Interpolation, Invariant and Ranking Function Generation
Model Checking (IN2050)

Andrey Rybalchenko

Technische Universität München

We use the program shown in Figure 1 as a source of termination, interpolation and safety proving obligations. When translating the program instructions into the corresponding transition relations we approximate integer program variables by rationals, in order to reduce the complexity the resulting constraint generation and solving tasks. Hence, the relation ρ_2 has a guard $x + 1 \leq y$. Furthermore, the failure of the assert statement is represented by reachability of the control location ℓ_5.

1 Linear ranking functions

Program termination is an important property that ensures its responsiveness. Proving program terminations requires construction of ranking functions that over-approximate the number of execution steps that the program can make from a given state until termination. Linear ranking functions express such approximations by linear assertions over the program variables.

1.1 Example

Input We illustrate the construction of ranking functions on the while loop from the program in Figure 1, as shown below. See [5] for its detailed description and pointers to the related work.

```
while (x < y) {
  x++;
}
```

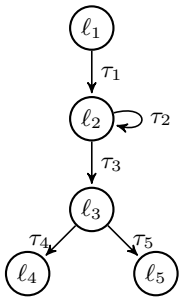
We deliberately choose a loop that neither contains further nesting loops nor branching control flow inside the loop body in order to highlight the main ideas of the constraint-based ranking function generation.

Our algorithm will search for a linear expression over the program variables that proves termination. Such an expression is determined by the coefficients of the occurring variables. Let f_x and f_y be the coefficients for the variables x and y, respectively. Since the program variable z does not play a role in the loop, to simplify the presentation we do not take it into consideration.

A linear expression is a ranking function if its value is bounded from below for all states on which the loop can make a step, and is decreasing by some a priory fixed positive amount. Let δ_0 be the lower bound for the value of the

```
main(int x, int y, int z) {
  assume(y >= z);
  while (x < y) {
    x++;
  }
  assert(x >= z);
}
```

(a)



(b)

$$\rho_1 = (y \geq z \wedge x' = x \wedge y' = y \wedge z' = z)$$
$$\rho_2 = (x + 1 \leq y \wedge x' = x + 1 \wedge y' = y \wedge z' = z)$$
$$\rho_3 = (x \geq y \wedge x' = x \wedge y' = y \wedge z' = z)$$
$$\rho_4 = (x \geq z \wedge x' = x \wedge y' = y \wedge z' = z)$$
$$\rho_5 = (x + 1 \leq z \wedge x' = x \wedge y' = y \wedge z' = z)$$

(c)

Fig. 1. An example program (a), its control-flow graph (b), and the corresponding transition relations (c).

ranking function, and δ by the lower bound on the amount of decrease. Then, we obtain the following defining constraint on the ranking function coefficients and the bound values.

$$\begin{aligned} &\exists f_x\ \exists f_y\ \exists \delta_0\ \exists \delta \\ &\quad \forall x\ \forall y\ \forall x'\ \forall y' : \\ &\qquad (\delta \geq 1 \wedge \\ &\qquad \rho_2 \rightarrow (f_x x + f_y y \geq \delta_0 \wedge \\ &\qquad\qquad f_x x' + f_y y' \leq f_x x + f_y y - \delta)) \end{aligned} \tag{1}$$

Any satisfying assignment to f_x, f_y, δ_0 and δ determines a linear ranking function for the loop.

The constraint (1) contains universal quantification over the program variables and their primed version, which makes it difficult to solve directly using existing constraint solvers. At the next step, we will address this obstacle by eliminating the universal quantification.

