Revisiting Amalgamation and Stong Amalgamation

Roberto Bruttomesso, **Silvio Ghilardi**, Silvio Ranise

UniMi Milano, FBK Trento

TOLO III - Tblisi July 26, 2012

Quantifier-free Interpolation

A first-order theory T has quantifier-free interpolation iff for every quantifier free formulae ϕ, ψ such that $\psi \wedge \phi$ is T-unsatisfiable, there exists a quantifier free formula θ such that:

(i) $T \vdash \psi \to \theta$;

(ii) $\theta \wedge \phi$ is not T-satisfiable:

(iii) only variables occurring both in ψ and in ϕ occur in θ.[1]

Quantifier-free interpolants are commonly used in formal verification during abstraction-refinement cycles (since [McMillan CAV 03], [McMillan TACAS 04], ...).

[1] Warning: in these slides we use free variables and free constants interchangeably.

Reachability: an example

Let's start explaining the story with a toy example.
Below we consider a program manipulating integer variables $\underline{x} = pc, x, y$ (here pc is the program counter indicating the current location).
The code of the program is translated to a formula $T(\underline{x}, \underline{x}')$ expressing the relation between current $\underline{x}$ and next $\underline{x}'$ state variables.
There's an error location we do not want to be reachable.

Reachability: an example

Concrete Program	Transition Formula $T(\underline{x}, \underline{x}')$
1: y := x; 2: while (x $\geq$ 1) { 3: x := x - 1; 4: y := y - 1; 5: } 6: if (y $\geq$ 1 && x $\leq$ 0) 7: ERROR;	$(pc = 0 \wedge pc' = 2 \wedge x' = x = y')$ $\vee$ $(pc = 2 \wedge x \geq 1 \wedge pc' = 2 \wedge x' = x - 1 \wedge y' = y - 1)$ $\vee$ $(pc = 2 \wedge y \geq 1 \wedge x \leq 0 \wedge pc' = 7 \wedge x' = x \wedge y' = y)$

Bounded Model Checking

Let $\underline{x}^{(0)}, \dots, \underline{x}^{(n)}$ renamed copies of the $\underline{x}$.
The error location is reachable in n steps (n fixed) iff the formula

$$pc^{(0)} = 0 \wedge T(\underline{x}^{(0)}, \underline{x}^{(1)}) \wedge \cdots \wedge T(\underline{x}^{(n)}, \underline{x}^{(n)}) \wedge pc^{(n)} = E$$

($E := 7$ is the error location) is satisfiable.
We need satisfiability of quantifier-free formulae modulo a theory to discharge this[2]
SMT-solvers (Z3, Yices, MathSat, CVC, ...) are the dedicated tools.

[2]NB: our quantifier free formulae have variables, so satisfiablity of ϕ means that there are a model of the theory and an assignment to the variables making ϕ true.

Combination results

Usually, many theories are involved together in these problems: e.g. linear (real or integer) arithmetic + datastructure theories (arrays, lists, stacks, etc.). These theories, taken separatedly, have quantifier-free fragments decidable for satisfiability.

Combination results

Usually, many theories are involved together in these problems: e.g. linear (real or integer) arithmetic + datastructure theories (arrays, lists, stacks, etc.). These theories, taken separatedly, have quantifier-free fragments decidable for satisfiability.

What does it happen if we join them? We need decidability transfer results and modular combined satisfiability algorithms.

Combination results

Usually, many theories are involved together in these problems: e.g. linear (real or integer) arithmetic + datastructure theories (arrays, lists, stacks, etc.). These theories, taken separatedly, have quantifier-free fragments decidable for satisfiability.

What does it happen if we join them? We need decidability transfer results and modular combined satisfiability algorithms.

Classical Nelson-Oppen works gives an answer:

Combination results

Theorem (Nelson-Oppen 1979)

Let T_1, T_2 be first-order theories whose signatures are disjoint and whose quantifier-free fragment is decidable for satisfiability. If T_1, T_2 are both stably infinite, then $T_1 \cup T_2$ still has decidable quantifier-free fragment.

Combination results

Theorem (Nelson-Oppen 1979)

Let T_1, T_2 be first-order theories whose signatures are disjoint and whose quantifier-free fragment is decidable for satisfiability. If T_1, T_2 are both stably infinite, then $T_1 \cup T_2$ still has decidable quantifier-free fragment.

A first order theory T is stably infinite iff every model of T embeds into an infinite one. Without stable infiniteness, combined decidability can be lost [G. et al, IJCAR 2006].

Unbounded case

If we want to attack full verification (without a bound of the number of steps), we can proceed as follows.

Unbounded case

If we want to attack full verification (without a bound of the number of steps), we can proceed as follows.

Given, $\phi(\underline{x})$ and the transition $T(\underline{x}, \underline{x}')$, define

$$\begin{aligned} Pre_0(T, \phi) &:= \phi \\ Pre_n(T, \phi) &:= \exists \underline{x}' \, (T(\underline{x}, \underline{x}') \wedge Pre_{n-1}(T, \phi)). \end{aligned}$$

The formula $Pre_n(T, \phi)$ describes the set of states that can reach a state satisfying ϕ in n-steps.

Unbounded case

If we want to attack full verification (without a bound of the number of steps), we can proceed as follows.

Given, $\phi(\underline{x})$ and the transition $T(\underline{x}, \underline{x}')$, define

$$
\begin{aligned}
Pre_0(T, \phi) &:= \phi \\
Pre_n(T, \phi) &:= \exists \underline{x}' \, (T(\underline{x}, \underline{x}') \wedge Pre_{n-1}(T, \phi)).
\end{aligned}
$$

The formula $Pre_n(T, \phi)$ describes the set of states that can reach a state satisfying ϕ in n-steps.

Since T is usually a disjunction of guarded assignments (i.e. of formulae of the form $\psi(\underline{x}) \wedge \underline{x}' = \underline{t}(\underline{x})$ with quantifier-free ψ), it is easily checked that $Pre(T, \phi)$ is quantifier-free, in case ϕ is.

Unbounded case

Thus, we let ϕ be $pc = E$ (where E is the error location) and start computing

$$Pre_0(T, \phi), Pre_1(T, \phi), Pre_2(T, \phi), \ldots$$

untile either we find a formula $Pre_2(T, \phi)$ which is consistent with $pc = 0$ (which means that the program has a bug because 0 is the initial location), or until we stabilize, i.e. we get an n such that $Pre_n(T, \phi) \wedge \bigwedge_{m<n} \neg Pre_m(T, \phi)$ is unsatisfiable.

Unbounded case

Thus, we let ϕ be $pc = E$ (where E is the error location) and start computing

$$Pre_0(T, \phi), Pre_1(T, \phi), Pre_2(T, \phi), \dots$$

untile either we find a formula $Pre_2(T, \phi)$ which is consistent with $pc = 0$ (which means that the program has a bug because 0 is the initial location), or until we stabilize, i.e. we get an n such that $Pre_n(T, \phi) \wedge \bigwedge_{m<n} \neg Pre_m(T, \phi)$ is unsatisfiable.

