x_i' \simeq y_i$ and $y_i' \simeq x_i$. Then we have $A, \rho_i \vDash x_i = y_i \wedge y_i = x_i \wedge \gamma_i$. If it is not degenerate, then $x_i', y_i' \preceq B$, thus $(x_i' = y_i')|_B \simeq (x_i' = y_i')$. Since $A, \rho_i \vDash x_i = x_i' \wedge y_i = y_i' \wedge \gamma_i$, it follows that $A, \rho_i \wedge (\gamma_i \Longrightarrow (x_i' = y_i')|_B) \vDash x_i = y_i \wedge y_i = x_i$. Thus, by congruence, we have $A, \wedge_{i=1}^n (\rho_i \wedge (\gamma_i \Longrightarrow (x_i' = y_i')|_B) \vDash f(\bar{x}) = f(\bar{y}) \wedge f(\bar{y}) = f(\bar{x}) \wedge (\wedge_{i=1}^n \gamma_i)$ satisfying condition 1. If the ith antecedent is degenerate, and if x_i and y_i are not identical, we know that $x_i, y_i \not\preceq B$ (because the antecedent satisfies condition 3), thus $(x_i' = y_i')|_B \simeq \top$, thus $B \vDash \rho_i \wedge (\gamma_i \Longrightarrow (x_i' = y_i')|_B)$. If the ith antecedent is not degenerate, then, by validity of the antecedent, $B, \gamma_i \vDash x_i' = y_i'$, thus we also have $B \vDash \rho_i \wedge (\gamma_i \Longrightarrow (x_i' = y_i')|_B)$. Thus, $B \vDash \wedge_{i=1}^n (\rho_i \wedge (\gamma_i \Longrightarrow (x_i' = y_i')|_B))$. Moreover, since the consequent is always degenerate condition 2 is satisfied. Finally, since by the side condition, $f(\bar{x}), f(\bar{y})$ cannot occur in B, we know they must occur in A, satisfying condition 3.

Now we deal with the EQNEQ rule, which derives false from an equality and its negation. First, we consider the case where the disequality is contained in A:

$$\textsc{EqNeq-A}\ \frac{(A, B) \vdash x = y\ [x', y', \rho, \gamma]}{(A, B) \vdash \bot\ [0, \rho, \gamma \wedge (x' \neq y')]}\ (x \neq y) \in A,\ y' \not\preceq x \text{ or } x' \not\preceq y.$$

Notice that we derive an inequality interpolation here so that we can then apply the CONTRA rule. The idea is to translate the disequality over local terms to an equivalent disequality over global terms.

Soundness. Since $A, \rho \vDash x = x' \wedge y = y'$, and $A \vDash x \neq y$, we know $A, \rho \vDash x' \neq y'$, which gives us condition 1. Since by the side condition, the antecedent is not degenerate, we have $B, \gamma \vDash x' = y'$, thus $B, \gamma \wedge (x' \neq y') \vDash \bot$, which gives us condition 2. Condition 3 is trivial.

We handle the degenerate case separately:

$$\text{EQNEQ-A}'\frac{(A, B) \vdash x = y\ [y, x, \rho, \gamma]}{(A, B) \vdash \bot\ [0, \rho, \bot]}\ (x \neq y) \in A.$$

Soundness. Since $A, \rho \vDash x = y$, and $A \vDash x \neq y$, we know $A, \rho \vDash \bot$, which gives us condition 1. Further, $B, \bot \vDash \bot$, giving us condition 2. Condition 3 is trivial.

The case where the disequality comes from B is handled as follows:

$$\text{EQNEQ-B}\frac{(A, B) \vdash x = y\ [x', y', \rho, \gamma]}{(A, B) \vdash \bot\ [0, \rho, \gamma]}(x \neq y) \in B.$$

Soundness. Condition 1 is trivial. Since by the side condition, $x, y \preceq B$, by condition 3 of the antecedent, we know $x' \simeq x$ and $y' \simeq y$, thus $B, \gamma \vDash x = y$, thus $B, \gamma \vDash \bot$, satisfying condition 2. Condition 3 is trivial.