Constraints First, we represent the transition relation of the loop in matrix form, which will help us during the constraint generation. After replacing equalities by conjunctions of corresponding inequalities, we obtain the matrix form

below.

$$\begin{aligned}\rho_2 &= (x+1 \le y \wedge x' = x+1 \wedge y' = y)\\ &= (x-y \le -1 \wedge -x+x' \le 1 \wedge x-x' \le -1 \wedge -y+y' \le 0 \wedge y-y' \le 0)\\ &= \begin{pmatrix} 1 & -1 & 0 & 0\\ -1 & 0 & 1 & 0\\ 1 & 0 & -1 & 0\\ 0 & -1 & 0 & 1\\ 0 & 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} x\\ y\\ x'\\ y' \end{pmatrix} \le \begin{pmatrix} -1\\ 1\\ -1\\ 0\\ 0 \end{pmatrix}\end{aligned}$$

The bound and decrease conditions from (1) produce the following matrix forms.

$$f_x x + f_y y \ge \delta_0 = \begin{pmatrix} -f_x & -f_y & 0 & 0 \end{pmatrix} \begin{pmatrix} x\\ y\\ x'\\ y' \end{pmatrix} \le -\delta_0$$

$$f_x x' + f_y y' \le f_x x + f_y y - \delta = \begin{pmatrix} -f_x & -f_y & f_x & f_y \end{pmatrix} \begin{pmatrix} x\\ y\\ x'\\ y' \end{pmatrix} \le -\delta$$

Now we are ready to eliminate the universal quantification. For this purpose we apply Farkas' lemma, which formally states

$$((\exists x : Ax \le b) \wedge (\forall x : Ax \le b \to cx \le \gamma)) \leftrightarrow (\exists \lambda : \lambda \ge 0 \wedge \lambda A = c \wedge \lambda b \le \gamma)\ .$$

This statement asserts that every linear consequence of a satisfiable set of linear inequalities can be obtained as a non-negative linear combination of these inequalities. As an immediate consequence we obtain that for a non-satisfiable set of linear inequalities we can derive an unsatisfiable inequality, i.e.,

$$(\forall x : \neg(Ax \le b)) \leftrightarrow (\exists \lambda : \lambda \ge 0 \wedge \lambda A = 0 \wedge \lambda b \le -1)\ .$$

By applying Farkas' lemma on (1) we obtain the following constraint.

$$
\begin{aligned}
&\exists f_x\ \exists f_y\ \exists \delta_0\ \exists \delta \\
&\quad \exists \lambda\ \exists \mu : \\
&\qquad (\delta \geq 1 \wedge \\
&\qquad \lambda \geq 0 \wedge \\
&\qquad \mu \geq 0 \wedge \\
&\qquad \lambda \begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & -1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{pmatrix} = \begin{pmatrix} -f_x & -f_y & 0 & 0 \end{pmatrix} \wedge \lambda \begin{pmatrix} -1 \\ 1 \\ -1 \\ 0 \\ 0 \end{pmatrix} \leq -\delta_0 \wedge \\
&\qquad \mu \begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & -1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{pmatrix} = \begin{pmatrix} -f_x & -f_y & f_x & f_y \end{pmatrix} \wedge \mu \begin{pmatrix} -1 \\ 1 \\ -1 \\ 0 \\ 0 \end{pmatrix} \leq -\delta
\end{aligned}
\tag{2}
$$

This constraint contains only existentially quantified rational variables and consists of linear (in)equalities. Thus, it can be efficiently solved by the existing tools for Linear Programming over rationals.

Solution We apply a linear constraint solver on (2) and obtain the following solution.

$$
\begin{aligned}
\lambda &= (1\ 0\ 0\ 0\ 0) \\
\mu &= (0\ 0\ 1\ 1\ 0) \\
f_x &= -1 \\
f_y &= 1 \\
\delta_0 &= 1 \\
\delta &= 1
\end{aligned}
$$

This solution states that the expression $-x + y$ decreases during each iteration of the loop by at least 1, and is greater than 1 for all states that satisfy the loop guard.

1.2 Algorithm

Now we briefly summarize the above illustration as an algorithm. See [5] for its detailed description and pointers to the related work.