Since all proof obbligations are quantifier-free and in the practical cases they involve stably infinite theories over disjoint signatures whose quantifier-free fragments are decidable, the plan is viable and SMT solvers can accomplish the task.

Unbounded case

The key problem is divergence. In our toy example, the preimages sequence gives

$$
\begin{aligned}
&pc = 7,\\
&pc = 2 \wedge y \geq 1 \wedge x \leq 0,\\
&pc = 2 \wedge y \geq 2 \wedge x = 1,\\
&pc = 2 \wedge y \geq 3 \wedge x = 2,\\
&\qquad\qquad \cdots
\end{aligned}
$$

Unbounded case

The key problem is divergence. In our toy example, the preimages sequence gives

$$
\begin{gathered}
pc = 7, \\
pc = 2 \land y \geq 1 \land x \leq 0, \\
pc = 2 \land y \geq 2 \land x = 1, \\
pc = 2 \land y \geq 3 \land x = 2, \\
\dots
\end{gathered}
$$

The idea is to make an abstraction of reachable states and to use interpolants to refine the abstraction.

Unbounded case

The key problem is divergence. In our toy example, the preimages sequence gives

$$\begin{gathered} pc = 7, \\ pc = 2 \wedge y \geq 1 \wedge x \leq 0, \\ pc = 2 \wedge y \geq 2 \wedge x = 1, \\ pc = 2 \wedge y \geq 3 \wedge x = 2, \\ \cdots \end{gathered}$$

The idea is to make an abstraction of reachable states and to use interpolants to refine the abstraction.

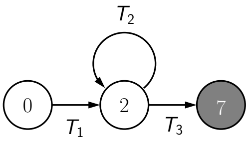
We show what happens in our case.

Interpolation Example

Interpolants in software verification

Original (Concrete) Program	Control Flow and Transitions

```
1:  y := x;
2:  while ( x ≥ 1 ) {
3:      x := x - 1;
4:      y := y - 1;
5:  }
6:  if ( y ≥ 1 && x ≤ 0 )
7:      ERROR;
```



$$T_1\colon\ \top \wedge \begin{cases} x' := x \\ y' := x \end{cases}$$

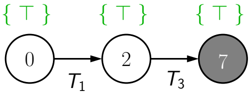
$$T_2\colon\ x \geq 1 \wedge \begin{cases} x' := x - 1 \\ y' := y - 1 \end{cases}$$

$$T_3\colon\ x \leq 0 \wedge y \geq 1 \wedge \begin{cases} x' := x \\ y' := y \end{cases}$$

Interpolation Example

Interpolants in software verification

(Abstract) Program Unwinding



true

$x_1 = x_0$

$y_1 = x_0$

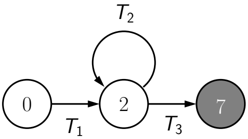
$x_1 \leq 0$

$y_1 \geq 1$

$x_2 = x_1$

$y_2 = y_1$

Control Flow and Transitions



$$T_1\colon\ \top \wedge \begin{cases} x' := x \\ y' := x \end{cases}$$

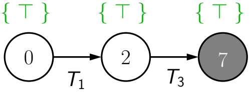
$$T_2\colon\ x \geq 1 \wedge \begin{cases} x' := x - 1 \\ y' := y - 1 \end{cases}$$

$$T_3\colon\ x \leq 0 \wedge y \geq 1 \wedge \begin{cases} x' := x \\ y' := y \end{cases}$$

Interpolation Example

Interpolants in software verification

(Abstract) Program Unwinding



{ ⊤ }

true

$x_1 = x_0$

$y_1 = x_0$

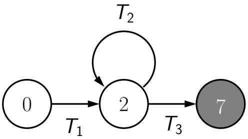
$x_1 \leq 0$

$y_1 \geq 1$

$x_2 = x_1$

$y_2 = y_1$

Control Flow and Transitions



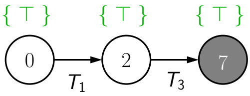
$$T_1\colon\ \top \wedge \begin{cases} x' := x \\ y' := x \end{cases}$$

$$T_2\colon\ x \geq 1 \wedge \begin{cases} x' := x - 1 \\ y' := y - 1 \end{cases}$$

$$T_3\colon\ x \leq 0 \wedge y \geq 1 \wedge \begin{cases} x' := x \\ y' := y \end{cases}$$

Interpolation Example
Interpolants in software verification

(Abstract) Program Unwinding	Control Flow and Transitions



$\{\ \top\ \}$

true

$x_1 = x_0$

$y_1 = x_0$

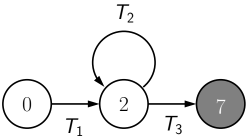
$\{\ y_1 - x_1 \le 0\ \}$

$x_1 \le 0$

$y_1 \ge 1$

$x_2 = x_1$

$y_2 = y_1$

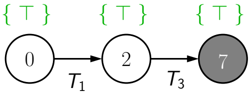


T_1: $\top \wedge \begin{cases} x' := x \\ y' := x \end{cases}$

T_2: $x \ge 1 \wedge \begin{cases} x' := x - 1 \\ y' := y - 1 \end{cases}$

T_3: $x \le 0 \wedge y \ge 1 \wedge \begin{cases} x' := x \\ y' := y \end{cases}$

(Abstract) Program Unwinding



$\{\ \top\ \}$

true

$x_1 = x_0$

$y_1 = x_0$

$\{\ y_1 - x_1 \leq 0\ \}$

$x_1 \leq 0$

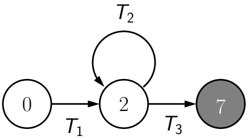
$y_1 \geq 1$

$x_2 = x_1$

$y_2 = y_1$

$\{\ \bot\ \}$

Control Flow and Transitions



$$T_1\!: \ \top \wedge \begin{cases} x' := x \\ y' := x \end{cases}$$

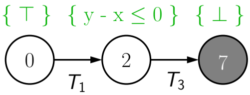
$$T_2\!: \ x \geq 1 \wedge \begin{cases} x' := x - 1 \\ y' := y - 1 \end{cases}$$

$$T_3\!: \ x \leq 0 \wedge y \geq 1 \wedge \begin{cases} x' := x \\ y' := y \end{cases}$$

