VTSA summer school 2015

Exploiting SMT for Verification of Infinite-State Systems

2. Interpolation in SMT and in Verification

Alberto Griggio
Fondazione Bruno Kessler – Trento, Italy

Outline

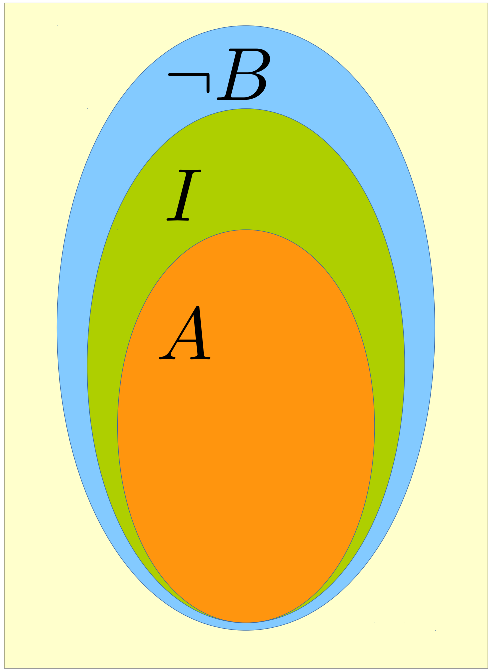
Introduction

Interpolants in Formal Verification

Computing interpolants in SMT

Introduction

- (Craig) Interpolant for an ordered pair (A, B) of formulae s.t. $A \wedge B \models_T \bot$ (or: $A \models_T \neg B$) is a formula I s.t.
  - $A \models_T I$
  - $I \wedge B \models_T \bot$ $(I \models_T \neg B)$
  - All the uninterpreted (in T) symbols of I are shared between A and B



- Why are interpolants useful?
  - Overapproximation of A relative to B
    - Overapprox. of $\exists_{\{x \notin B\}} \vec{x}.A$
  - “Local” explanation of why A is inconsistent with B

Importance of interpolation

Several important applications in formal verification:

- Approximate image computation for model checking of infinite-state systems
- Predicate discovery for Counterexample-Guided Abstraction Refinement
- Approximation of transition relation for infinite-state systems
- An alternative to (lazy) predicate abstraction for program verification
- Automatic generation of loop invariants
- ...

Outline

Introduction

Interpolants in Formal Verification

Computing interpolants in SMT

Background

Symbolic transition systems

- State variables X
- Initial states formula $I(X)$
- Transition relation formula $T(X, X')$

- A state σ is an assignment to the state vars $\bigwedge_{x_i \in X} x_i = v_i$
- A path of the system S is a sequence of states $\sigma_0, \ldots, \sigma_k$ such that $\sigma_0 \models I$ and $\sigma_i, \sigma'_{i+1} \models T$
- A *k*-step (symbolic) unrolling of S is a formula

$$I(X^0) \wedge \bigwedge_{i=0}^{k-1} T(X^i, X^{i+1})$$

  - Encodes all possible paths of length up to *k*
- A state property is a formula P over X
  - Encodes all the states σ such that $\sigma \models P$

Forward reachability checking

- Forward image computation
  - Compute all states reachable from σ in one transition:

$$\mathrm{Img}(\sigma(X)) := \exists X.\sigma(X) \wedge T(X, X')[X/X']$$

- Prove that a set of states $\mathrm{Bad}(X)$ is not reachable:

$R(X) := I(X)$

$\mathrm{Bad}(X)$

$\mathrm{Img}(R(X))$

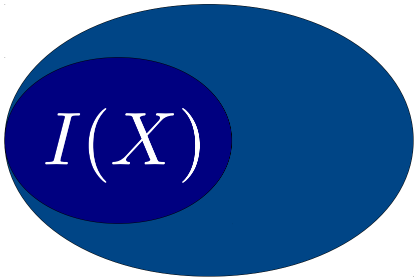
Forward reachability checking

- Forward image computation
  - Compute all states reachable from σ in one transition:

$$\mathrm{Img}(\sigma(X)) := \exists X.\sigma(X) \wedge T(X, X')[X/X']$$

- Prove that a set of states $\mathrm{Bad}(X)$ is not reachable:

$R(X) :=$



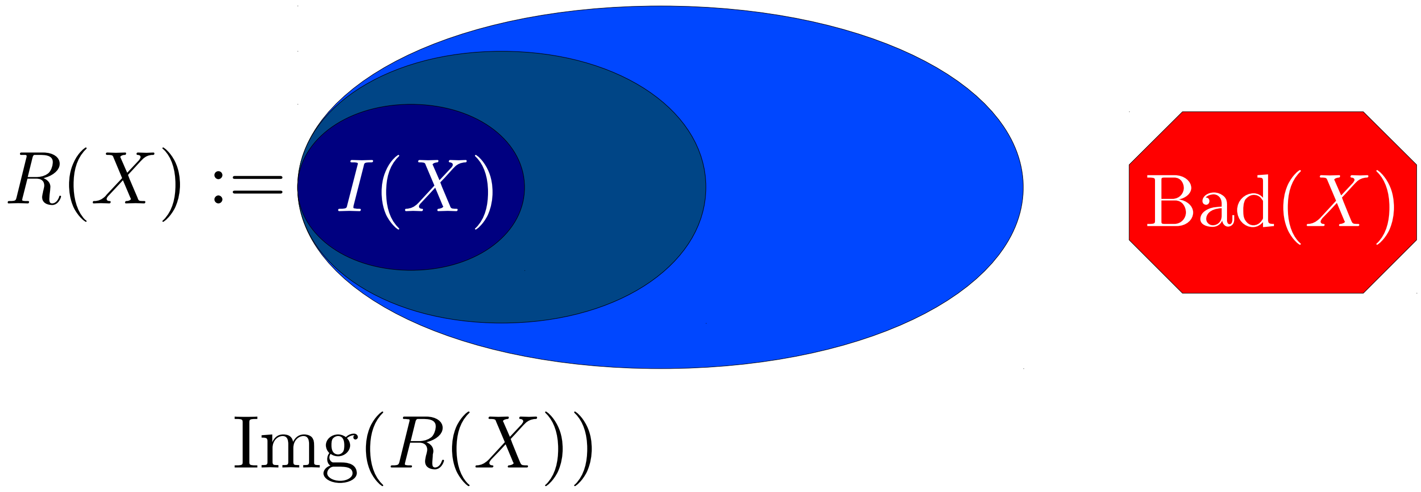
$\mathrm{Img}(R(X))$

Forward reachability checking

- Forward image computation
  - Compute all states reachable from σ in one transition:

$$\mathrm{Img}(\sigma(X)) := \exists X.\sigma(X) \wedge T(X, X')[X/X']$$

- Prove that a set of states $\mathrm{Bad}(X)$ is not reachable:

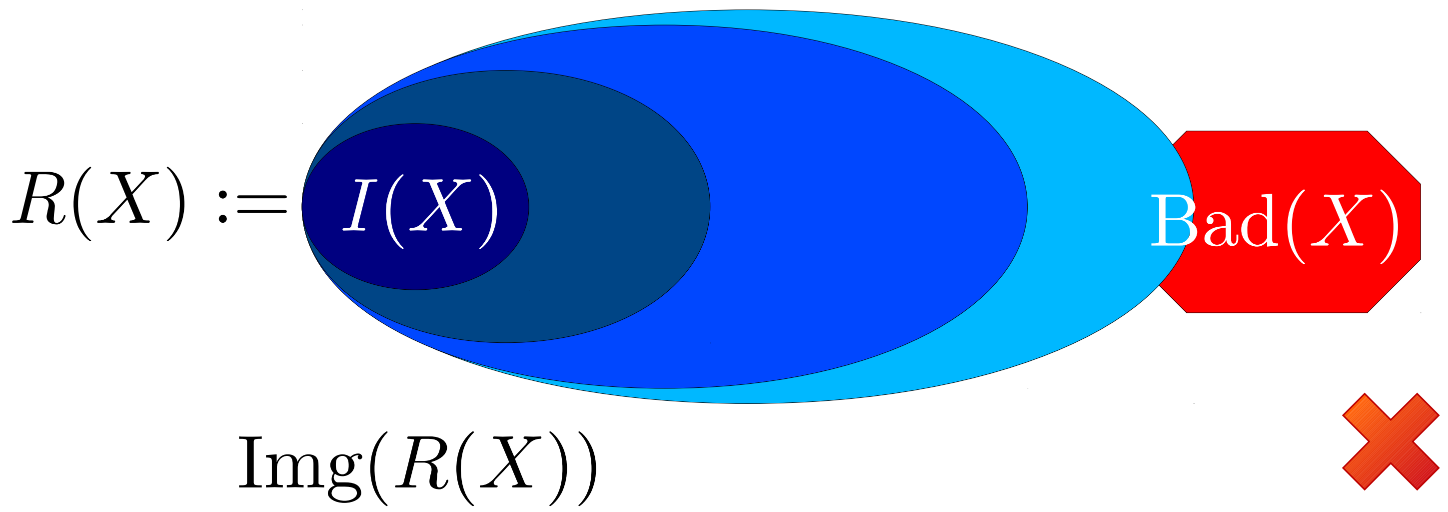


Forward reachability checking

- Forward image computation
  - Compute all states reachable from σ in one transition:

$$\mathrm{Img}(\sigma(X)) := \exists X.\sigma(X) \wedge T(X, X')[X/X']$$

- Prove that a set of states $\mathrm{Bad}(X)$ is not reachable:

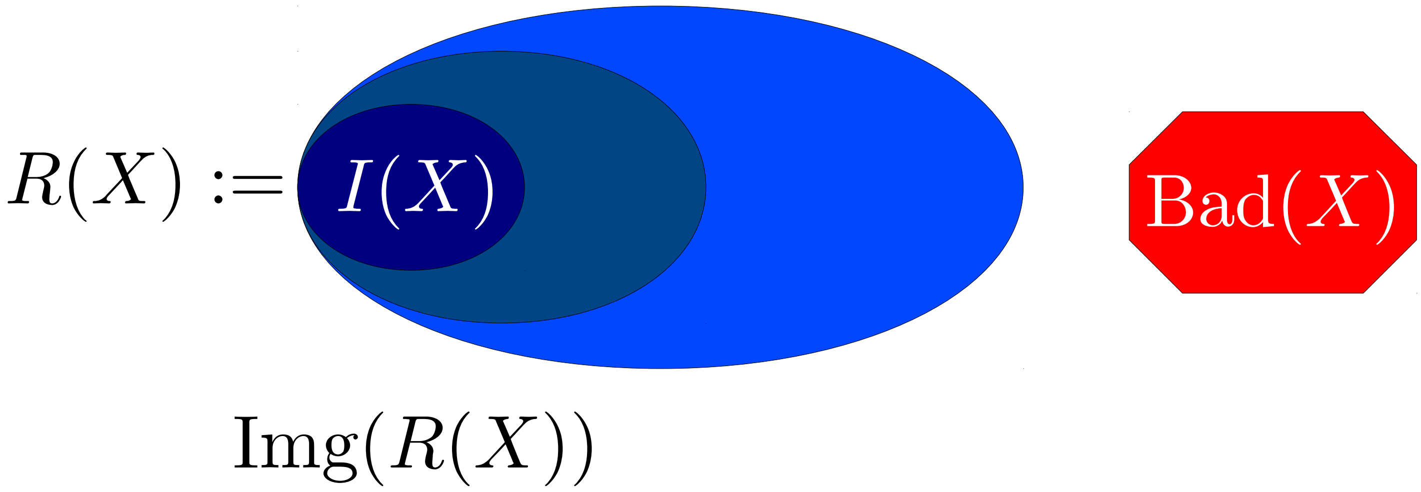


Forward reachability checking

- Forward image computation
  - Compute all states reachable from σ in one transition:

$$\mathrm{Img}(\sigma(X)) := \exists X.\sigma(X) \wedge T(X, X')[X/X']$$

- Prove that a set of states $\mathrm{Bad}(X)$ is not reachable:

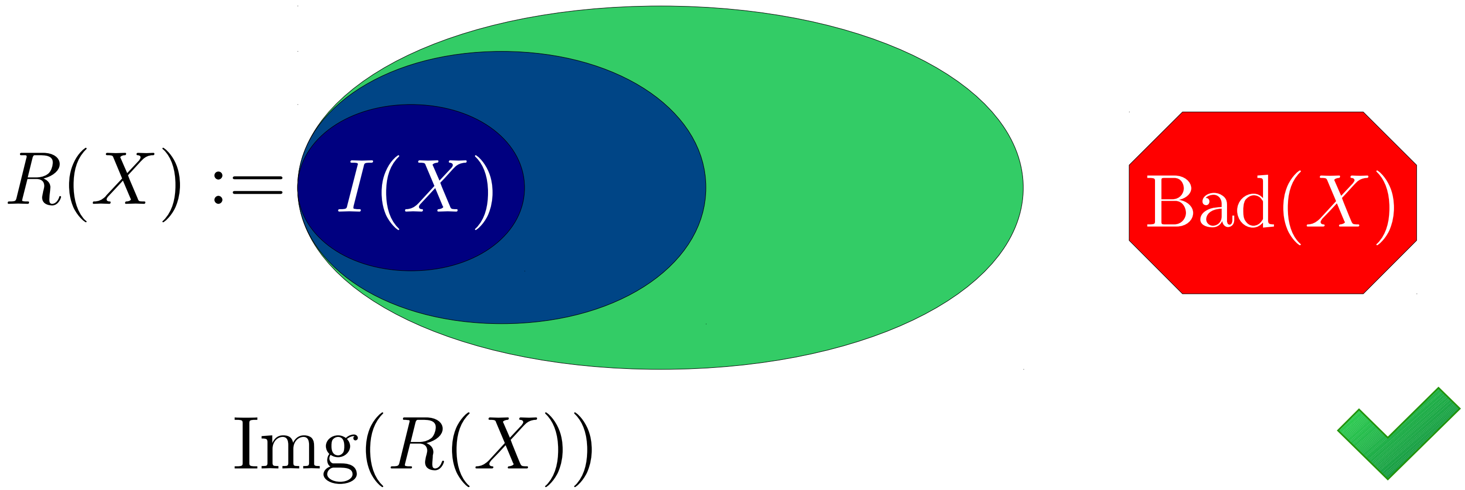


Forward reachability checking

- Forward image computation
  - Compute all states reachable from σ in one transition:

$$\mathrm{Img}(\sigma(X)) := \exists X.\sigma(X) \wedge T(X, X')[X/X']$$

- Prove that a set of states $\mathrm{Bad}(X)$ is not reachable:



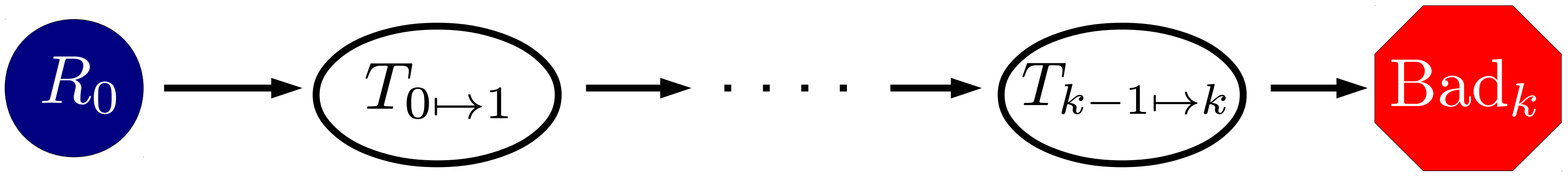
Interpolation-based reachability

- Image computation requires quantifier elimination, which is typically very expensive (both in theory and in practice)

- Interpolation-based algorithm (McMillan CAV'03): use interpolants to overapproximate image computation
  - much more efficient than the previous algorithm
    - interpolation is often much cheaper than quantifier elimination
    - abstraction (overapproximation) accelerates convergence
  - termination is still guaranteed for finite-state systems