The ranking function generation algorithm takes as input a transition relation $\rho(v, v')$ given by a set of linear inequalities over the program variables and their primed versions.

$$\rho(v, v') = R \begin{pmatrix} v \\ v' \end{pmatrix} \leq r$$

Then, the condition that a vector of coefficients f for the variables v defines a linear ranking function is represented by the constraint

$$\exists f\ \exists \delta_0\ \exists \delta\ \forall v\ \forall v' : \delta \geq 1 \wedge \rho(v, v') \rightarrow (fv \geq \delta_0 \wedge fv' \leq fv - \delta)\ . \tag{3}$$

We apply Farkas' lemma to (3) and obtain the following existentially quantified linear constraints that can be solved using off-the-shelf Linear Programming tools.

$$\begin{array}{l} \exists f\ \exists \delta_0\ \exists \delta \\ \quad \exists \lambda\ \exists \mu : \\ \qquad \delta \geq 1 \wedge \\ \qquad \lambda \geq 0 \wedge \mu \geq 0 \wedge \\ \qquad \lambda R = (-f\ 0) \wedge \lambda r \leq -\delta_0 \wedge \\ \qquad \mu R = (-f\ f) \wedge \mu r \leq -\delta \end{array} \tag{4}$$

2 Interpolation

Interpolants are logical assertions over program states that can separate program states that satisfy a desired property from the ones that violate the property. Interpolants play an important role in automated abstraction of sets of program states and their automatic construction is a crucial building block for program verification tools. In this section we present an algorithm for the computation of linear interpolants. A unique feature of our algorithm is the possibility to bias the outcome using additional constraints.

2.1 Example

In program verification, interpolants are computed for formulas that are extracted from program paths, i.e., sequences of program statements that follow the control flow graph of the program. We illustrate the interpolant computation algorithm using a program path from Figure 1, and refer to [7] for a detailed description of the algorithm and a discussion of the related work.

Input We consider a path $\tau_1\tau_3\tau_5$, which corresponds to an execution of the program that does not enter the loop and fails the assert statement. This path does not modify the values of the program variables, but rather imposes a sequence of conditions $y \geq z \wedge x \geq y \wedge x + 1 \leq z$. Since this sequence is not satisfiable, a program verifier can issue an interpolation query that needs to compute a separation between the states that the program reaches after taking the transition τ_3 and the states that violate the assertion. Formally, we are interested in an

inequality $i_x x + i_y y + i_z z \leq i_0$, called an interpolant, such that

$$\begin{aligned}
&\exists i_x\ \exists i_y\ \exists i_z\ \exists i_0 \\
&\quad \forall x\ \forall y\ \forall z\ : \\
&\qquad ((y \geq z \wedge x \geq y) \rightarrow i_x x + i_y y + i_z z \leq i_0) \wedge \\
&\qquad ((i_x x + i_y y + i_z z \leq i_0 \wedge x + 1 \leq z) \rightarrow 0 \leq -1)
\end{aligned} \tag{5}$$

Furthermore, we require that $i_x x + i_y y + i_z z \leq i_0$ only refers to the variables that appear both in $y \geq z \wedge x \geq y$ and $x + 1 \leq z$, which are x and z. Hence, i_z needs to be equal to 0, which is ensured by the above constraint without any additional effort.

Constraints First we represent the sequence of conditions in matrix form as follows.

$$\begin{aligned}
&(y \geq z \wedge x \geq y \wedge x + 1 \leq z) = \\
&(-y + z \leq 0 \wedge -x + y \leq 0 \wedge x - z \leq -1) = \\
&\begin{pmatrix} 0 & -1 & 1 \\ -1 & 1 & 0 \\ 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \leq \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}
\end{aligned}$$

Since (5) contains universal quantification, we apply Farkas' to enable applicability of Linear Programming tools and obtain the following constraint.