Interpolation Example

Interpolants in software verification

(Abstract) Program Unwinding	Control Flow and Transitions



$\{\ \top\ \}$

true

$x_1 = x_0$

$y_1 = x_0$

$\{\ y_1 - x_1 \leq 0\ \}$

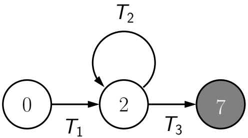
$x_1 \leq 0$

$y_1 \geq 1$

$x_2 = x_1$

$y_2 = y_1$

$\{\ \bot\ \}$



$$T_1\colon\ \top \wedge \begin{cases} x' := x \\ y' := x \end{cases}$$

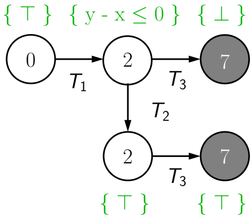
$$T_2\colon\ x \geq 1 \wedge \begin{cases} x' := x - 1 \\ y' := y - 1 \end{cases}$$

$$T_3\colon\ x \leq 0 \wedge y \geq 1 \wedge \begin{cases} x' := x \\ y' := y \end{cases}$$

Interpolation Example

Interpolants in software verification

(Abstract) Program Unwinding

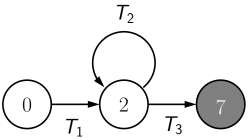


true
$x_1 = x_0$
$y_1 = x_0$

$x_1 \geq 1$
$x_2 = x_1 - 1$
$y_2 = y_1 - 1$

$x_2 \leq 0$
$y_2 \geq 1$
$x_3 = x_2$
$y_3 = y_2$

Control Flow and Transitions



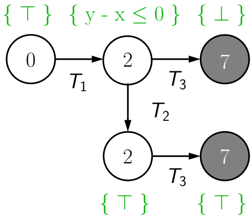
$$T_1\colon\ \top \wedge \begin{cases} x' := x \\ y' := x \end{cases}$$

$$T_2\colon\ x \geq 1 \wedge \begin{cases} x' := x - 1 \\ y' := y - 1 \end{cases}$$

$$T_3\colon\ x \leq 0 \wedge y \geq 1 \wedge \begin{cases} x' := x \\ y' := y \end{cases}$$

Interpolation Example
Interpolants in software verification

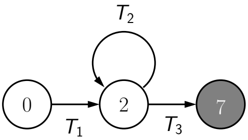
(Abstract) Program Unwinding	Control Flow and Transitions



{ ⊤ }
true
$x_1 = x_0$
$y_1 = x_0$

$x_1 \geq 1$
$x_2 = x_1$ - 1
$y_2 = y_1$ - 1

$x_2 \leq 0$
$y_2 \geq 1$
$x_3 = x_2$
$y_3 = y_2$

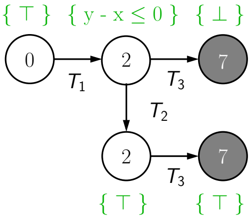


$$T_1\colon\ \top \wedge \begin{cases} x' := x \\ y' := x \end{cases}$$

$$T_2\colon\ x \geq 1 \wedge \begin{cases} x' := x - 1 \\ y' := y - 1 \end{cases}$$

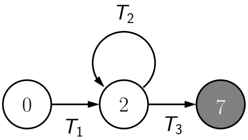
$$T_3\colon\ x \leq 0 \wedge y \geq 1 \wedge \begin{cases} x' := x \\ y' := y \end{cases}$$

(Abstract) Program Unwinding	Control Flow and Transitions



{ ⊤ }
true
$x_1 = x_0$
$y_1 = x_0$
{ $y_1 - x_1 \leq 0$ }
$x_1 \geq 1$
$x_2 = x_1 - 1$
$y_2 = y_1 - 1$

$x_2 \leq 0$
$y_2 \geq 1$
$x_3 = x_2$
$y_3 = y_2$



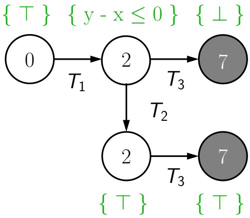
$$T_1\colon\ \top \wedge \begin{cases} x' := x \\ y' := x \end{cases}$$

$$T_2\colon\ x \geq 1 \wedge \begin{cases} x' := x - 1 \\ y' := y - 1 \end{cases}$$

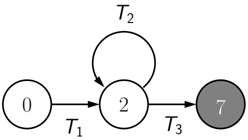
$$T_3\colon\ x \leq 0 \wedge y \geq 1 \wedge \begin{cases} x' := x \\ y' := y \end{cases}$$

Interpolation Example
Interpolants in software verification

(Abstract) Program Unwinding	Control Flow and Transitions



$\{ \top \}$
true
$x_1 = x_0$
$y_1 = x_0$
$\{ y_1 - x_1 \leq 0 \}$
$x_1 \geq 1$
$x_2 = x_1 - 1$
$y_2 = y_1 - 1$
$\{ y_2 - x_2 \leq 0 \}$
$x_2 \leq 0$
$y_2 \geq 1$
$x_3 = x_2$
$y_3 = y_2$



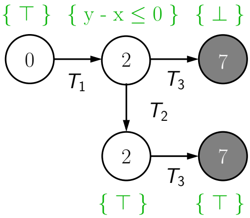
$$T_1\colon\ \top \wedge \begin{cases} x' := x \\ y' := x \end{cases}$$

$$T_2\colon\ x \geq 1 \wedge \begin{cases} x' := x - 1 \\ y' := y - 1 \end{cases}$$

$$T_3\colon\ x \leq 0 \wedge y \geq 1 \wedge \begin{cases} x' := x \\ y' := y \end{cases}$$

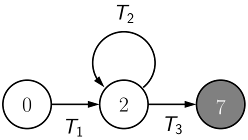
Interpolation Example
Interpolants in software verification

(Abstract) Program Unwinding



{ ⊤ }
true
$x_1 = x_0$
$y_1 = x_0$
{ $y_1 - x_1 \leq 0$ }
$x_1 \geq 1$
$x_2 = x_1 - 1$
$y_2 = y_1 - 1$
{ $y_2 - x_2 \leq 0$ }
$x_2 \leq 0$
$y_2 \geq 1$
$x_3 = x_2$
$y_3 = y_2$
{ ⊥ }

Control Flow and Transitions

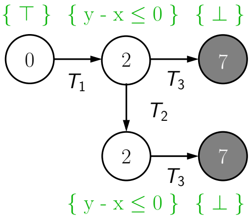


$$T_1\colon\ \top \wedge \begin{cases} x' := x \\ y' := x \end{cases}$$

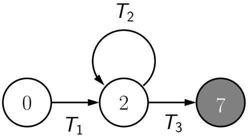
$$T_2\colon\ x \geq 1 \wedge \begin{cases} x' := x - 1 \\ y' := y - 1 \end{cases}$$

$$T_3\colon\ x \leq 0 \wedge y \geq 1 \wedge \begin{cases} x' := x \\ y' := y \end{cases}$$

(Abstract) Program Unwinding	Control Flow and Transitions



{ $\top$ }
true
$x_1 = x_0$
$y_1 = x_0$
{ $y_1 - x_1 \leq 0$ }
$x_1 \geq 1$
$x_2 = x_1 - 1$
$y_2 = y_1 - 1$
{ $y_2 - x_2 \leq 0$ }
$x_2 \leq 0$
$y_2 \geq 1$
$x_3 = x_2$
$y_3 = y_2$
{ $\bot$ }