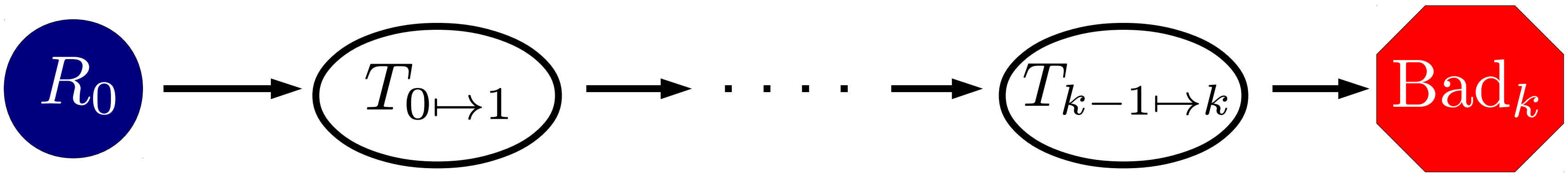
Interpolation-based reachability

- Set $R(X) := I(X)$
- Check satisfiability of $R_0 \wedge \bigwedge_{i=0}^{k-1} T_i \wedge \mathrm{Bad}_k$



Interpolation-based reachability

- Set $R(X) := I(X)$
- Check satisfiability of $R_0 \wedge \bigwedge_{i=0}^{k-1} T_i \wedge \mathrm{Bad}_k$

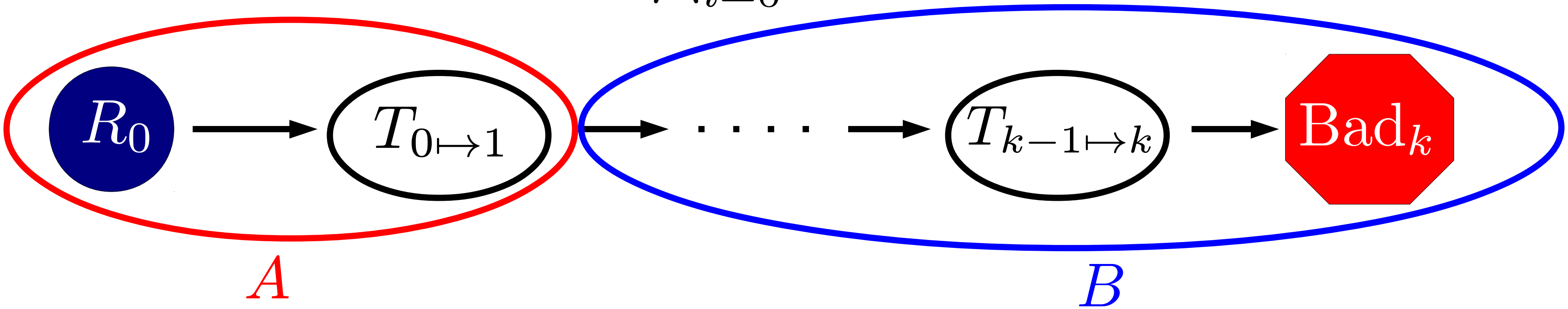


- If SAT:
  - If $R \equiv I$, return REACHABLE
  - else, increase *k* and repeat

the unrolling hits Bad

Interpolation-based reachability

- Set $R(X) := I(X)$
- Check satisfiability of $R_0 \wedge \bigwedge_{i=0}^{k-1} T_i \wedge \mathrm{Bad}_k$

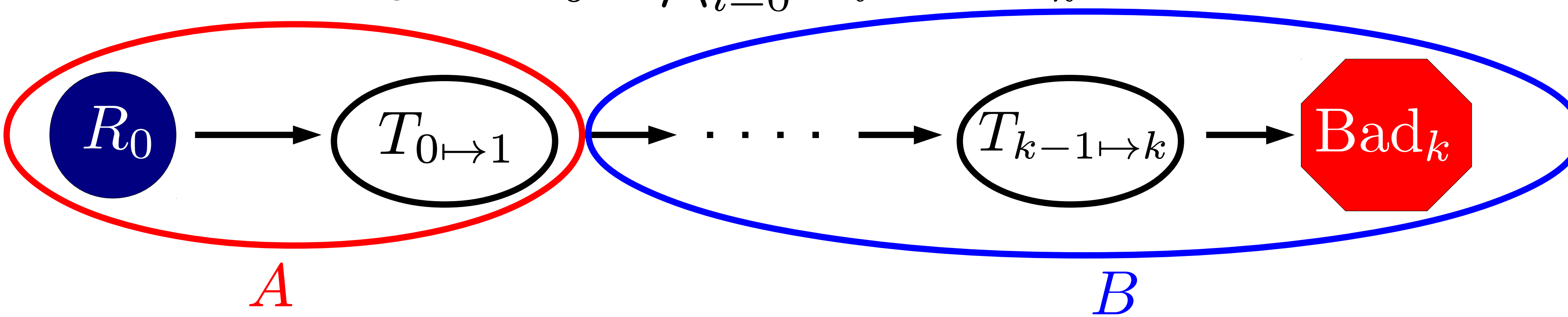


- If UNSAT:
  - Set $\varphi(X) := \mathrm{Interpolant}(A, B)[X'/X]$

φ is an abstraction of the forward image guided by the property

Interpolation-based reachability

- Set $R(X) := I(X)$
- Check satisfiability of $R_0 \wedge \bigwedge_{i=0}^{k-1} T_i \wedge \mathrm{Bad}_k$



- If UNSAT:
  - Set $\varphi(X) := \mathrm{Interpolant}(A, B)[X'/X]$

    φ is an abstraction of the forward image guided by the property

  - If $\varphi \models R$, return UNREACHABLE

    fixpoint found

  - else, set $R(X) := R(X) \vee \varphi(X)$ and continue

Interpolation-based Abstraction Refinement

(Lazy) Predicate abstraction

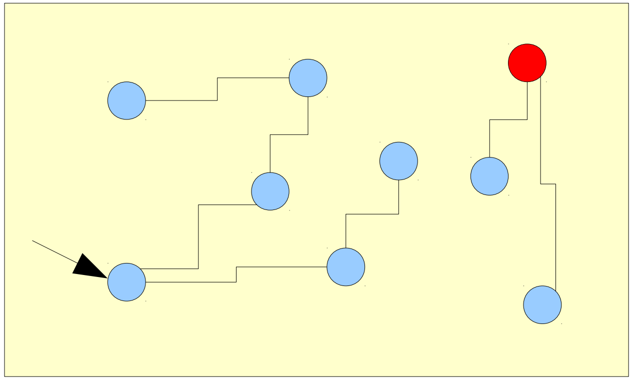
- Given a Transition System $S := (I, T)$ and predicates $\mathbb{P}$
  - Abstract initial states

$$\widehat{I(X)}_{\mathbb{P}} := \exists X.(I(X) \wedge \bigwedge_{p \in \mathbb{P}} (x_p \leftrightarrow p(X))[p(X)/x_p]$$

  - Abstract forward image

$$\widehat{\mathrm{Img}}(\varphi(X))_{\mathbb{P}} := \exists X, X', \vec{x_p}.(\varphi(X) \wedge T(X, X') \wedge \bigwedge_{p \in \mathbb{P}} (x_p \leftrightarrow p(X) \wedge x'_p \leftrightarrow p(X'))[p(X)/x'_p]$$

- Standard technique applied in many verification tools
  - In conjunction with counterexample-guided refinement (CEGAR)
    - Extract new predicates from spurious counterexamples and compute a more precise abstraction



Interpolation-based Abstraction Refinement

(Lazy) Predicate abstraction

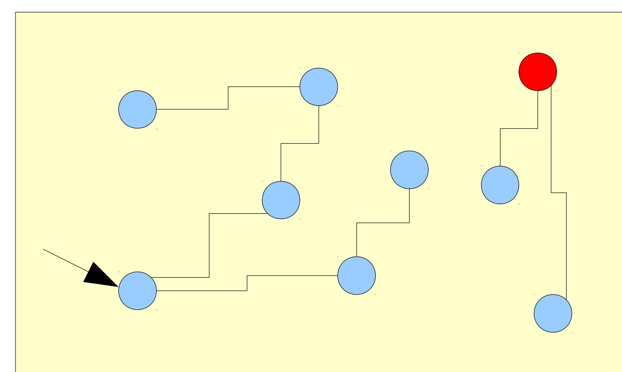
- Given a Transition System $S := (I, T)$ and predicates $\mathbb{P}$
  - Abstract initial states