2.3. Combining LI and EUF

In the combined logic, we will say that a *term* is an individual variable or a function application $f(x_1, \ldots, x_n)$, where f is a n-ary function symbol and $x_1 \ldots x_n$ are terms. An *arithmetic term* is a linear combination $c_0 + c_1 v_1 + \cdots c_n v_n$, where $v_1 \ldots v_n$ are distinct terms and $c_0 \ldots c_n$ are integer constants, and where $c_1 \ldots c_n$ are non-zero. An *atomic predicate* is either a propositional variable, an inequality of the form $0 \leqslant x$, where x is an arithmetic term, or an equality of the form $x = y$, where x and y are terms.

Our proof system consists of all the previous proof rules, with the addition of the following two rules that connect equality and inequality reasoning:

$$\text{LEQEQ}\frac{\Gamma \vdash x = y}{\Gamma \vdash 0 \leqslant x - y},$$

$$\text{EQLEQ}\frac{\Gamma \vdash 0 \leqslant x - y \quad \Gamma \vdash 0 \leqslant y - x}{\Gamma \vdash x = y}\ \dagger.$$

The LEQEQ rule, inferring an inequality from an equality, can be handled by the following interpolation rules:

$$\text{LEQEQ}\frac{(A, B) \vdash x = y\ [x', y', \rho, \gamma]}{(A, B) \vdash 0 \leqslant x - y\ [x - x' - y + y', \rho, \gamma]}\ y' \not\simeq x \text{ or } x' \not\simeq y.$$

Soundness. Since $A, \rho \vDash x' = x \wedge y' = y$, we have $A, \rho \vDash 0 \leqslant x - x' - y + y'$, satisfying condition 1. Since $B, \gamma \vDash x' = y'$, we have $B, \gamma \vDash 0 \leqslant (x - x' - y + y') - (x - y)$, satisfying

condition 2. Finally, since $x', y' \preceq B$, it follows that the coefficients of any $v \not\preceq B$ must be the same in $x - x' - y + y'$ and $x - y$, satisfying condition 3.

We deal separately with the special case where the antecedent is degenerate:

$$\textsc{LeqEq}'\frac{(A, B) \vdash x = y\ [y, x, \rho, \gamma]}{(A, B) \vdash 0 \leqslant x - y\ [x - y, \rho, \gamma]}.$$

Soundness. Since $A, \rho \vDash x = y$, we have $A, \rho \vDash 0 \leqslant x - y$, satisfying condition 1. Conditions 2 and 3 are trivial.

We now consider the EQLEQ rule, which derives an equality from a pair of inequalities. We distinguish three cases, depending on whether x and y are local or global. The first case is when both x and y are global, and is straightforward:

$$\textsc{EqLeq-BB}\frac{\begin{array}{c}(A, B) \vdash 0 \leqslant x - y\ [x', \rho, \gamma],\\ (A, B) \vdash 0 \leqslant y - x\ [y', \rho', \gamma']\end{array}}{\begin{array}{l}(A, B) \vdash x = y\ [x, y, \rho \wedge \rho',\\ \qquad\qquad \gamma \wedge \gamma' \wedge 0 \leqslant x' \wedge 0 \leqslant y']\end{array}} \dagger \quad x \preceq B, y \preceq B.$$

Soundness. Condition 1 is trivial. By validity of the antecedents, $B, \gamma \vDash 0 \leqslant (x - y) - x'$, thus $B, \gamma \wedge 0 \leqslant x' \vDash 0 \leqslant x - y$ (and similarly $B, \gamma' \wedge 0 \leqslant y' \vDash 0 \leqslant y - x$). Thus, $B, \gamma \wedge \gamma' \wedge 0 \leqslant x' \wedge 0 \leqslant y' \vDash x = y$, satisfying condition 2. Finally, by the side condition and by condition 3 of the antecedents, we know that $x', y' \preceq B$ and $x', y' \preceq A$. Thus, condition 3 is satisfied.