$$\begin{aligned}
&\exists i_x\ \exists i_y\ \exists i_z\ \exists i_0 \\
&\quad \exists \lambda\ \exists \mu : \\
&\qquad \lambda \geq 0 \wedge \mu \geq 0 \wedge \\
&\qquad (\lambda\ \mu) \begin{pmatrix} 0 & -1 & 1 \\ -1 & 1 & 0 \\ 1 & 0 & -1 \end{pmatrix} = 0 \wedge (\lambda\ \mu) \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} \leq -1 \wedge \\
&\qquad (i_x\ i_y\ i_z) = \lambda \begin{pmatrix} 0 & -1 & 1 \\ -1 & 1 & 0 \end{pmatrix} \wedge i_0 = \lambda \begin{pmatrix} 0 \\ 0 \end{pmatrix}
\end{aligned} \tag{6}$$

This constraint uses two vectors λ and μ to represent the linear combination that derives the unsatisfiable inequality $0 \leq -1$. The vector λ tracks the first two inequalities, and μ tracks the third inequality.

Solution By solving (6) we obtain

$$\begin{aligned}
\lambda &= (1\ 1), \\
\mu &= 1\ , \\
i_x &= -1\ , \\
i_y &= 0\ , \\
i_z &= 1\ , \\
i_0 &= 0\ .
\end{aligned}$$

The resulting interpolant is $-x + z \leq 0$.

2.2 Algorithm

The above example illustrate a constraint-based interpolation algorithm proposed in [7]. We refer to [7] for its detailed description and pointers to the related work, while the presentation below briefly sketches a simplified version.

Our interpolation algorithm takes as input two sets of linear inequalities, $Av \leq a$ and $Bv \leq b$, that are mutually unsatisfiable and computes an interpolating inequality $iv \leq i_0$, which satisfies the following constraint.

$$\begin{array}{l} \exists i\ \exists i_0 \\ \quad \forall v : \\ \qquad (Av \leq a \rightarrow iv \leq i_0) \wedge \\ \qquad ((iv \leq i_0 \wedge Bv \leq b) \rightarrow 0 \leq -1) \end{array} \tag{7}$$

The above formulation yields an existentially quantified linear constraint by applying Farkas' lemma. As a result we obtain

$$\begin{array}{l} \exists i\ \exists i_0 \\ \quad \exists \lambda\ \exists \mu : \\ \qquad \lambda \geq 0 \wedge \mu \geq 0 \wedge \\ \qquad (\lambda\ \mu) \begin{pmatrix} A \\ B \end{pmatrix} = 0 \wedge (\lambda\ \mu) \begin{pmatrix} a \\ b \end{pmatrix} \leq -1 \wedge \\ \qquad i = \lambda A \wedge i_0 = \lambda a\ . \end{array} \tag{8}$$

The constraint-based approach to interpolant computation offers a unique opportunity to bias the resulting interpolant using additional constraints. That is, (6) can be extended with an additional constraint $C(\begin{smallmatrix} i \\ i_0 \end{smallmatrix}) \leq c$ that encode the bias condition.

3 Linear invariants

Invariants are assertions over program variables whose value does not change during program execution. In program verification invariants are used to describe sets of reachable program states, and are an indispensable tool for reasoning about program correctness. In this section, we show how invariants proving the non-reachability of the error location in the program can be computed by using constraint-based techniques, and present a testing-based approach for simplifying the resulting constraint generation task. Furthermore, we briefly present a close connection between invariant and bound generation. See [4, 3] for the corresponding algorithms and further details.

3.1 Example

We illustrate the invariant generation on the program shown in Figure 1 and construct an invariant that proves the non-reachability of the location ℓ_5, which serves as the error location.