    $\widehat{I(X)}_{\mathbb{P}} := \exists X.(I(X) \wedge \bigwedge_{p\in\mathbb{P}}(x_p \leftrightarrow p(X)))[p(X)/x_p]$
  - Abstract forward image

$$\widehat{\mathrm{Img}}(\varphi(X))_{\mathbb{P}} := \exists X, X', \vec{x_p}.(\varphi(X) \wedge T(X, X') \wedge \bigwedge_{p\in\mathbb{P}}(x_p \leftrightarrow p(X) \wedge x'_p \leftrightarrow p(X'))[p(X)/x'_p]$$

The strongest boolean combination of predicates in $\mathbb{P}$ that is implied by $\mathrm{Img}(\varphi(X))$

- Standard technique applied in many verification tools
  - In conjunction with counterexample-guided refinement (CEGAR)
    - Extract new predicates from spurious counterexamples and compute a more precise abstraction



Interpolation-based Abstraction Refinement

(Lazy) Predicate abstraction

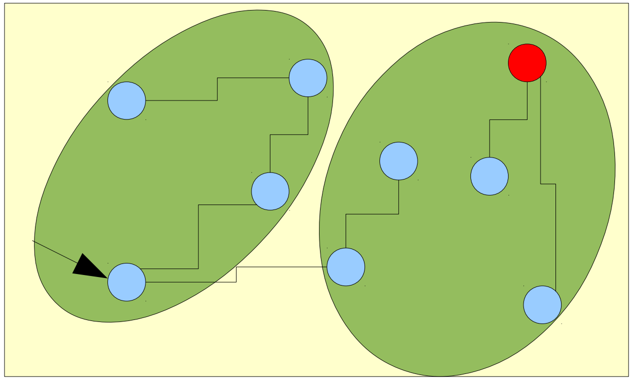
- Given a Transition System $S := (I, T)$ and predicates $\mathbb{P}$
  - Abstract initial states

$$\widehat{I(X)}_{\mathbb{P}} := \exists X.(I(X) \wedge \bigwedge_{p \in \mathbb{P}} (x_p \leftrightarrow p(X))[p(X)/x_p]$$

  - Abstract forward image

$$\widehat{\mathrm{Img}}(\varphi(X))_{\mathbb{P}} := \exists X, X', \vec{x_p}.(\varphi(X) \wedge T(X, X') \wedge \bigwedge_{p \in \mathbb{P}} (x_p \leftrightarrow p(X) \wedge x'_p \leftrightarrow p(X'))[p(X)/x'_p]$$

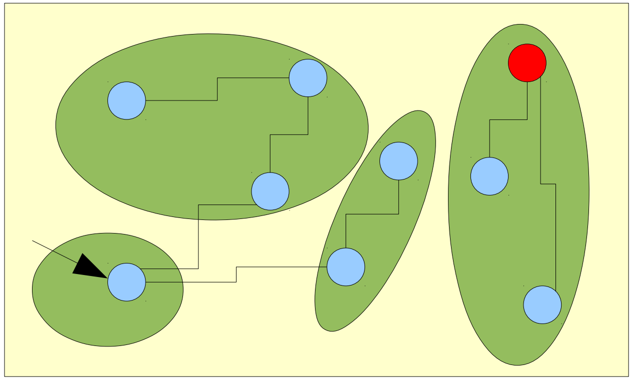
- Standard technique applied in many verification tools
  - In conjunction with counterexample-guided refinement (CEGAR)
    - Extract new predicates from spurious counterexamples and compute a more precise abstraction



Interpolation-based Abstraction Refinement

(Lazy) Predicate abstraction

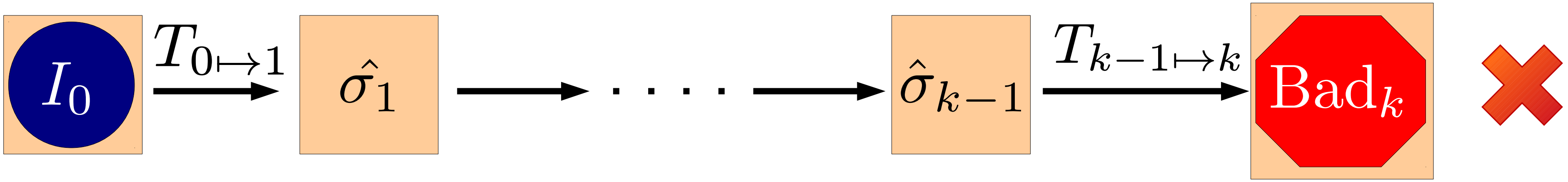
- Given a Transition System $S := (I, T)$ and predicates $\mathbb{P}$
  - Abstract initial states
    $\widehat{I(X)}_{\mathbb{P}} := \exists X.(I(X) \wedge \bigwedge_{p \in \mathbb{P}}(x_p \leftrightarrow p(X))[p(X)/x_p]$
  - Abstract forward image
    $\widehat{\mathrm{Img}}(\varphi(X))_{\mathbb{P}} := \exists X, X', \vec{x_p}.(\varphi(X) \wedge T(X, X') \wedge \bigwedge_{p \in \mathbb{P}}(x_p \leftrightarrow p(X) \wedge x'_p \leftrightarrow p(X'))[p(X)/x'_p]$



- Standard technique applied in many verification tools
  - In conjunction with counterexample-guided refinement (CEGAR)
    - Extract new predicates from spurious counterexamples and compute a more precise abstraction

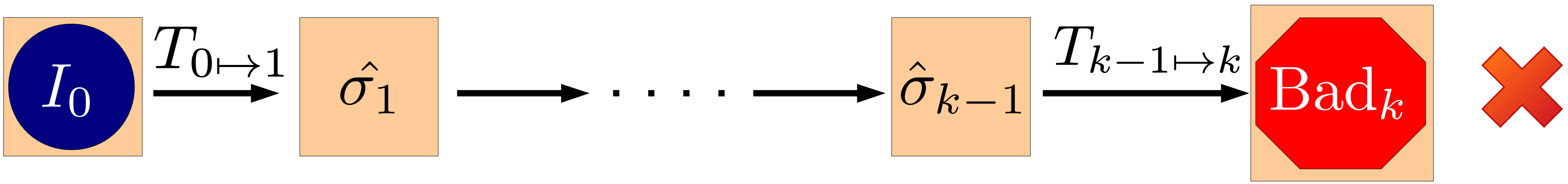
Interpolation-based Abstraction Refinement

- An abstract cex path $\hat{\sigma}_0, \ldots, \hat{\sigma}_k$ (wrt. $\mathbb{P}$) might be spurious
  - Because abstraction is overapproximating



Interpolation-based Abstraction Refinement

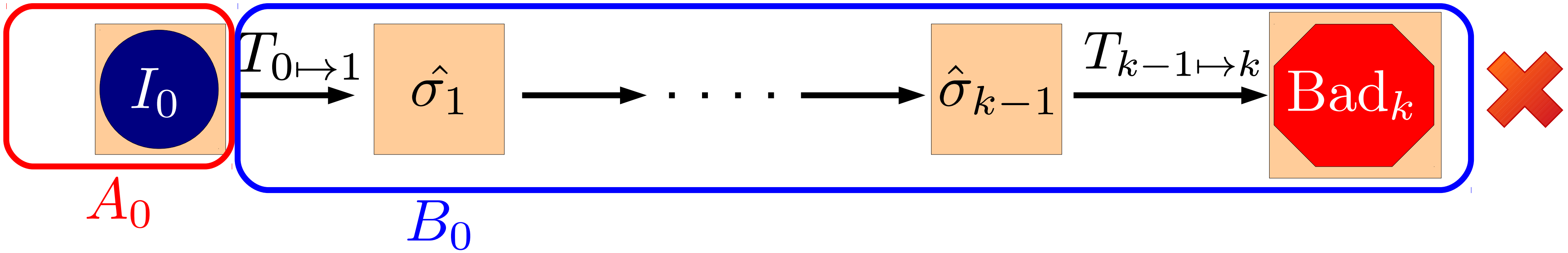
- An abstract cex path $\hat{\sigma}_0, \ldots, \hat{\sigma}_k$ (wrt. $\mathbb{P}$) might be spurious
  - Because abstraction is overapproximating