The case when x is local and y is global is more problematic. Suppose, for example, that A is $(0 \leqslant x - a)(0 \leqslant b - x)$ and B is $(0 \leqslant y - b)(0 \leqslant a - y)$. From this we can infer $0 \leqslant x - y$ and $0 \leqslant y - x$, using the COMB rule. Thus, using the EQLEQ rule, we infer $x = y$. To make an interpolation for this, we must have a solution for x in terms of global variables, implied by A. Unfortunately, there are no equalities that can be inferred from A alone. However, we can derive a *conditional* solution, using the ρ component of the interpolation. In our example, we will have

$$(A, B) \vdash x = y\ [b, y, 0 \leqslant a - b, 0 \leqslant b - a].$$

That is, A proves $x = b$, under the condition ρ that $0 \leqslant a - b$. This interpolation is valid, since from B we can prove $0 \leqslant a - b$. Using A and this fact, we can infer $x = b$. From A we can also infer $0 \leqslant b - a$, which, with B, gives us $b = y$, hence $x = y$. This approach can be generalized to the following rule:

$$\textsc{EqLeq-AB}\frac{\begin{array}{c}(A, B) \vdash 0 \leqslant x - y\ [x', \rho, \gamma],\\ (A, B) \vdash 0 \leqslant y - x\ [y', \rho', \gamma']\end{array}}{\begin{array}{l}(A, B) \vdash x = y\ [x + y', y, \rho \wedge \rho' \wedge 0 \leqslant\ -x' - y',\\ \qquad\qquad \gamma \wedge \gamma' \wedge 0 \leqslant x' + y']\end{array}} \dagger \quad x \not\preceq B, y \preceq B.$$

Soundness. By validity of the antecedents, we have $A, \rho \wedge \rho' \vDash 0 \leqslant y' \wedge\ 0 \leqslant x'$. Thus, summing inequalities we have $A, \rho \wedge \rho' \wedge 0 \leqslant\ -x' - y' \vDash 0 \leqslant y' \wedge 0 \leqslant -y' \wedge 0 \leqslant x' + y'$, thus $A, \rho \wedge \rho' \wedge 0 \leqslant\ -x' - y' \vDash x = x + y' \wedge 0 \leqslant x' + y'$, satisfying condition 1. Since $(y - x - y') \preceq B$, by condition 3 of the second antecedent, and since $y \preceq B$, we know that the coefficients of local variables in $-x$ and y' are the same, so $(x + y') \preceq B$. Moreover, by

validity of the antecedents, we have $B, \gamma \vDash 0 \leqslant x - y - x'$ and $B, \gamma' \vDash 0 \leqslant y - x - y'$. From the former, summing inequalities, we have $B, \gamma, 0 \leqslant x' + y' \vDash 0 \leqslant (x + y') - y$. Combining with the latter, we have $B, \gamma \wedge \gamma' \wedge 0 \leqslant x' + y' \vDash x + y' = y$, satisfying condition 2. We know that the coefficients of local variables in $-x$ and y' are the same, and similarly for x and x'. If follows that $(x' + y') \preceq B$. Moreover, since $x \not\preceq B$, we know that x occurs in A, and we know by condition 3 of the second antecedent that $y' \preceq A$. Thus $x + y' \preceq A$, satisfying condition 3.