Input Our goal is to compute two linear inequalities over program variables $p_x x + p_y y + p_z z \leq p_0$ and $q_x x + q_y y + q_z z \leq q_0$ for the locations ℓ_2 and ℓ_3, respectively, such that these inequalities (1) represent all program states that are reachable at the respective locations, (2) serve as an induction hypothesis for proving (1) by induction over the number of program steps required to reach a program state, and (3) imply that no program execution can reach the error location ℓ_5. We encode the conditions (1–3) on the unknown invariant coefficients as the following constraint.

$$\begin{aligned}
&\exists p_x\ \exists p_y\ \exists p_z\ \exists p_0\ \exists q_x\ \exists q_y\ \exists q_z\ \exists q_0 \\
&\quad \forall x\ \forall y\ \forall z\ \forall x'\ \forall y'\ \forall z' : \\
&\qquad (\rho_1 \to p_x x' + p_y y' + p_z z' \leq p_0) \wedge \\
&\qquad ((p_x x + p_y y + p_z z \leq p_0 \wedge \rho_2) \to p_x x' + p_y y' + p_z z' \leq p_0) \wedge \\
&\qquad ((p_x x + p_y y + p_z z \leq p_0 \wedge \rho_3) \to q_x x' + q_y y' + q_z z' \leq q_0) \wedge \\
&\qquad ((q_x x + q_y y + q_z z \leq p_0 \wedge \rho_4) \to 0 \leq 0) \wedge \\
&\qquad ((q_x x + q_y y + q_z z \leq p_0 \wedge \rho_5) \to 0 \leq -1)
\end{aligned} \tag{9}$$

For each program transition this constraint contains a corresponding conjunct. The conjunct ensures that given a set of states at the start location of the transition all states reachable by applying the transition are represented by the assertion associated with the destination location. For example, the first conjunct asserts that applying τ_1 on any state leads to a state represented by $p_x x + p_y y + p_z z \leq p_0$.

Constraints Since (9) contains universal quantification, we resort to the Farkas' lemma-based elimination, which yields the following constraint.

$$\begin{aligned}
&\exists p_x\ \exists p_y\ \exists p_z\ \exists p_0\ \exists q_x\ \exists q_y\ \exists q_z\ \exists q_0 \\
&\quad \exists \lambda_1\ \exists \lambda_2\ \exists \lambda_3\ \exists \lambda_4\ \exists \lambda_5 : \\
&\qquad \lambda_1 \geq 0 \wedge \cdots \wedge \lambda_5 \geq 0 \wedge \\
&\qquad \lambda_1 R_1 = (0\ p_x\ p_y\ p_z) \wedge \lambda_1 r_1 \leq p_0 \wedge \\
&\qquad \lambda_2 \begin{pmatrix} p_x\ p_y\ p_z\ 0 \\ R_2 \end{pmatrix} = (0\ p_x\ p_y\ p_z) \wedge \lambda_2 \begin{pmatrix} p_0 \\ r_2 \end{pmatrix} \leq p_0 \wedge \\
&\qquad \lambda_3 \begin{pmatrix} p_x\ p_y\ p_z\ 0 \\ R_3 \end{pmatrix} = (0\ q_x\ q_y\ q_z) \wedge \lambda_3 \begin{pmatrix} p_0 \\ r_3 \end{pmatrix} \leq q_0 \wedge \\
&\qquad \lambda_4 \begin{pmatrix} q_x\ q_y\ q_z\ 0 \\ R_4 \end{pmatrix} = 0 \wedge \lambda_4 \begin{pmatrix} q_0 \\ r_4 \end{pmatrix} \leq 0 \wedge \\
&\qquad \lambda_5 \begin{pmatrix} q_x\ q_y\ q_z\ 0 \\ R_5 \end{pmatrix} = 0 \wedge \lambda_5 \begin{pmatrix} q_0 \\ r_5 \end{pmatrix} \leq -1
\end{aligned} \tag{10}$$

Unfortunately, this constraint is non-linear since it contains multiplication between unknown components of $\lambda_1, \ldots, \lambda_5$ and the unknown coefficients $p_x, p_y, p_z, p_0, q_x, q_y, q_z, q_0$.