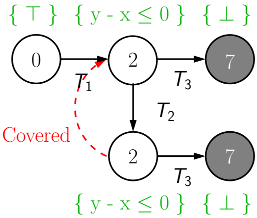
$$T_1\colon\ \top \wedge \begin{cases} x' := x \\ y' := x \end{cases}$$

$$T_2\colon\ x \geq 1 \wedge \begin{cases} x' := x - 1 \\ y' := y - 1 \end{cases}$$

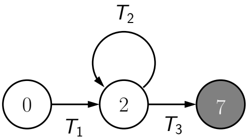
$$T_3\colon\ x \leq 0 \wedge y \geq 1 \wedge \begin{cases} x' := x \\ y' := y \end{cases}$$

Interpolation Example
Interpolants in software verification

(Abstract) Program Unwinding	Control Flow and Transitions



$\{ \top \}$
true
$x_1 = x_0$
$y_1 = x_0$
$\{ y_1 - x_1 \leq 0 \}$
$x_1 \geq 1$
$x_2 = x_1 - 1$
$y_2 = y_1 - 1$
$\{ y_2 - x_2 \leq 0 \}$
$x_2 \leq 0$
$y_2 \geq 1$
$x_3 = x_2$
$y_3 = y_2$
$\{ \perp \}$



$$T_1\colon\ \top \wedge \begin{cases} x' := x \\ y' := x \end{cases}$$

$$T_2\colon\ x \geq 1 \wedge \begin{cases} x' := x - 1 \\ y' := y - 1 \end{cases}$$

$$T_3\colon\ x \leq 0 \wedge y \geq 1 \wedge \begin{cases} x' := x \\ y' := y \end{cases}$$

Quantifier-free Interpolation

The success of the technology often depends on crucial heuristics guiding interpolation algorithms towards the production of good quality interpolants.

Quantifier-free Interpolation

The success of the technology often depends on crucial heuristics guiding interpolation algorithms towards the production of good quality interpolants.
Many theories used in software verification have quantifier-free interpolants:

Quantifier-free Interpolation

The success of the technology often depends on crucial heuristics guiding interpolation algorithms towards the production of good quality interpolants.

Many theories used in software verification have quantifier-free interpolants:

- linear real arithmetic (LA) [McMillan TACAS 04];

Quantifier-free Interpolation

The success of the technology often depends on crucial heuristics guiding interpolation algorithms towards the production of good quality interpolants.
Many theories used in software verification have quantifier-free interpolants:

- linear real arithmetic (LA) [McMillan TACAS 04];
- Presburger arithmetic (PA) [Brillout et al. IJCAR 10];

Quantifier-free Interpolation

The success of the technology often depends on crucial heuristics guiding interpolation algorithms towards the production of good quality interpolants.
Many theories used in software verification have quantifier-free interpolants:

- linear real arithmetic (LA) [McMillan TACAS 04];
- Presburger arithmetic (PA) [Brillout et al. IJCAR 10];
- more generally, every theory having QE (but QE algorithms usually are not efficient);

Quantifier-free Interpolation

The success of the technology often depends on crucial heuristics guiding interpolation algorithms towards the production of good quality interpolants.

Many theories used in software verification have quantifier-free interpolants:

- linear real arithmetic (LA) [McMillan TACAS 04];
- Presburger arithmetic (PA) [Brillout et al. IJCAR 10];
- more generally, every theory having QE (but QE algorithms usually are not efficient);
- the theory (EUF) of equality with uninterpreted function symbols [McMillan TACAS 04], [Fuchs et al. TACAS 09];

Quantifier-free Interpolation

The success of the technology often depends on crucial heuristics guiding interpolation algorithms towards the production of good quality interpolants.
Many theories used in software verification have quantifier-free interpolants:

- linear real arithmetic (LA) [McMillan TACAS 04];
- Presburger arithmetic (PA) [Brillout et al. IJCAR 10];
- more generally, every theory having QE (but QE algorithms usually are not efficient);
- the theory (EUF) of equality with uninterpreted function symbols [McMillan TACAS 04], [Fuchs et al. TACAS 09];
- some combinations of the above like (LA)+(EUF) [McMillan TACAS 04].

The theory $\mathcal{AX}_{\mathrm{ext}}$ of arrays with extensionality

This is an important theory in verification:

- we have three sorts `INDEX`, `ELEM`, `ARRAY`;

The theory $\mathcal{AX}_{\rm ext}$ of arrays with extensionality

This is an important theory in verification:

- we have three sorts $\texttt{INDEX}$, $\texttt{ELEM}$, $\texttt{ARRAY}$;
- besides equality, we have function symbols

$$rd : \texttt{ARRAY} \times \texttt{INDEX} \longrightarrow \texttt{ELEM},$$
$$wr : \texttt{ARRAY} \times \texttt{INDEX} \times \texttt{ELEM} \longrightarrow \texttt{ARRAY}$$

The theory $\mathcal{AX}_{\text{ext}}$ of arrays with extensionality

This is an important theory in verification:

- we have three sorts $\mathtt{INDEX}$, $\mathtt{ELEM}$, $\mathtt{ARRAY}$;
- besides equality, we have function symbols

$$rd : \mathtt{ARRAY} \times \mathtt{INDEX} \longrightarrow \mathtt{ELEM},$$
$$wr : \mathtt{ARRAY} \times \mathtt{INDEX} \times \mathtt{ELEM} \longrightarrow \mathtt{ARRAY}$$

- as axioms, we have

$$\forall y, i, e. \qquad rd(wr(y, i, e), i) = e \tag{1}$$
$$\forall y, i, j, e. \qquad i \neq j \Rightarrow rd(wr(y, i, e), j) = rd(y, j) \tag{2}$$
$$\forall x, y. \qquad x \neq y \Rightarrow (\exists i.\ rd(x, i) \neq rd(y, i)) \tag{3}$$

The theory $\mathcal{AX}_{\rm ext}$ of arrays with extensionality

Unfortunately, $\mathcal{AX}_{\rm ext}$ does not have interpolation, witness the following well-known counterexample (due to Ranjit Jhala).

The theory $\mathcal{AX}_{\text{ext}}$ of arrays with extensionality

Unfortunately, $\mathcal{AX}_{\text{ext}}$ does not have interpolation, witness the following well-known counterexample (due to Ranjit Jhala).

$$\begin{aligned} A := & \quad \{a = wr(b, i, e)\} \\ B := & \quad \{rd(a, j_1) \neq rd(b, j_1), rd(a, j_2) \neq rd(b, j_2), j_1 \neq j_2\} \end{aligned}$$

The theory $\mathcal{AX}_{\mathrm{ext}}$ of arrays with extensionality

Unfortunately, $\mathcal{AX}_{\mathrm{ext}}$ does not have interpolation, witness the following well-known counterexample (due to Ranjit Jhala).

$$
\begin{aligned}
A &:= \quad \{a = wr(b, i, e)\} \\
B &:= \quad \{rd(a, j_1) \neq rd(b, j_1), rd(a, j_2) \neq rd(b, j_2), j_1 \neq j_2\}
\end{aligned}
$$

Take ψ, ϕ to be the conjunctions of the literals from A, B, respectively. Then $\psi \wedge \phi$ is $\mathcal{AX}_{\mathrm{ext}}$-unsatisfiable, but no quantifier-free interpolant exists (notice that it should mention only a, b).