We can also write a symmetric rule EQLEQ-BA. The final case for the EQLEQ rule is when $x \not\preceq B$ and $y \not\preceq B$:

$$\text{EQLEQ-AA}\frac{\begin{array}{c}(A, B) \vdash 0 \leqslant x - y\ [x', \rho, \gamma],\\ (A, B) \vdash 0 \leqslant y - x\ [y', \rho', \gamma']\end{array}}{\begin{array}{r}(A, B) \vdash x = y\ [y, x, \rho \wedge \rho' \wedge 0 \leqslant y - x - y'\\ \wedge\ 0 \leqslant x - y - x', \gamma \wedge \gamma']\end{array}}\ \dagger\quad x \not\preceq B, y \not\preceq B.$$

Soundness. By validity of the antecedents, we have $A, \rho \vDash 0 \leqslant x'$ and $A, \rho' \vDash 0 \leqslant y'$. Thus, summing equalities, we have $A, \rho, 0 \leqslant x - y - x' \vDash 0 \leqslant x - y$ and A, ρ', $0 \leqslant y - x - y' \vDash 0 \leqslant y - x$, thus $A, \rho \wedge \rho' \wedge 0 \leqslant y - x - y' \wedge 0 \leqslant x - y - x' \vDash x = y$, satisfying condition 1. Also by validity of the antecedents, we have $B, \gamma \vDash 0 \leqslant x - y - x'$ and $B, \gamma' \vDash 0 \leqslant y - x - y'$. Moreover, the consequent is degenerate, so condition 2 is satisfied. By the side condition, we know x, y occur in A. Moreover, by condition 3 of the antecedents, we have $x' \preceq A$, $y' \preceq A$, $y - x - y' \preceq B$ and $x - y - x' \preceq B$. Thus, condition 3 is satisfied.

2.4. Soundness and completeness

We are now ready to state soundness and completeness results for our interpolation system as a whole.

Definition 4. A formula ϕ is said to be an *interpolant* for a pair of formula sets (A, B) when
(1) $A \vDash \phi$, and
(2) $B, \phi \vDash \bot$, and
(3) $\phi \preceq A$ and $\phi \preceq B$.

Theorem 1 (*Soundness*). *If a clause interpolation of the form* $(A, B) \vdash \langle\rangle\ [\phi]$ *is derivable, then* ϕ *is an interpolant for* (A, B).

Proof sketch. Validity of the interpolation is by the soundness of the individual interpolation rules and induction over the derivation length. By Definition 2 we know that A implies ϕ, that B and ϕ are inconsistent and that $\phi \preceq B$ and $\phi \preceq A$.

Theorem 2 (*Completeness*). *For any derivable sequent* $A, B \vdash \psi$, *there is a derivable interpolation of the form* $(A, B) \vdash \psi\ [X]$.

Proof sketch. We split cases on the rule used to derive the sequent, and show in each case that there is always a rule to derive an interpolation for the consequent from interpolations for the antecedents.

In effect, the proof of the completeness theorem gives us an algorithm for constructing an interpolant from a refutation proof. This algorithm is linear in the proof size, and the result is a formula (not in CNF) whose circuit size is also linear in the proof size. [2]

2.5. Completeness issues for rational and integer arithmetic

Our system of interpolation rules is complete relative to the original proof system in the sense that for every derivable sequent there is a corresponding derivable interpolation. However, the original proof system itself is not complete as given. For rational models, we can obtain a complete system by simply treating the literal $\neg(0 \leqslant x)$ as a synonym for $(0 \leqslant -x) \wedge (x \neq 0)$. That is, if we replace every occurrence of $\neg(0 \leqslant x)$ in the antecedent of the CONTRA rule with the equivalent pair of literals $(0 \leqslant -x)$ and $(x \neq 0)$, both the original CONTRA rule and the corresponding interpolation rule remain sound. The resulting system is complete for refutations over rational models.

The case for integers is somewhat more problematic. We can obtain an incomplete system by treating $\neg(0 \leqslant x)$ as a synonym for $0 \leqslant -1-x$, as is done in [10]. As noted in [10], the solution space for a set of integer linear inequalities is not convex. Thus, for completeness it may be necessary to split cases until the solution space becomes convex. Unfortunately, the restriction we put on the CONTRA rule prevents us from splitting cases on atomic predicates not already present in A or B. Thus, we cannot make arbitrary cuts. However, *can* effectively split cases on any atomic predicate p so long as $p \preceq A$ or $p \preceq B$. Suppose, for example, that $p \preceq A$. In this case, we can add the tautology clause $(p \vee \neg p)$ to A while preserving both its extension and its support, and thus the validity of any interpolant we may obtain. In this way, we can introduce the predicate p into the proof, and thus we can split cases on it.