- Compute a sequence of interpolants $\varphi_0, \ldots, \varphi_{k-1}$ such that $T_{i \mapsto i+1} \wedge \varphi_i \models \varphi_{i+1}$ for all $i \in [0, k-1)$

Interpolation-based Abstraction Refinement

- An abstract cex path $\hat{\sigma}_0, \ldots, \hat{\sigma}_k$ (wrt. $\mathbb{P}$) might be spurious
  - Because abstraction is overapproximating

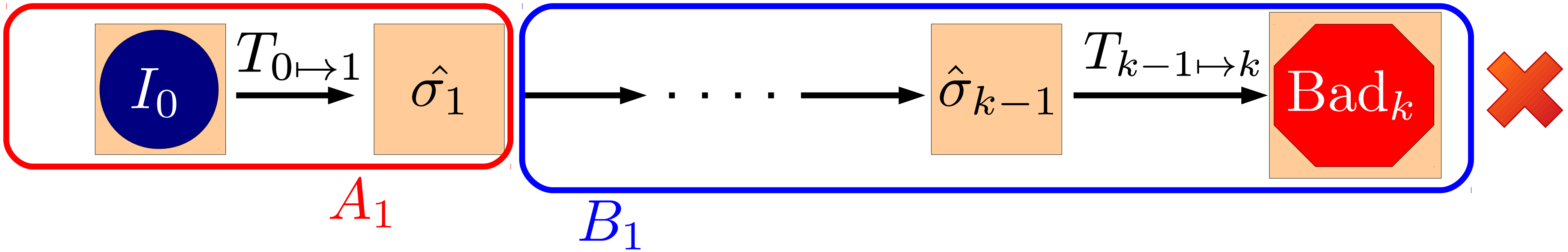


- Compute a sequence of interpolants $\varphi_0, \ldots, \varphi_{k-1}$

  such that $T_{i \mapsto i+1} \wedge \varphi_i \models \varphi_{i+1}$ for all $i \in [0, k-1)$

Interpolation-based Abstraction Refinement

- An abstract cex path $\hat{\sigma}_0, \ldots, \hat{\sigma}_k$ (wrt. $\mathbb{P}$) might be spurious
  - Because abstraction is overapproximating

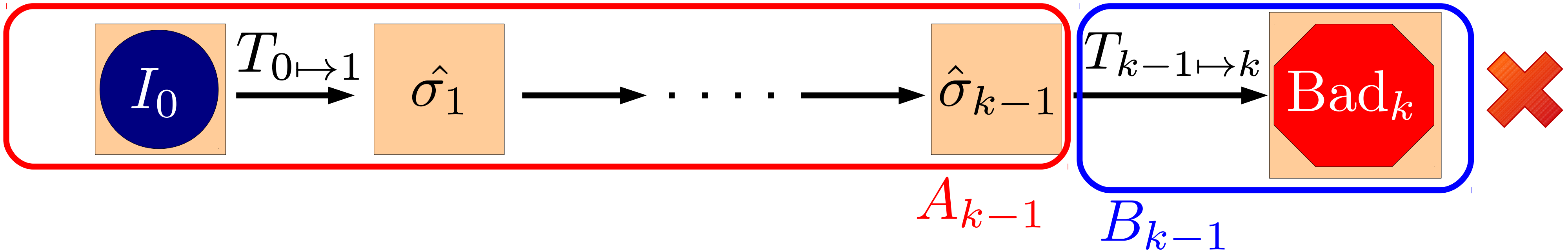


- Compute a sequence of interpolants $\varphi_0, \ldots, \varphi_{k-1}$

  such that $T_{i \mapsto i+1} \wedge \varphi_i \models \varphi_{i+1}$ for all $i \in [0, k-1)$

Interpolation-based Abstraction Refinement

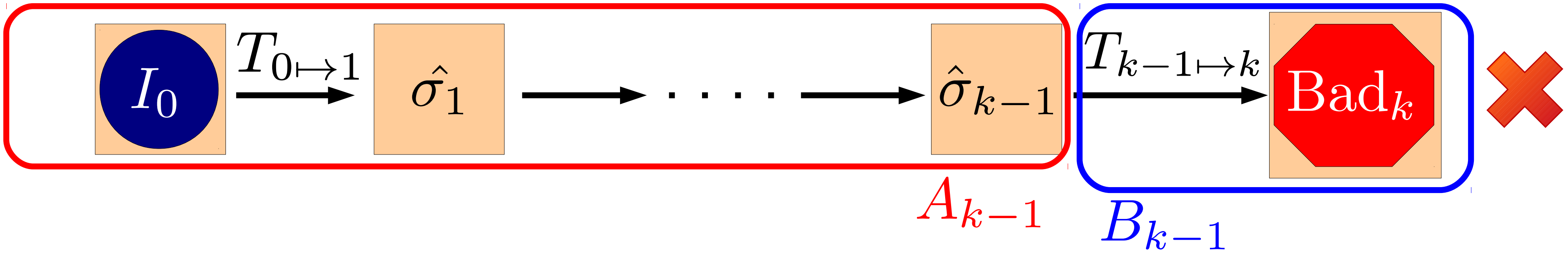
- An abstract cex path $\hat{\sigma}_0, \ldots, \hat{\sigma}_k$ (wrt. $\mathbb{P}$) might be spurious
  - Because abstraction is overapproximating



- Compute a sequence of interpolants $\varphi_0, \ldots, \varphi_{k-1}$ such that $T_{i \mapsto i+1} \wedge \varphi_i \models \varphi_{i+1}$ for all $i \in [0, k-1)$

Interpolation-based Abstraction Refinement

- An abstract cex path $\hat{\sigma}_0, \ldots, \hat{\sigma}_k$ (wrt. $\mathbb{P}$) might be spurious
  - Because abstraction is overapproximating



- Compute a sequence of interpolants $\varphi_0, \ldots, \varphi_{k-1}$
  such that $T_{i \mapsto i+1} \wedge \varphi_i \models \varphi_{i+1}$ for all $i \in [0, k-1)$
- Let $\mathbb{P}_{\text{new}}$ be the set of all the predicates in $\varphi_0, \ldots, \varphi_{k-1}$
- Set $\mathbb{P}' := \mathbb{P} \cup \mathbb{P}_{\text{new}}$
- **Theorem:** $\hat{\sigma}_0, \ldots, \hat{\sigma}_k$ is not an abstract cex path wrt. $\mathbb{P}'$

Proof sketch

- φ_i is an overapproximation of the states reachable in *i* steps, compatible with the abstract trace $\hat{\sigma}_0, \ldots, \hat{\sigma}_i$
- φ_i is also incompatible with the rest of the abstract trace $\hat{\sigma}_{i+1}, \ldots, \hat{\sigma}_k$ (since it is an interpolant)
- By the requirement that $T_{i \mapsto i+1} \wedge \varphi_i \models \varphi_{i+1}$

  it follows that $\mathrm{Img}(\varphi_i) \models \varphi_{i+1}$
- Therefore, $\mathrm{Img}(\underbrace{\ldots}_{k-2} \mathrm{Img}(\varphi_0)) \models \varphi_{k-1}$ and $\mathrm{Img}(\varphi_{k-1}) \models \bot$

  (since the trace is spurious)
- Since we add all the atomic predicates of $\varphi_0, \ldots, \varphi_{k-1}$ to $\mathbb{P}'$ and the abstraction is precise wrt. $\mathbb{P}'$, then

$$\widehat{\mathrm{Img}}(\underbrace{\ldots}_{k-1} \widehat{\mathrm{Img}}(\varphi_0)_{\mathbb{P}'})_{\mathbb{P}'} \models \bot$$