The theory $\mathcal{AX}_{\texttt{diff}}$ of arrays with diff

Since $\mathcal{AX}_{\mathrm{ext}}$ does not have quantifier-free interpolants, we propose the following variant, which we call $\mathcal{AX}_{\texttt{diff}}$. We add a further symbol in the signature

$$\texttt{diff} : \texttt{ARRAY} \times \texttt{ARRAY} \longrightarrow \texttt{INDEX}$$

The theory $\mathcal{AX}_{\texttt{diff}}$ of arrays with diff

Since $\mathcal{AX}_{\rm ext}$ does not have quantifier-free interpolants, we propose the following variant, which we call $\mathcal{AX}_{\texttt{diff}}$. We add a further symbol in the signature

$$\texttt{diff} : \texttt{ARRAY} \times \texttt{ARRAY} \longrightarrow \texttt{INDEX}$$

We replace the extensionality axiom (3) by its skolemization

$$\forall x, y. \qquad x \neq y \Rightarrow \ rd(x, \texttt{diff}(x, y)) \neq rd(y, \texttt{diff}(x, y))$$

The theory $\mathcal{AX}_{\mathtt{diff}}$ of arrays with diff

Since $\mathcal{AX}_{\mathrm{ext}}$ does not have quantifier-free interpolants, we propose the following variant, which we call $\mathcal{AX}_{\mathtt{diff}}$. We add a further symbol in the signature

$$\mathtt{diff} : \mathtt{ARRAY} \times \mathtt{ARRAY} \longrightarrow \mathtt{INDEX}$$

We replace the extensionality axiom (3) by its skolemization

$$\forall x, y. \qquad x \neq y \Rightarrow \; rd(x, \mathtt{diff}(x, y)) \neq rd(y, \mathtt{diff}(x, y))$$

Theorem (BGR RTA '11)

The theory $\mathcal{AX}_{\mathtt{diff}}$ has quantifier-free interpolation.

Our main concern

We investigate when quantifier-free interpolation transfers to combined theories (we assume signature disjointness).

Our main concern

We investigate when quantifier-free interpolation transfers to combined theories (we assume signature disjointness).

There are combination results [Yorsh-Musuvathi CADE 05], but often quantifier-free interpolation does not transfer to combined theories: for instance, in (PA)+(EUF) interpolants require quantifiers [Brillout et al. IJCAR 10].

Our main concern

We investigate when quantifier-free interpolation transfers to combined theories (we assume signature disjointness).

There are combination results [Yorsh-Musuvathi CADE 05], but often quantifier-free interpolation does not transfer to combined theories: for instance, in (PA)+(EUF) interpolants require quantifiers [Brillout et al. IJCAR 10].

We shall first take a semantic approach to clarify the situation.

Definition

A theory T has the *sub-amalgamation property* iff whenever we are given models $\mathcal{M}_1$ and $\mathcal{M}_2$ of T and a common substructure $\mathcal{A}$ of them, there exists a further model $\mathcal{M}$ of T endowed with embeddings $\mu_1 : \mathcal{M}_1 \longrightarrow \mathcal{M}$ and $\mu_2 : \mathcal{M}_2 \longrightarrow \mathcal{M}$ whose restrictions to $|\mathcal{A}|$ coincide.

$$\begin{tikzcd}
\mathcal{A} \arrow[r] \arrow[d] & \mathcal{M}_1 \arrow[d, "\mu_1"] \\
\mathcal{M}_2 \arrow[r, "\mu_2"'] & \mathcal{M}
\end{tikzcd}$$

Amalgamation

Theorem (Bacsich 75)

A (universal) theory T has the amalgamation property iff it has quantifier-free interpolation.

This theorem is useful both for negative and for positive results. It gives the essential information about existence of interpolants: once the essential information is achieved, concrete algorithms can be designed.

Strong Amalgamation

We need a stronger form of amalgamation for combined interpolation:

Definition

A theory T has the *strong sub-amalgamation property* iff whenever we are given models $\mathcal{M}_1$ and $\mathcal{M}_2$ of T and a common substructure $\mathcal{A}$ of them, there exists a further model $\mathcal{M}$ of T endowed with embeddings $\mu_1 : \mathcal{M}_1 \longrightarrow \mathcal{M}$ and $\mu_2 : \mathcal{M}_2 \longrightarrow \mathcal{M}$ whose restrictions to $|\mathcal{A}|$ coincide. Moreover, the embeddings μ_1, μ_2 satisfy the following additional condition: if for some m_1, m_2 we have $\mu_1(m_1) = \mu_2(m_2)$, then there exists an element a in $|\mathcal{A}|$ such that $m_1 = a = m_2$.

No identification is made in the amalgamated model!

Strong Amalgamation

Theorem

Let T be a theory admitting quantifier-free interpolation and Σ be a signature disjoint from the signature of T containing at least a unary predicate symbol. Then, $T \cup EUF(\Sigma)$ has quantifier-free interpolation iff T has the strong sub-amalgamation property.

Strong Amalgamation

Theorem

Let T be a theory admitting quantifier-free interpolation and Σ be a signature disjoint from the signature of T containing at least a unary predicate symbol. Then, $T \cup EUF(\Sigma)$ has quantifier-free interpolation iff T has the strong sub-amalgamation property.

Here you are the relevant modularity result:

Strong Amalgamation

Theorem

Let T be a theory admitting quantifier-free interpolation and Σ be a signature disjoint from the signature of T containing at least a unary predicate symbol. Then, $T \cup EUF(\Sigma)$ has quantifier-free interpolation iff T has the strong sub-amalgamation property.

Here you are the relevant modularity result:

Theorem

Let T_1 and T_2 be two universal, stably infinite theories over disjoint signatures Σ_1 and Σ_2. If both T_1 and T_2 have the strong sub-amalgamation property, then so does $T_1 \cup T_2$. In particular, $T_1 \cup T_2$ admits quantifier-free interpolation.

Strong Amalgamation

In verification theory, people uses the following stronger property for a theory T:

Strong Amalgamation

In verification theory, people uses the following stronger property for a theory T:

Definition

Let T be a theory in a signature Σ; we say that T has the general quantifier-free interpolation property iff for every signature Σ' (disjoint from Σ) and for every ground $\Sigma \cup \Sigma'$-formulæ ϕ, ψ such that $\phi \wedge \psi$ is T-unsatisfiable, there is a ground formula θ such that:

(i) $T \vdash \psi \to \theta$;

(ii) $\theta \wedge \phi$ is not T-satisfiable:

(iii) all predicate, constants and function symbols from Σ' occurring in θ occur also in ϕ and in ψ.