In particular, in the case of integer arithmetic, for any predicate $0 \leqslant x$ occurring in A or B, we can split cases on $0 \leqslant -x$. This allows us to split cases until the solution space becomes convex. With such case splits, our system becomes complete for integer linear arithmetic where all coefficients are 1 or -1, though it still cannot disprove equalities such as $2x-2y=1$. For non-unit coefficients, quantifier-free interpolants do not in general exist. Consider, for example, the case where A is $x=2y$ and B is $x=2z+1$. The only interpolant for this pair is "x is even", which is not expressible in the logic without a quantifier. Thus we cannot expect to obtain a complete system for general integer linear arithmetic.

2.6. Interpolants for quantified formulas

Although the primary purpose of this work is to generate interpolants without quantifiers, we should note that the method can also be applied to quantified formulas, generating quantified interpolants. Suppose, for example, that formulas A and B contain quantifiers, and that we have Skolemized these formulas to reduce them to universal prenex form. We then instantiate the universal quantifiers with free individual variables to create quantifier-free formulas A' and B'. In effect, this allows us to instantiate the quantifiers with any term t occurring in A or B, by creating a new variable v_t and adding the equality $v_t = t$ to A or B

[2] A sample implementation of this procedure in the OCAML language is available at `http://www-cad.eecs.berkeley.edu/~kenmcmil`.

as appropriate. Now we can compute an interpolant ϕ' for the pair of instantiated formulas (A', B'). The interpolant ϕ' may contain some v_t variables not occurring in A. However, as in [2], we can eliminate these variables by quantifying them universally. Similarly, v_t variables not occurring in B can be eliminated by quantifying them existentially. The resulting quantified formula is still implied by A and inconsistent with B, thus it is an interpolant for (A, B).

3. An interpolating prover

Thus far we have described a proof system for a logic with linear inequalities and uninterpreted functions, and set of rules for deriving interpolants from proofs in this system. There are two further problems that we must address: constructing an efficient proof-generating decision procedure for our system, and translating interpolation problems for general formulas into interpolation problems in clause form.

3.1. Generating proofs

The prover combines a DPLL style SAT solver, similar to Chaff [9], for propositional reasoning, with a proof-generating Nelson–Oppen style ground decision procedure for theory reasoning. They are combined using the "lazy" approach of [3]. That is, the SAT solver treats all atomic predicates in a given formula f as free Boolean variables. When it finds an assignment to the atomic predicates that satisfies f propositionally, it passes this assignment to the theory decision procedure in the form of a set of literals $l_1 \ldots l_n$. The ground decision procedure then attempts to derive a refutation of this set of literals. If it succeeds, the literals used as hypotheses in the refutation are gathered (call them $m_1, \ldots, m_k$). The CONTRA rule is then used to derive the new clause $\langle \neg m_1, \ldots, \neg m_k \rangle$. This clause is added to the SAT solver's clause set. We will refer to it as a *blocking clause*. Since it is in conflict in the current assignment, the SAT solver now backtracks, continuing where it left off. On the other hand, if the ground decision procedure cannot refute the satisfying assignment, the formula f is satisfiable and the process terminates.

The SAT solver is modified in a straightforward way to generate refutation proofs by resolution (see [8] for details). When a conflict occurs in the search (i.e., when all the literals in some clause are assigned to false), the solver resolves the conflicting clause with other clauses to infer a so-called "conflict clause" (a technique introduced in the GRASP solver [14] and common to most modern DPLL solvers). This inferred clause is added to the clause set, and in effect prevents the same conflict from occurring in the future. The clause set is determined to be unsatisfiable when the empty clause (false) is inferred as a conflict clause. To derive a proof of the empty clause, we have only to record the sequence of resolution steps used to derive each conflict clause.