Outline

Introduction

Interpolants in Formal Verification

Computing interpolants in SMT

Efficient interpolation in SAT

- Interpolants for Boolean CNF formulae (A, B) can be computed from resolution refutations in linear time
- Traverse the resolution proof, annotating each node with a partial interpolant I
  - The partial interpolant for the root node (the empty clause) is the computed interpolant

Efficient interpolation in SAT

- Interpolants for Boolean CNF formulae (A, B) can be computed from resolution refutations in linear time
- Traverse the resolution proof, annotating each node with a partial interpolant I
  - The partial interpolant for the root node (the empty clause) is the computed interpolant
- McMillan's annotation rules (others exist):

Efficient interpolation in SAT

- Interpolants for Boolean CNF formulae *(A, B)* can be computed from resolution refutations in linear time
- Traverse the resolution proof, annotating each node with a partial interpolant *I*
  - The partial interpolant for the root node (the empty clause) is the computed interpolant
- McMillan's annotation rules (others exist):
  - For each leaf node (input clause) C in the proof:
    - If $C \in A$, set $I := \bigvee\{l \in C \mid \mathrm{var}(l) \in B\}$
    - Otherwise ($C \in B$), set $I := \top$

Efficient interpolation in SAT

- Interpolants for Boolean CNF formulae *(A, B)* can be computed from resolution refutations in linear time
- Traverse the resolution proof, annotating each node with a partial interpolant *I*
  - The partial interpolant for the root node (the empty clause) is the computed interpolant
- McMillan's annotation rules (others exist):
  - For each leaf node (input clause) C in the proof:
    - If $C \in A$, set $I := \bigvee\{l \in C \mid \mathrm{var}(l) \in B\}$
    - Otherwise ($C \in B$), set $I := \top$
  - For each inner node (resolution) with parents $\varphi \vee l$ and $\psi \vee \neg l$ and annotations I_1 and I_2
    - If $\mathrm{var}(l) \in B$, set $I := I_1 \wedge I_2$; otherwise, set $I := I_1 \vee I_2$

Example

$$A := (x \lor \neg y_1) \land (\neg x \lor \neg y_2) \land y_1$$

$$B := (\neg y_1 \lor y_2) \land (y_1 \lor z) \land \neg z$$

$$\dfrac{x \lor \neg y_1 \qquad \neg x \lor \neg y_2}{\neg y_1 \lor \neg y_2}$$

$$\dfrac{\neg y_1 \lor \neg y_2 \qquad y_1}{\neg y_2}$$

$$\dfrac{\neg y_2 \qquad \neg y_1 \lor y_2}{\neg y_1}$$

$$\dfrac{y_1 \lor z \qquad \neg y_1}{z}$$

$$\dfrac{z \qquad \neg z}{\bot}$$

Example

$$A := (x \lor \neg y_1) \land (\neg x \lor \neg y_2) \land y_1$$

$$B := (\neg y_1 \lor y_2) \land (y_1 \lor z) \land \neg z$$

$$\dfrac{x \lor \neg y_1 \;\; [\neg y_1] \qquad \neg x \lor \neg y_2 \;\; [\neg y_2]}{\neg y_1 \lor \neg y_2 \;\; [\neg y_1 \lor \neg y_2]}$$

$$\dfrac{\neg y_1 \lor \neg y_2 \;\; [\neg y_1 \lor \neg y_2] \qquad y_1 \;\; [y_1]}{\neg y_2 \;\; [(\neg y_1 \lor \neg y_2) \land y_1]}$$

$$\dfrac{\neg y_2 \;\; [(\neg y_1 \lor \neg y_2) \land y_1] \qquad \neg y_1 \lor y_2 \;\; [\top]}{\neg y_1 \;\; [(\neg y_1 \lor \neg y_2) \land y_1]}$$

$$\dfrac{y_1 \lor z \;\; [\top] \qquad \neg y_1 \;\; [(\neg y_1 \lor \neg y_2) \land y_1]}{z \;\; [(\neg y_1 \lor \neg y_2) \land y_1]}$$

$$\dfrac{z \;\; [(\neg y_1 \lor \neg y_2) \land y_1] \qquad \neg z \;\; [\top]}{\bot \;\; [(\neg y_1 \lor \neg y_2) \land y_1]}$$

Proof of correctness

- By induction on the structure of the resolution refutation

- Lemma: for each annotated node $C\ [I]$, we have
  1) $A \models I \vee \bigvee\{l \in C \mid \mathrm{var}(l) \notin B\}$
  2) $B \wedge I \models \vee \bigvee\{l \in C \mid \mathrm{var}(l) \in B\}$
  3) *I* contains only variables that occur in both *A* and *B*

- Then as a corollary, for the root $\perp\ [I]$, *I* is an interpolant

- The lemma trivially holds for leaf nodes (check)

Proof of correctness – resolution steps

Resolution step with parents $(\varphi \vee l)\ [I_1]$ and $(\psi \vee \neg l)\ [I_2]$

- Case $\mathrm{var}(l) \in B$

1) By ind. hyp $A \models I_1 \vee \bigvee\{p \in \varphi \mid \mathrm{var}(p) \notin B\}$ and
$A \models I_2 \vee \bigvee\{p \in \psi \mid \mathrm{var}(p) \notin B\}$

Therefore $A \models (I_1 \wedge I_2) \vee \bigvee\{p \in \varphi \wedge \psi \mid \mathrm{var}(p) \notin B\}$

2) By inductive hypotesis $B \wedge I_1 \models \bigvee\{p \in \varphi \vee l \mid \mathrm{var}(p) \in B\}$
which means $B \models \neg I_1 \vee \bigvee\{p \in \varphi \vee l \mid \mathrm{var}(p) \in B\}$
Similarly, $B \models \neg I_2 \vee \bigvee\{p \in \psi \vee \neg l \mid \mathrm{var}(p) \in B\}$
By resolution on $\mathrm{var}(l)$, then

$$B \models \neg I_1 \vee \neg I_2 \vee \bigvee\{p \in \varphi \vee \psi \mid \mathrm{var}(p) \in B\}$$

3) Trivial by the inductive hypothesis

Proof of correctness – resolution steps

Resolution step with parents $(\varphi \vee l)\ [I_1]$ and $(\psi \vee \neg l)\ [I_2]$

- Case $\mathrm{var}(l) \notin B$

1) By ind. hyp $A \models I_1 \vee \bigvee\{p \in \varphi \vee l \mid \mathrm{var}(p) \notin B\}$ and
$A \models I_2 \vee \bigvee\{p \in \psi \vee \neg l \mid \mathrm{var}(p) \notin B\}$

By resolution on $\mathrm{var}(l)$, then

$$A \models (I_1 \vee I_2) \vee \bigvee\{p \in \varphi \vee \psi \mid \mathrm{var}(p) \notin B\}$$

2) By ind. hyp $B \models \neg I_1 \vee \bigvee\{p \in \varphi \mid \mathrm{var}(p) \in B\}$ and
$B \models \neg I_2 \vee \bigvee\{p \in \psi \mid \mathrm{var}(p) \in B\}$

Therefore $B \models \neg I_1 \vee \bigvee\{p \in \varphi \vee \psi \mid \mathrm{var}(p) \in B\}$ and
$B \models \neg I_2 \vee \bigvee\{p \in \varphi \vee \psi \mid \mathrm{var}(p) \in B\}$

and so $B \wedge (I_1 \vee I_2) \models \bigvee\{p \in \varphi \vee \psi \mid \mathrm{var}(p) \in B\}$

3) Trivial by the inductive hypothesis

Interpolants in SMT

- Resolution refutations in SMT:

Boolean part (ground resolution) **+** *T*-specific part for conjunctions of constraints (negated *T*-lemmas)

Interpolants in SMT

- Resolution refutations in SMT:

Boolean part (ground resolution) + *T*-specific part for conjunctions of constraints (negated *T*-lemmas)