Strong Amalgamation

This property implies quantifier-free interpolation property for the combined theory $T \cup EUF(\Sigma')$ and looks stronger than it. Nevertheless, we have

Strong Amalgamation

This property implies quantifier-free interpolation property for the combined theory $T \cup EUF(\Sigma')$ and looks stronger than it. Nevertheless, we have

Theorem

A theory T has the general quantifier free interpolation property iff it is strongly sub-amalgamable.

Strong Amalgamation

This property implies quantifier-free interpolation property for the combined theory $T \cup EUF(\Sigma')$ and looks stronger than it. Nevertheless, we have

Theorem

A theory T has the general quantifier free interpolation property iff it is strongly sub-amalgamable.

Thus, the interpolation property commonly used in verification corresponds to strong sub-amalgamability (not just to plain sub-amalgamability).

Strong Amalgamation Syntactically

For computational purposes, it is essential to have a syntactic characterization of strong amalgamability in order to design combined interpolation algorithms.

NOTATION. Given two finite tuples $\underline{t} \equiv t_1, \ldots, t_n$ and $\underline{v} \equiv v_1, \ldots, v_m$ of terms,

the notation $\underline{t} \cap \underline{v} \neq \emptyset$ stands for the formula $\bigvee_{i=1}^{n} \bigvee_{j=1}^{m} (t_i = v_j)$.

We use $\underline{t}_1 \underline{t}_2$ to denote the juxtaposition of the two tuples $\underline{t}_1$ and $\underline{t}_2$ of terms. So, for example, $\underline{t}_1 \underline{t}_2 \cap \underline{v} \neq \emptyset$ is equivalent to

$$(\underline{t}_1 \cap \underline{v} \neq \emptyset) \vee (\underline{t}_2 \cap \underline{v} \neq \emptyset) \ .$$

Strong Amalgamation Syntactically

Definition

A theory T is equality interpolating iff it has the quantifier-free interpolation property and satisfies the following condition:

- for every quintuple $\underline{x}, \underline{y}_1, \underline{z}_1, \underline{y}_2, \underline{z}_2$ of tuples of variables and pair of quantifier-free formulae $\delta_1(\underline{x}, \underline{z}_1, \underline{y}_1)$ and $\delta_2(\underline{x}, \underline{z}_2, \underline{y}_2)$ such that

$$\delta_1(\underline{x}, \underline{z}_1, \underline{y}_1) \wedge \delta_2(\underline{x}, \underline{z}_2, \underline{y}_2) \vdash_T \underline{y}_1 \cap \underline{y}_2 \neq \emptyset \tag{4}$$

  there exists a tuple $\underline{v}(\underline{x})$ of terms (called interpolant terms) such that

$$\delta_1(\underline{x}, \underline{z}_1, \underline{y}_1) \wedge \delta_2(\underline{x}, \underline{z}_2, \underline{y}_2) \vdash_T \underline{y}_1\underline{y}_2 \cap \underline{v} \neq \emptyset\ . \tag{5}$$

Strong Amalgamation Syntactically

As an example, consider IDL (= the theory of integers under zero, successor, predecessor, ordering). We have

$$a_1 \neq a_2 \wedge 3 \leq a_1 < 5 \wedge 3 \leq a_2 < 5 \wedge 3 \leq b < 5 \vdash a_1 a_2 \cap b \neq \emptyset$$

and in fact for ground $\underline{v} = 3, 4$

$$a_1 \neq a_2 \wedge 3 \leq a_1 < 5 \wedge 3 \leq a_2 < 5 \wedge 3 \leq b < 5 \vdash a_1 a_2 b \cap \underline{v} \neq \emptyset.$$

The following result is useful in order to find examples:

Strong Amalgamation Syntactically

As an example, consider IDL (= the theory of integers under zero, successor, predecessor, ordering). We have

$$a_1 \neq a_2 \wedge 3 \leq a_1 < 5 \wedge 3 \leq a_2 < 5 \wedge 3 \leq b < 5 \vdash a_1 a_2 \cap b \neq \emptyset$$

and in fact for ground $\underline{v} = 3, 4$

$$a_1 \neq a_2 \wedge 3 \leq a_1 < 5 \wedge 3 \leq a_2 < 5 \wedge 3 \leq b < 5 \vdash a_1 a_2 b \cap \underline{v} \neq \emptyset.$$

The following result is useful in order to find examples:

Theorem

A universal theory admitting quantifier elimination is equality interpolating.

Strong Amalgamation Syntactically

The main result is now the following:

Theorem

A theory T has the strong amalgamation property iff it is equality interpolating.

We are now in the position of making a large list of theories that can be combined while keeping quantifier-free interpolation property (all these theories are universal, stably infinite and strongly amalgamable/equality interpolating).

Strong Amalgamation Syntactically

- LA, IDL, UTVPI: show universal quantifier eliminating axiomatization;

Strong Amalgamation Syntactically

- LA, IDL, UTVPI: show universal quantifier eliminating axiomatization;
- PA (but with integer division modulo n, each n): idem;

Strong Amalgamation Syntactically

- LA, IDL, UTVPI: show universal quantifier eliminating axiomatization;
- PA (but with integer division modulo n, each n): idem;
- acyclic lists: idem;

Strong Amalgamation Syntactically

- LA, IDL, UTVPI: show universal quantifier eliminating axiomatization;
- PA (but with integer division modulo n, each n): idem;
- acyclic lists: idem;
- EUF: (easy) ad hoc argument;

Strong Amalgamation Syntactically

- LA, IDL, UTVPI: show universal quantifier eliminating axiomatization;
- PA (but with integer division modulo n, each n): idem;
- acyclic lists: idem;
- EUF: (easy) ad hoc argument;
- RDS (recursive data structures): by reduction to the previous case;

Strong Amalgamation Syntactically

- LA, IDL, UTVPI: show universal quantifier eliminating axiomatization;
- PA (but with integer division modulo n, each n): idem;
- acyclic lists: idem;
- EUF: (easy) ad hoc argument;
- RDS (recursive data structures): by reduction to the previous case;
- $\mathcal{AX}_{\texttt{diff}}$: (non trivial) ad hoc argument

Strong Amalgamation Syntactically

- LA, IDL, UTVPI: show universal quantifier eliminating axiomatization;
- PA (but with integer division modulo n, each n): idem;
- acyclic lists: idem;
- EUF: (easy) ad hoc argument;
- RDS (recursive data structures): by reduction to the previous case;
- $\mathcal{AX}_{\texttt{diff}}$: (non trivial) ad hoc argument
- ...

Strong Amalgamation Syntactically

- LA, IDL, UTVPI: show universal quantifier eliminating axiomatization;
- PA (but with integer division modulo n, each n): idem;
- acyclic lists: idem;
- EUF: (easy) ad hoc argument;
- RDS (recursive data structures): by reduction to the previous case;
- $\mathcal{AX}_{\mathtt{diff}}$: (non trivial) ad hoc argument
- ...