The SAT solver's clause set therefore consists of three classes of clauses: the original clauses of f, blocking clauses (which are tautologies proved by the ground decision procedure) and conflict clauses (proved by resolution). When the empty clause is derived, we construct a refutation of f using the stored proofs of the blocking clauses and the conflict clauses.

3.2. Interpolants for structured formulas

Of course, the interpolation problem (A, B) is not in general given in the clause form required by our proof system. In general, A and B have arbitrary nesting of Boolean operators. We now show how to reduce the problem of finding an interpolant for arbitrary formulas (A, B) into the problem of finding an interpolant for (A_c, B_c), where A_c and B_c are in clause form.

It is well known that *satisfiability* of an arbitrary formula f can be reduced in linear time to satisfiability of a clause form formula [11]. This transformation uses a set V of fresh Boolean variables, containing a variable v_g for each non-atomic propositional subformula g of f. A small set of clauses is introduced for each occurrence of a Boolean operator in f. For example, if the formula contains $g \wedge h$, we add the clauses $\langle v_g, \neg v_{g\wedge h}\rangle$, $\langle v_h, \neg v_{g\wedge h}\rangle$ and $\langle \neg v_g, \neg v_h, v_{g\wedge h}\rangle$. These clauses constrain $v_{g\wedge h}$ to be the conjunction of v_g and v_h. We will refer to the collection of these clauses for all non-atomic subformulas of f as $\text{CNF}_V(f)$. We then add the clause $\langle v_f\rangle$ to require that the entire formula is true. The resulting set of clauses is satisfiable exactly when f is satisfiable.

In fact, we can show something stronger, which is that any formula implied by $\text{CNF}_V(f) \wedge v_f$ that does not refer to the fresh variables in V is also implied by f. This gives us the following result:

Theorem 3. *Let $A_c = \text{CNF}_U(A), \langle u_A\rangle$ and $B_c = \text{CNF}_V(B), \langle v_B\rangle$, where U, V are disjoint sets of fresh variables, and A, B are arbitrary formulas. An interpolant for (A_c, B_c) is also an interpolant for (A, B).*

This theorem allows us to compute interpolants for structured formulas by using the standard translation to clause form.

4. Applications

The interpolating prover described above has a number of possible applications in formal verification. These include refinement in predicate abstraction, and model checking infinite-state systems, with and without predicate abstraction.

4.1. Using interpolation for predicate refinement

Predicate abstraction [13] is a technique commonly used in software model checking in which the state of an infinite-state system is represented abstractly by the truth values of a chosen set of predicates. In effect, the method computes the strongest inductive invariant of the program expressible as a Boolean combination of the predicates. Typically, if this invariant is insufficient to prove the property in question, the abstraction is refined by adding predicates. For this purpose, the BLAST software model checker uses the interpolating prover in a technique due to Ranjit Jhala [4].

The basic idea of the technique is as follows. A counterexample is a sequence of program locations (a path) that leads from the program entry point to an error location. When the

model checker finds a counterexample in the abstraction, it builds a formula that is satisfiable exactly when the path is a counterexample in the concrete model. This formula consists of a set of constraints: equations that define the values of program variables in each location in the path, and predicates that must be true for execution to continue along the path from each location (these correspond to program branch conditions).

Now let us divide the path into two parts, at state k. Let A_k be the set of constraints on transitions preceding state k and let B_k be the set of constraints on transitions subsequent to state k. Note that the common variables of A and B represent the values of the program variables at state k. An interpolant for (A_k, B_k) is a fact about state k that must hold if we take the given path to state k, but is inconsistent with the remainder of the path. In fact, if we derive such interpolants for every state of the path from the same refutation of the constraint set, we can show that the interpolant for state k is sufficient to prove the interpolant for state $k + 1$. As a result, if we add the atomic predicates occurring in the interpolants to the set of predicates defining the abstraction, we are guaranteed to rule out the given path as a counterexample in the abstract model. Note that it is important here that interpolants be quantifier-free, since the predicate abstraction method can synthesize any Boolean combination of atomic predicates, but cannot synthesize quantifiers.