Standard Boolean interpolation

T-specific interpolation for conjunctions only

Theory interpolation only for sets of *T*-literals

Interpolants in SMT

- Resolution refutations in SMT:

Boolean part (ground resolution) **+** *T*-specific part for conjunctions of constraints (negated *T*-lemmas)

Standard Boolean interpolation

T-specific interpolation for conjunctions only

Theory interpolation only for sets of *T*-literals

- Annotation for a *T*-lemma *C*:

$$I := T\text{-interpolant}\left(\bigwedge\{l \in \neg C \mid \text{var}(l) \notin B\},\ \bigwedge\{l \in \neg C \mid \text{var}(l) \in B\}\right)$$

Equality (EUF)

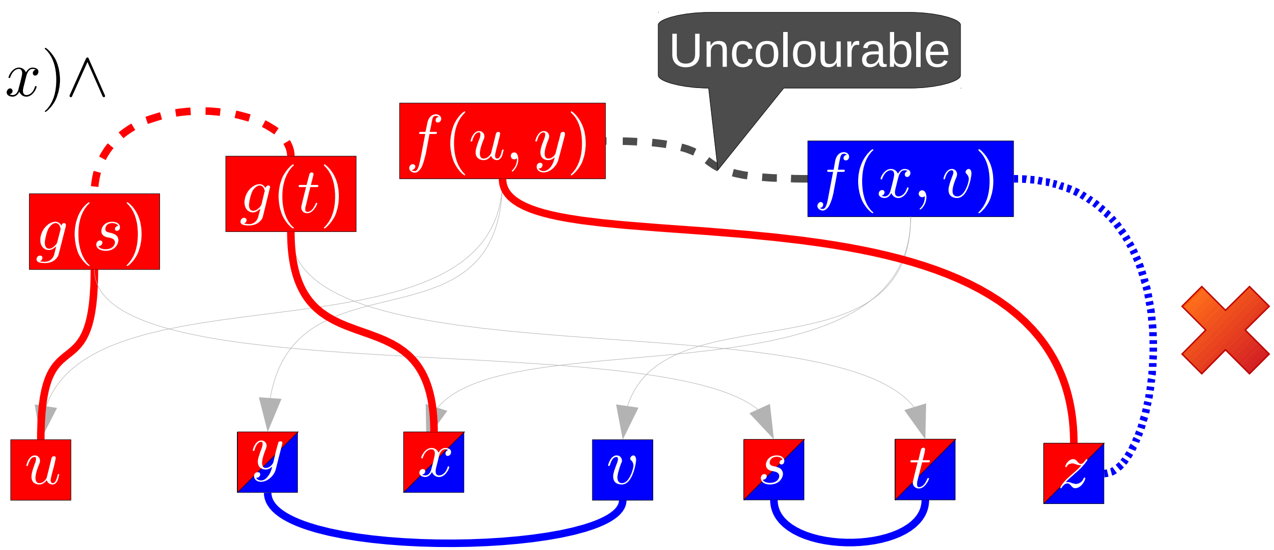
- Interpolants from coloured congruence graphs
  - Nodes with colours: [red] if term occurs in *A*; [blue] if term occurs in *B*; [red/blue] if term is shared
  - Edges with colours of the nodes they connect
    - Uncolorable edge: connects nodes of two different colours
  - Always possible to obtain a coloured graph
    - (by introducing new nodes)

Equality (EUF)

- Interpolants from coloured congruence graphs
  - Nodes with colours: ■ if term occurs in *A* ■ if term occurs in *B* ■ if term is shared
  - Edges with colours of the nodes they connect
    - Uncolorable edge: connects nodes of two different colours
  - Always possible to obtain a coloured graph
    - (by introducing new nodes)

$$A := (u = g(s)) \wedge (g(t) = x) \wedge (f(u, y) = z)$$

$$B := (v = y) \wedge (s = t) \wedge \neg(f(x, v) = z)$$

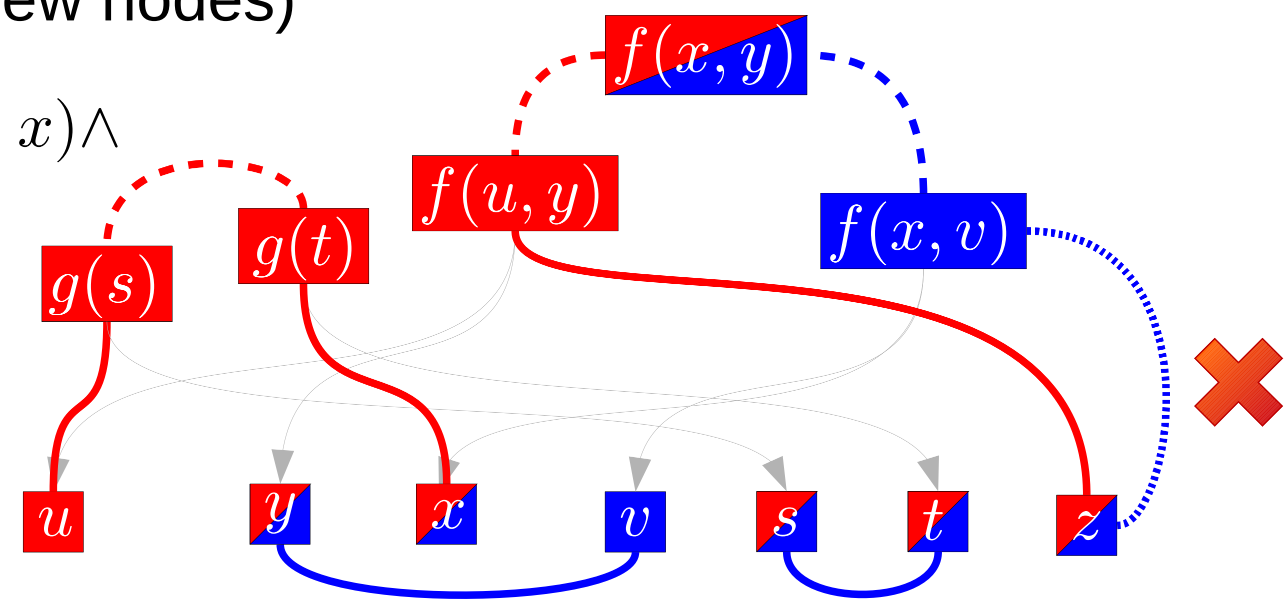


Equality (EUF)

- Interpolants from coloured congruence graphs
  - Nodes with colours:
    - (red) if term occurs in *A*
    - (blue) if term occurs in *B*
    - (red/blue) if term is shared
  - Edges with colours of the nodes they connect
    - Uncolorable edge: connects nodes of two different colours
  - Always possible to obtain a coloured graph
    - (by introducing new nodes)

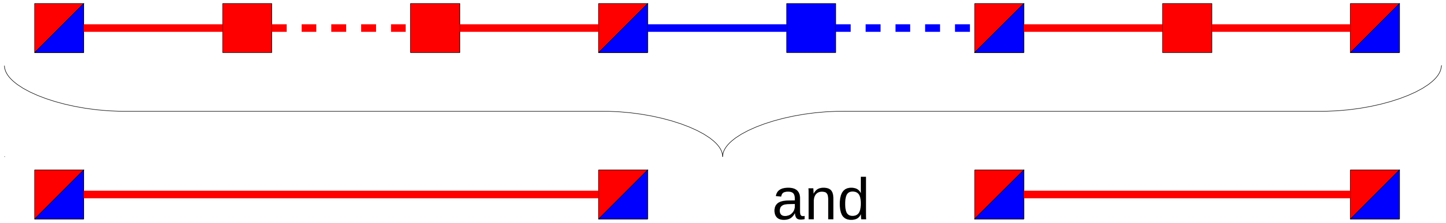
$$A := (u = g(s)) \wedge (g(t) = x) \wedge (f(u, y) = z)$$

$$B := (v = y) \wedge (s = t) \wedge \neg(f(x, v) = z)$$



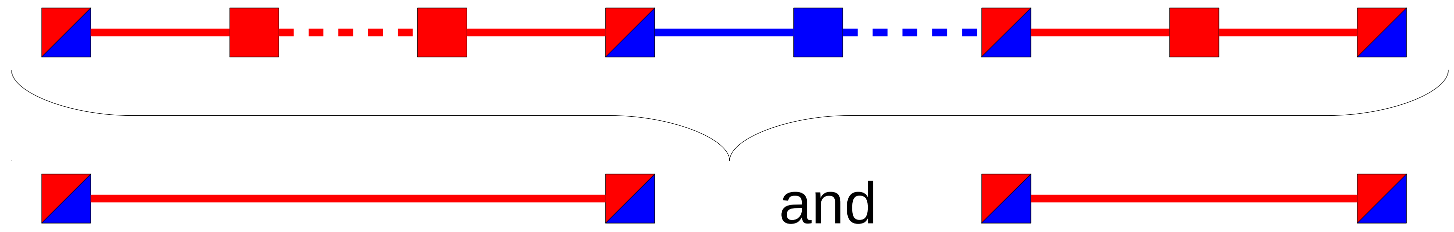
Interpolation algorithm (sketch)