For convex theories, our notion of equality interpolating theory coincides with [YM] one, so all examples from there can be imported.

Beth definability property

Relationship between equality interpolating property and suitable variants of Beth definability property can be shown.

Beth definability property

Relationship between equality interpolating property and suitable variants of Beth definability property can be shown.

A primitive formula is obtained from a conjunction of literals by prefixing to it a string of existential quantifiers.

Beth definability property

Relationship between equality interpolating property and suitable variants of Beth definability property can be shown.

A primitive formula is obtained from a conjunction of literals by prefixing to it a string of existential quantifiers.

A theory T has the Beth definability property for these formulae iff:

- for every tuple of variables $\underline{x}$, for every further variable y and for every *primitive* formula $\theta(\underline{x}, y)$ such that $\theta(\underline{x}, y') \wedge \theta(\underline{x}, y'') \vdash_T y' = y''$, there is a term $v(\underline{x})$ such that $\theta(\underline{x}, y) \vdash_T y = v$.

Beth definability property

Theorem

A convex amalgamating first order theory T has the above Beth definability property iff it is equality interpolating.

A first order theory T is said to be *convex* iff for every conjunction of literals δ, if

$$\delta \vdash_T x_1 = y_1 \vee \cdots \vee x_n = y_n$$

$(n \geq 1)$ then there exists $i = 1, \ldots, n$ such that

$$\delta \vdash_T x_i = y_i \ .$$

Beth definability property

The above Beth definability property is equivalent to regularity of monomorphisms for the category $\mathbf{C_H}$ of models of a universal Horn theory $\mathbf{H}$ in a functional language.[3]

[3] The language must have at least a constant function, because we formulated the property using literals, not just atoms.

Beth definability property

The above Beth definability property is equivalent to regularity of monomorphisms for the category $\mathbf{C_H}$ of models of a universal Horn theory $\mathbf{H}$ in a functional language.[3]

This matches with old known results in universal algebra (see Tholen et al. 1982 and the literature quoted therein):

[3] The language must have at least a constant function, because we formulated the property using literals, not just atoms.

Beth definability property

The above Beth definability property is equivalent to regularity of monomorphisms for the category $\mathbf{C_H}$ of models of a universal Horn theory $\mathbf{H}$ in a functional language.[3]

This matches with old known results in universal algebra (see Tholen et al. 1982 and the literature quoted therein):

Theorem

Let $\mathbf{C_H}$ have the amalgamation property; then $\mathbf{C_H}$ has the strong amalgamation property iff epis in $\mathbf{C_H}$ are regular iff monos in $\mathbf{C_H}$ are regular.

[3] The language must have at least a constant function, because we formulated the property using literals, not just atoms.

Combined Interpolation Algorithm

We show here how to exploit equality interpolation in order to design a combined interpolation algorithm. We shall keep our exposition at a high and informal level.

Combined Interpolation Algorithm

We show here how to exploit equality interpolation in order to design a combined interpolation algorithm. We shall keep our exposition at a high and informal level.

We fix two stably infinite equality interpolating Σ_1, Σ_2-theories T_1, T_2 ($\Sigma_1 \cap \Sigma_2 = \emptyset$) and we suppose we have for both of them modules for deciding satisfiability of quantifier-free formulae, extracting interpolants from refutations, computing interpolant terms, etc.

Combined Interpolation Algorithm

We show here how to exploit equality interpolation in order to design a combined interpolation algorithm. We shall keep our exposition at a high and informal level.

We fix two stably infinite equality interpolating Σ_1, Σ_2-theories T_1, T_2 ($\Sigma_1 \cap \Sigma_2 = \emptyset$) and we suppose we have for both of them modules for deciding satisfiability of quantifier-free formulae, extracting interpolants from refutations, computing interpolant terms, etc.

We also fix finite sets of quantifiers-free formulae A, B such that $\bigwedge A \wedge \bigwedge B$ is not $T_1 \cup T_2$-satisfiable.

Combined Interpolation Algorithm

Conventions, notations and free assumptions on A, B:

- we replace variables with free constants;
- we assume that all atoms occurring in it are pure, i.e. either Σ_1- or Σ_2-atoms;
- constants, literals, formulae, etc. are called transparent if they contain either only free constants from A or only free constants from B;
- we shall manipulate only ground formulae built up from pure and transparent atoms;
- constants, literals, formulae, etc. are called shared if they contain only free constants occurring both in A and in B;
- we call A_i ($i = 1, 2$) the set of Σ_i-literals from A (same for B_i).

Combined Interpolation Algorithm

The following operation can be freely performed. Take a pure and transparent literal L (let it e.g. contain only A-symbols), make a case-split and add L or $\neg L$ to A (case-split interpolants can be combined).

Combined Interpolation Algorithm

The following operation can be freely performed. Take a pure and transparent literal L (let it e.g. contain only A-symbols), make a case-split and add L or $\neg L$ to A (case-split interpolants can be combined).

Call A-relevant (resp. B-relevant) the atoms occurring in A (resp. in B) plus equalities between transparent free constants. Because of Nelson-Oppen results, $A \cup B$ is consistent if (i) $A_i \cup B_i$ ($i = 1, 2$) are both T_i-consistent; (ii) all A-relevant and B-relevant atoms are decided; (iii) non transparent equalities between free constants are decided as well.

Combined Interpolation Algorithm

The following operation can be freely performed. Take a pure and transparent literal L (let it e.g. contain only A-symbols), make a case-split and add L or $\neg L$ to A (case-split interpolants can be combined).

Call A-relevant (resp. B-relevant) the atoms occurring in A (resp. in B) plus equalities between transparent free constants. Because of Nelson-Oppen results, $A \cup B$ is consistent if (i) $A_i \cup B_i$ ($i = 1, 2$) are both T_i-consistent; (ii) all A-relevant and B-relevant atoms are decided; (iii) non transparent equalities between free constants are decided as well.

So the problem is just how to decide non-transparent equalities between free constants. These cannot be added explicitly to A and B.

Combined Interpolation Algorithm

Suppose that we decided all relevant literals and that we implicitly decided all non transparent equalities negatively, i.e. we decided that $a = b$ never holds whenever the equality $a = b$ is not transparent.

Combined Interpolation Algorithm

Suppose that we decided all relevant literals and that we implicitly decided all non transparent equalities negatively, i.e. we decided that $a = b$ never holds whenever the equality $a = b$ is not transparent.

By the above, since $A \cup B$ is supposed not to be consistent, we must have that $A_i \wedge B_i \cup (\underline{a} \cap \underline{b} = \emptyset)$ is not T_i-consistent for some $i = 1, 2$ (we let $\underline{a} = a_1, \ldots, a_n$ be from A and $\underline{b} = b_1, \ldots, b_m$ be from B)

Combined Interpolation Algorithm

Suppose that we decided all relevant literals and that we implicitly decided all non transparent equalities negatively, i.e. we decided that $a = b$ never holds whenever the equality $a = b$ is not transparent.