This interpolation approach to predicate refinement has the advantage that it tells us which predicates are relevant to each program location in the path. By using at each program location only predicates that are relevant to that location, a substantial reduction in the number of abstract states can be achieved, resulting in greatly increased performance of the model checker [4]. The fact that the interpolating prover can handle both linear inequalities and uninterpreted functions is useful, since linear arithmetic can represent operations on index variables, while uninterpreted functions can be used to represent array lookups or pointer dereferences, or to abstract unsupported operations (such as multiplication). [3]

4.2. Model checking with interpolants

Image computation is the fundamental operation of symbolic model checking [1]. This requires quantifier elimination, which is generally the most computationally expensive aspect of the technique. In [7] a method of approximate image computation is described that is based on interpolation, and does not require quantifier elimination. While the method is over-approximate, it is shown that it can always be made sufficiently precise to prevent false negatives for systems of finite diameter. While [7] treats only the propositional case, the same theory applies to interpolation for first order logic. Thus, in principle the interpolating prover can be used for interpolation-based model checking of infinite-state systems whose transition relation can be expressed in LIUF.

One potential application would be model checking with predicate abstraction. This is a case where the transition relation is expressible in first order logic and the state space is finite,

[3] Unfortunately, array updates cannot be handled directly, since the theory of *store* and *select* does not allow quantifier-free interpolants. Suppose, for example that A is $M' = store(M, a, x)$ and B is $(b \neq c) \wedge (select(M', b) \neq select(M, b)) \wedge (select(M', c) \neq select(M, c))$. The common variables here are M and M', but no facts expressible using only these variables are implied by A (except true), thus there is no interpolant for this pair. This problem can be avoided for deterministic programs by rewriting terms from B_k into terms over program variables at step k. In general, however, we need quantifiers to deal with array updates.

guaranteeing convergence. That is, the state is defined in terms of a set of Boolean variables $v_1 \ldots v_k$ corresponding to the truth values of first-order predicates $p_1 \ldots p_k$. The abstraction relation α is characterized symbolically by the formula $\bigwedge_i v_i' \leftrightarrow p_i$. If the concrete transition relation is characterized by R, the abstract transition relation can be written as the relational composition $\alpha^{-1} \circ R \circ \alpha$. Note that the relational composition can be accomplished by a simple renaming, replacing the "internal" variables with fresh variables that are implicitly existentially quantified. That is, $R \circ S$ can be written as $R\langle U/V'\rangle \wedge S\langle U/V\rangle$ where V and V' are the current and next-state variables respectively, and U is a set of fresh variables. Thus, if the concrete transition relation can be written as a formula in LIUF, then so can the abstract transition relation.

This formula can in turn be rewritten as a satisfiability-equivalent Boolean formula, as is done in [6]. This allows the application of finite-state methods for image computation, but has the disadvantage that it introduces a large number of auxiliary boolean variables, making BDD-based image computations impractical. Although SAT-based quantifier elimination techniques are more effective in this case, this approach limits the technique to a small number of predicates. On the other hand, the interpolation-based approach does not require quantifier elimination or translation of the transition relation to a Boolean formula, and thus avoids these problems.