Solution For our program we obtain the following solution.

$$\begin{array}{cccc} & & \lambda_1 = (1\ 1\ 1\ 1) & \\ & & \lambda_2 = (1\ 0\ 1\ 1\ 1) & \\ & & \lambda_3 = (1\ 1\ 1\ 1\ 1) & \\ & & \lambda_4 = (0\ 0\ 0\ 0\ 0) & \\ & & \lambda_5 = (1\ 1\ 0\ 0\ 0) & \\ p_x = 0 & p_y = -1 & p_z = 1 & p_0 = 0 \\ q_x = -1 & q_y = 0 & q_z = 1 & q_0 = 0 \end{array}$$

This solution defines an invariant $-y + x \leq 0$ at the location ℓ_2 and $-x + z \leq 0$ at the location ℓ_3.

3.2 Algorithm

Next, we sketch the constraint-based invariant algorithm. See [2, 1, 4] for further details and a discussion of related work.

Input The algorithm takes as the first input a program $P = (v, pc, \mathcal{L}, \mathcal{T}, \ell_\mathcal{I}, \ell_\mathcal{E})$ consists of data variables v, a program counter variable pc, a finite set of control locations $\mathcal{L}$, a finite set of transitions $\mathcal{T}$, a start location $\ell_\mathcal{I} \in \mathcal{L}$, and an error location $\ell_\mathcal{E} \in \mathcal{L}$. Each transition $(\ell, \rho(v, v'), \ell') \in \mathcal{T}$ consists of a start location ℓ, a transition relation $\rho(v, v')$, and destination location $\ell' \in \mathcal{L}$.

As the second input, the algorithm takes a template map that assigns to each control location ℓ a set of linear inequalities over program variables $I_\ell v \leq i_\ell$ with unknown coefficients I_ℓ and i_ℓ. The goal of the algorithm is to find a valuation of these coefficients such that the following constraint holds.

$$\begin{array}{l} \exists I_{\ell \in \mathcal{L}}\ \exists i_{\ell \in \mathcal{L}} \\ \quad \forall v\ \forall v' : \\ \qquad (I_{\ell_\mathcal{I}} = 0 \wedge i_{\ell_\mathcal{I}} = 0) \wedge (I_{\ell_\mathcal{E}} = 0 \wedge i_{\ell_\mathcal{E}} = -1) \wedge \\ \qquad (\forall (\ell, \rho(v, v'), \ell') \in \mathcal{T} : \\ \qquad\quad (I_\ell v \leq i_\ell \wedge \rho(v, v')) \rightarrow I_{\ell'} v' \leq i_{\ell'}) \end{array} \tag{11}$$

First this constraint ensures that there is no restriction on the start state of the programs imposed by the template $I_{\ell_\mathcal{I}} v \leq i_{\ell_\mathcal{I}}$ at the start location $\ell_\mathcal{I}$. Then, the constraint requires that no execution reaches the error location, i.e., the corresponding template $I_{\ell_\mathcal{E}} v \leq i_{\ell_\mathcal{E}}$ yields an unsatisfiable set of inequalities. For each program transition the constraint requires that set of states reachable by taking this transition is captured by the respective sets of inequalities.

Constraints Since the constraint (11) contains universal quantification, we apply Farkas' lemma and obtain

$$\begin{aligned}
&\exists I_{\ell\in\mathcal{L}}\ \exists i_{\ell\in\mathcal{L}}\\
&\quad \exists \Lambda_{\tau\in\mathcal{T}}:\\
&\qquad (I_{\ell_\mathcal{I}} = 0 \wedge i_{\ell_\mathcal{I}} = 0) \wedge (I_{\ell_\mathcal{E}} = 0 \wedge i_{\ell_\mathcal{E}} = -1) \wedge\\
&\qquad (\forall \tau = (\ell, R(\begin{smallmatrix} v \\ v' \end{smallmatrix}) \leq r, \ell') \in \mathcal{T}: \\
&\qquad\quad \Lambda_\tau \geq 0 \wedge\\
&\qquad\quad \Lambda_\tau \begin{pmatrix} I_\ell & 0 \\ R & \end{pmatrix} = I_{\ell'} \wedge \Lambda_\tau \begin{pmatrix} i_\ell \\ r \end{pmatrix} \leq i_{\ell'})
\end{aligned} \tag{12}$$