By the above, since $A \cup B$ is supposed not to be consistent, we must have that $A_i \wedge B_i \cup (\underline{a} \cap \underline{b} = \emptyset)$ is not T_i-consistent for some $i = 1, 2$ (we let $\underline{a} = a_1, \dots, a_n$ be from A and $\underline{b} = b_1, \dots, b_m$ be from B)

Thus we have that

$$A_i \cup B_i \vdash_{T_i} (\underline{a} \cap \underline{b} \neq \emptyset)$$

(with $A_i \cup B_i$ alone T_i-consistent, otherwise we have our interpolant).

Combined Interpolation Algorithm

Since T_i is equality interpolating, there must exist shared Σ_i-ground terms $\underline{v} \equiv v_1, \ldots, v_p$ such that

$$A_i \cup B_i \vdash_{T_i} (\underline{a} \cap \underline{v} \neq \emptyset) \vee (\underline{b} \cap \underline{v} \neq \emptyset).$$

Combined Interpolation Algorithm

Since T_i is equality interpolating, there must exist shared Σ_i-ground terms $\underline{v} \equiv v_1, \ldots, v_p$ such that

$$A_i \cup B_i \vdash_{T_i} (\underline{a} \cap \underline{v} \neq \emptyset) \vee (\underline{b} \cap \underline{v} \neq \emptyset).$$

Thus the union of $A_i \cup \{\underline{a} \cap \underline{v} = \emptyset\}$ and of $B_i \cup \{\underline{b} \cap \underline{v} = \emptyset\}$ is not T_i-satisfiable and invoking the available interpolation algorithm for T_i, we can compute a ground shared Σ_i-formula θ such that

$$A \vdash_{T_i} \theta \vee \underline{a} \cap \underline{v} \neq \emptyset \quad \text{and} \quad \theta \wedge B \vdash_{T_i} \underline{b} \cap \underline{v} \neq \emptyset.$$

Since T_i is equality interpolating, there must exist shared Σ_i-ground terms $\underline{v} \equiv v_1, \ldots, v_p$ such that

$$A_i \cup B_i \vdash_{T_i} (\underline{a} \cap \underline{v} \neq \emptyset) \vee (\underline{b} \cap \underline{v} \neq \emptyset).$$

Thus the union of $A_i \cup \{\underline{a} \cap \underline{v} = \emptyset\}$ and of $B_i \cup \{\underline{b} \cap \underline{v} = \emptyset\}$ is not T_i-satisfiable and invoking the available interpolation algorithm for T_i, we can compute a ground shared Σ_i-formula θ such that

$$A \vdash_{T_i} \theta \vee \underline{a} \cap \underline{v} \neq \emptyset \quad \text{and} \quad \theta \wedge B \vdash_{T_i} \underline{b} \cap \underline{v} \neq \emptyset.$$

By case-split, we have $n * p + m * p$ alternatives in order to non-deterministically update A, B. For the first $n * p$ alternatives, we add some $a_i = v_j$ (for $1 \leq i \leq n$, $1 \leq j \leq p$) to A. For the last $m * p$ alternatives, we add θ to A and some $\{\theta, b_i = v_j\}$ to B (for $1 \leq i \leq m$, $1 \leq j \leq p$).

Combined Interpolation Algorithm

The key observation is that in all alternative there is a non-shared constant $a \in A$ (or $b \in B$) that becomes 'morally shared', in the sense that the updated A (resp. B) contains $a = v$ (resp. $b = v$) for some shared v. Morally shared constants are in fact shared for practical purposes, because it can be shown that they can be eliminated (by replacement with shared terms) from interpolants.

Combined Interpolation Algorithm

The key observation is that in all alternative there is a non-shared constant $a \in A$ (or $b \in B$) that becomes 'morally shared', in the sense that the updated A (resp. B) contains $a = v$ (resp. $b = v$) for some shared v. Morally shared constants are in fact shared for practical purposes, because it can be shown that they can be eliminated (by replacement with shared terms) from interpolants.

Thus, in the end, if we exhaustively apply case-split and the above procedure making constants shared, we must result in a situation where $A_i \cup B_i$ is T_i-inconsistent (for some $i = 1, 2$) and thus interpolants can be computed.

Thanks for attention!

YM conditions

We say that a theory T satisfies condition YMc iff it has the quantifier free interpolation property and for every pair y_1, y_2 of variables, for further tuples $\underline{x}, \underline{z}_1, \underline{z}_2$, for every pair of conjunctions of literals $\delta_1(\underline{x}, \underline{z}_1, y_1), \delta_2(\underline{x}, \underline{z}_2, y_2)$ such that

$$\delta_1(\underline{x}, \underline{z}_1, y_1) \wedge \delta_2(\underline{x}, \underline{z}_2, y_2) \vdash_T y_1 = y_2 \tag{6}$$

there exists a term $v(\underline{x})$ such that

$$\delta_1(\underline{x}, \underline{z}_1, y_1) \wedge \delta_2(\underline{x}, \underline{z}_2, y_2) \vdash_T y_1 = v \wedge y_2 = v. \tag{7}$$

Condition YMc is equivalent to our condition of being equality interpolating in case T is convex. In case T is not convex, YMc is insufficient for combined interpolation: there is an example of a theory T (the 'golden cuff links theory') that satisfies YMc but such that $T \cup \mathcal{EUF}$ does not have quantifier free interpolation.

YM conditions

We say that a theory T satisfies condition YMc iff it has the quantifier free interpolation property and for every tuples $\underline{x}$, $\underline{z}_1$, $\underline{z}_2$ of variables, further tuples $\underline{y}_1 = y_{11}, \dots, y_{1n}$, $\underline{y}_2 = y_{21}, \dots, y_{2n}$ of variables, and pairs $\delta_1(\underline{x}, \underline{z}_1, \underline{y}_1), \delta_2(\underline{x}, \underline{z}_2, \underline{y}_2)$ of conjunctions of literals,

$$\text{if } \delta_1(\underline{x}, \underline{z}_1, \underline{y}_1) \wedge \delta_2(\underline{x}, \underline{z}_2, \underline{y}_2) \vdash_T \bigvee_{i=1}^{n} (y_{1i} = y_{2i}) \text{ holds,}$$

then there exists a tuple $\underline{v}(\underline{x}) = v_1, \dots, v_n$ of terms such that

$$\delta_1(\underline{x}, \underline{z}_1, \underline{y}_1) \wedge \delta_2(\underline{x}, \underline{z}_2, \underline{y}_2) \vdash_T \bigvee_{i=1}^{n} (y_{1i} = v_i \wedge v_i = y_{2i}).$$

Condition YM is sufficient to guarantee combined quantifier free interpolation but it is too strong in this sense (it is stronger than our equality interpolating condition).