Another possible approach would be to model check the concrete, infinite-state system directly using the interpolation method of [7]. For this purpose, it is also important that the interpolants be quantifier-free. This is because the procedure is iterative—each reached-state set approximation is an interpolant which must be fed back into a ground decision procedure to compute the next approximation. For infinite state systems in general this process is not guaranteed to converge. However, in the special case when the model has a finite bisimulation quotient, convergence is guaranteed. This is the case, for example, for timed automata. Since the transition relation of a timed automaton can be expressed in LI, it follows that reachability for timed automata can be verified using the interpolation method. As an example, a model of Fischer's timed mutual exclusion protocol has been verified in this way. Similarly, a simple model of Lamport's "bakery" mutual exclusion, with unbounded ticket numbers, has been modeled and verified (for safety). Using the method described above for quantified interpolants, and some simple quantifier instantiation heuristics, it was also possible to prove the simple bakery model for an arbitrary number of processes. In principle, this method could be applied to software model checking.

5. Conclusions and future work

The primary contribution of this work is a method of computing quantifier-free Craig interpolants from refutations in a theory that includes linear inequalities and uninterpreted functions. This extends earlier results that apply only to linear inequalities or only to propositional logic. This procedure has been integrated with a proof generating decision procedure, combining a SAT solver and a Nelson–Oppen style prover to create an interpolating prover.

While the motivation for this work is mainly to experiment with interpolation-based model checking of infinite-state systems, it has also been applied in a manner quite unexpected by its author, to the problem of predicate refinement in the BLAST tool.

For future work, it is hoped that the interpolating prover will be useful for direct interpolation-based software model checking, perhaps in a hybrid approach between the fully symbolic method of [7] and the explicit search method of BLAST. It is also interesting to consider what other theories might be usefully incorporated into the prover.

References

[1] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, L.J. Hwang, Symbolic model checking: 10^{20} states and beyond, in: Proc. Fifth Annu. IEEE Symp. on Logic in Computer Science, IEEE Computer Society Press, Washington, DC, 1990, pp. 1–33.
[2] W. Craig, Three uses of the herbrand-gentzen theorem in relating model theory and proof theory, J. Symbolic Logic 22 (3) (1957) 269–285.
[3] L. de Moura, H. Rueß, M. Sorea, Lazy theorem proving for bounded model checking over infinite domains, in: 18th Conf. on Automated Deduction, CADE 2002, Lecture Notes in Computer Science, Springer, Copenhagen, Denmark, July 27–30, 2002.
[4] T.A. Henzinger, R. Jhala, R. Majumdar, K.L. McMillan, Abstractions from proofs, in: ACM Symp. on Principles of Programming Language, POPL 2004, 2004, pp. 232–244.
[5] J. Krajíček, Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic, J. Symbolic Logic 62 (2) (1997) 457–486.
[6] S.K. Lahiri, R.E. Bryant, B. Cook, A symbolic approach to predicate abstraction, in: Computer-Aided Verification, CAV 2003, 2003, pp. 141–153.
[7] K.L. McMillan, Interpolation and sat-based model checking, in: Computer-Aided Verification, CAV 2003, 2003, pp. 1–13.
[8] K.L. McMillan, N. Amla, Automatic abstraction without counterexamples, in: Internat. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2003, 2003, pp. 2–17.
[9] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, S. Malik, Chaff: engineering an efficient SAT solver, in: Design Automation Conference, 2001, pp. 530–535.
[10] G. Nelson, D.C. Oppen, Simplification by cooperating decision procedures, ACM Trans. Progr. Language Systems 1 (2) (1979) 245–257.
[11] D. Plaisted, S. Greenbaum, A structure preserving clause form translation, J. Symbolic Comput. 2 (1986) 293–304.
[12] P. Pudlák, Lower bounds for resolution and cutting plane proofs and monotone computations, J. Symbolic Logic 62 (2) (1997) 981–998.
[13] H. Saïdi, S. Graf, Construction of abstract state graphs with PVS, in: O. Grumberg (Ed.), Computer-Aided Verification, CAV '97, Vol. 1254, Springer, Haifa, Israel, 1997, pp. 72–83.
[14] J.P.M. Silva, K.A. Sakallah, GRASP—a new search algorithm for satisfiability, in: Proc. Internat. Conf. on Computer-Aided Design, November 1996.