We observe that the multiplication between Λ_ℓ on once side with I_ℓ and i_ℓ leads to non-linearity. In theory, the non-linear constraint (10) can be solved by quantifier elimination procedures over rationals/reals, however in practice constraints quickly become too difficult for such direct approach.

4 Combination with uninterpreted functions (optional material)

In the previous sections we showed how auxiliary assertions represented by linear inequalities can be generated using constraint-based techniques. In this section we show that these techniques can be directly extended to deal with assertions represented by linear arithmetic combined with uninterpreted functions. This combined theory plays in important role in program verification, where uninterpreted functions are used to abstract functions that too complex to be modeled precisely. The basis of the extension is the hierarchical approach to the combination of logical theories [6]. We refer to [7, 1] for constraint-based interpolation and invariant generation algorithms for the combination of linear arithmetic and uninterpreted functions. Next, we will illustrate the interpolation algorithm for linear arithmetic and function symbols using a small example.

Input The interpolation algorithm takes as input a pair of mutually unsatisfiable assertions φ and ψ shown below.

$$\begin{aligned}
\varphi = &\ (x \leq a \wedge a \leq y \wedge f(a) \leq 0)\\
\psi = &\ (y \leq b \wedge b \leq x \wedge 1 \leq f(b))
\end{aligned}$$

The proof of unsatisfiability requires reasoning about linear arithmetic and uninterpreted function, which we represent by the logical consequence relation $\models_{\text{LI+UIF}}$.

$$\varphi \wedge \psi \models_{\text{LI+UIF}} \bot$$

The goal of the interpolation algorithm is to construct an assertion χ such that

$$\begin{aligned} &\varphi \models_{\text{LI+UIF}} \chi \ , \\ &\chi \wedge \psi \models_{\text{LI+UIF}} \bot \ , \\ &\chi \text{ is expressed over common symbols of } \varphi \text{ and } \psi \ . \end{aligned} \tag{13}$$

Constraints and solution As common in reasoning about combined theories, we first apply a purification step that separates arithmetic constraints from the function applications as follows.

$$\begin{aligned} \varphi_{\text{LI}} = \ & (x \leq a \wedge a \leq y \wedge c \leq 0) \\ \psi_{\text{LI}} = \ & (y \leq b \wedge b \leq x \wedge 1 \leq d) \\ D = \ & \{c \mapsto f(a), d \mapsto f(b)\} \\ X = \ & \{a = b \rightarrow c = d\} \end{aligned}$$

The sets of inequalities φ_{LI} and ψ_{LI} do not have any function symbols, which were replaced by fresh variables. The mapping between these fresh variables and the corresponding function applications is give by the set D. The set X contains functionality axiom instances that we create for all pairs of occurrences of function applications. These instances are expressed in linear arithmetic. For our example there is only one such instance.

The hierarchical reasoning approach guarantees that instances collected in X are sufficient for proving the mutual unsatisfiability of the pure assertions φ_{LI} and ψ_{LI}, i.e.,

$$\varphi_{\text{LI}} \wedge \psi_{\text{LI}} \wedge \bigwedge X \models_{\text{LI}} \bot$$

Unfortunately we cannot apply an algorithm for interpolation in linear arithmetic on the unsatisfiable conjunction presented above since the axiom instance in X contains variables that appear both in φ_{LI} and ψ_{LI}, which will lead to an interpolation result that violates the third condition in 13.

Instead, we resort to a case-based reasoning as follows. First, we attempt to compute an interpolant by considering the pure assertions, but do not succeed since they are mutually satisfiable, i.e.,

$$\varphi_{\text{LI}} \wedge \psi_{\text{LI}} \not\models_{\text{LI}} \bot$$

Nevertheless, the conjunction of pure assertions implies the precondition for applying the functionality axiom instance from X, i.e.,

$$\varphi_{\text{LI}} \wedge \psi_{\text{LI}} \models_{\text{LI}} a = b$$

From this implication follows that we can compute intermediate terms that are represented over variables that are common to φ_{LI} and ψ_{LI}. Formally, we have

$$\begin{aligned} &\varphi_{\text{LI}} \wedge \psi_{\text{LI}} \models_{\text{LI}} a \leq y \wedge y \leq b \ , \\ &\varphi_{\text{LI}} \wedge \psi_{\text{LI}} \models_{\text{LI}} a \geq x \wedge x \geq b \ . \end{aligned}$$

We rearrange these implications and obtain the following implications.

$$\varphi_{\mathrm{LI}} \models_{\mathrm{LI}} x \leq a \wedge a \leq y$$
$$\psi_{\mathrm{LI}} \models_{\mathrm{LI}} y \leq b \wedge b \leq x$$

These implications are used by our interpolation algorithm to derive appropriate case reasoning, which will be presented later on. Furthermore, our algorithm creates an additional function application $f(y)$ together with a corresponding fresh variable e, which is used for the purification and is recorded in the set D.

$$D \;=\; \{c \mapsto f(a), d \mapsto f(b), e \mapsto f(y)\}$$

The first step of the case reasoning requires computing an interpolant for the following unsatisfiable conjunction.

$$(\varphi_{\mathrm{LI}} \wedge a = e) \wedge (\psi_{\mathrm{LI}} \wedge e = b) \models_{\mathrm{LI}} \bot$$

By applying the algorithm presented in Section 2 we obtain a partial interpolant $e \leq 0$ such that

$$\varphi_{\mathrm{LI}} \wedge a = e \models_{\mathrm{LI}} e \leq 0 \ ,$$
$$e \leq 0 \wedge \psi_{\mathrm{LI}} \wedge e = b \models_{\mathrm{LI}} \bot \ .$$

The partial interpolant is completed using the case reasoning information as follows.

$$\chi_{\mathrm{LI}} \;=\; (x \neq y \vee (x = y \wedge e \leq 0))$$

After replacing the fresh variables by the corresponding function applications we obtain the following interpolant χ for the original input φ and ψ.

$$\begin{aligned}\chi = \;& (x \neq y \vee (x = y \wedge e \leq 0))[f(q)/e] \\ =\; & x \neq y \vee (x = y \wedge f(q) \leq 0)\end{aligned}$$

References

1. D. Beyer, T. A. Henzinger, R. Majumdar, and A. Rybalchenko. Invariant synthesis for combined theories. In *VMCAI*, 2007.
2. M. Colón, S. Sankaranarayanan, and H. Sipma. Linear invariant generation using non-linear constraint solving. In *CAV*, 2003.
3. B. Cook, A. Gupta, S. Magill, A. Rybalchenko, J. Simsa, S. Singh, and V. Vafeiadis. Finding heap-bounds for hardware synthesis. In *FMCAD*, 2009.
4. A. Gupta, R. Majumdar, and A. Rybalchenko. From tests to proofs. In *TACAS*, 2009.
5. A. Podelski and A. Rybalchenko. A complete method for the synthesis of linear ranking functions. In *VMCAI*, 2004.
6. V. Sofronie-Stokkermans. Hierarchic reasoning in local theory extensions. In *CADE*, 2005.
7. V. Sofronie-Stokkermans and A. Rybalchenko. Constraint solving for interpolation. *J. of Symbolic Computation*, 2010. to appear.