- Start from disequality edge
- Compute summaries for *A*-paths with shared endpoints

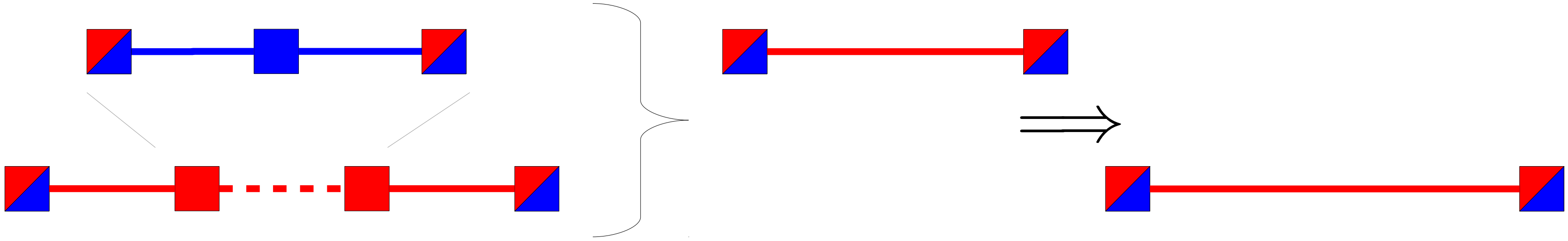


Interpolation algorithm (sketch)

- Start from disequality edge ....
- Compute summaries for *A*-paths with shared endpoints

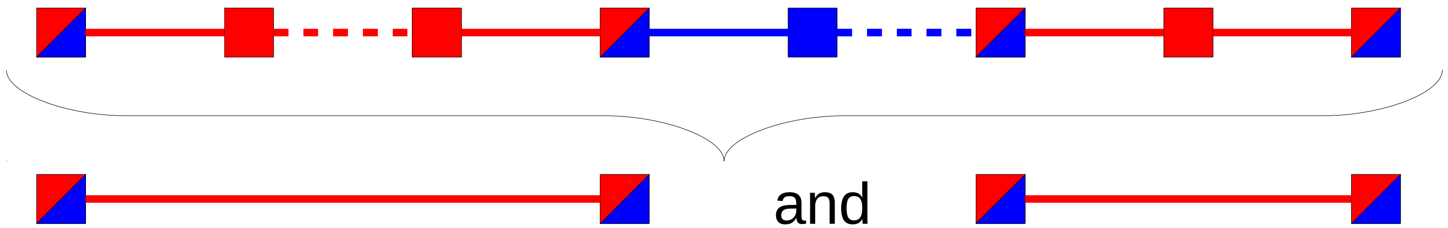


- If an *A*-summary involves a congruence edge, compute summaries recursively on function arguments
  - Use *B*-summaries as premises for the *A*-summary

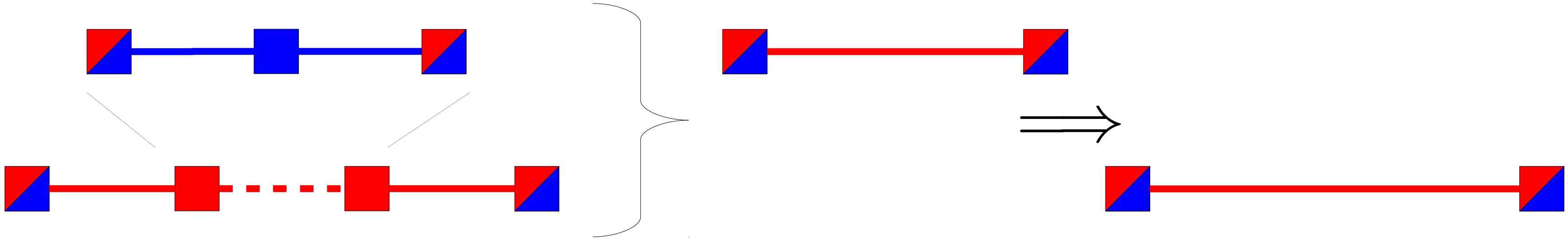


Interpolation algorithm (sketch)

- Start from disequality edge
- Compute summaries for *A*-paths with shared endpoints



- If an *A*-summary involves a congruence edge, compute summaries recursively on function arguments
  - Use *B*-summaries as premises for the *A*-summary

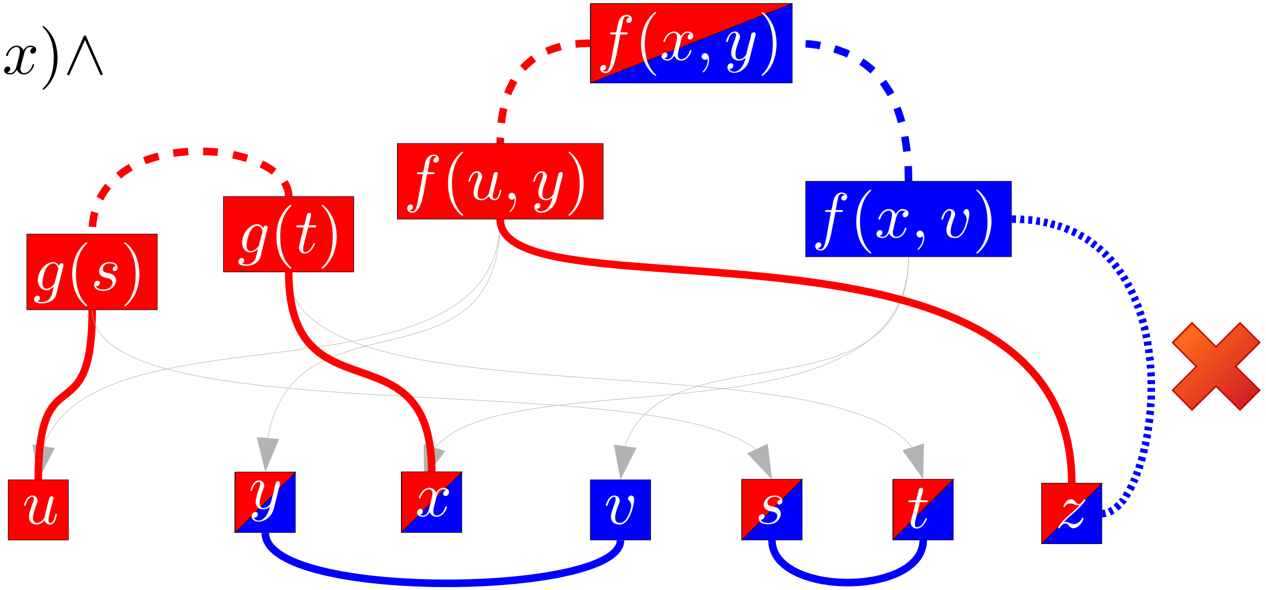


- (Several cases to consider)

Example

$A := (u = g(s)) \wedge (g(t) = x) \wedge$
$\quad (f(u, y) = z)$

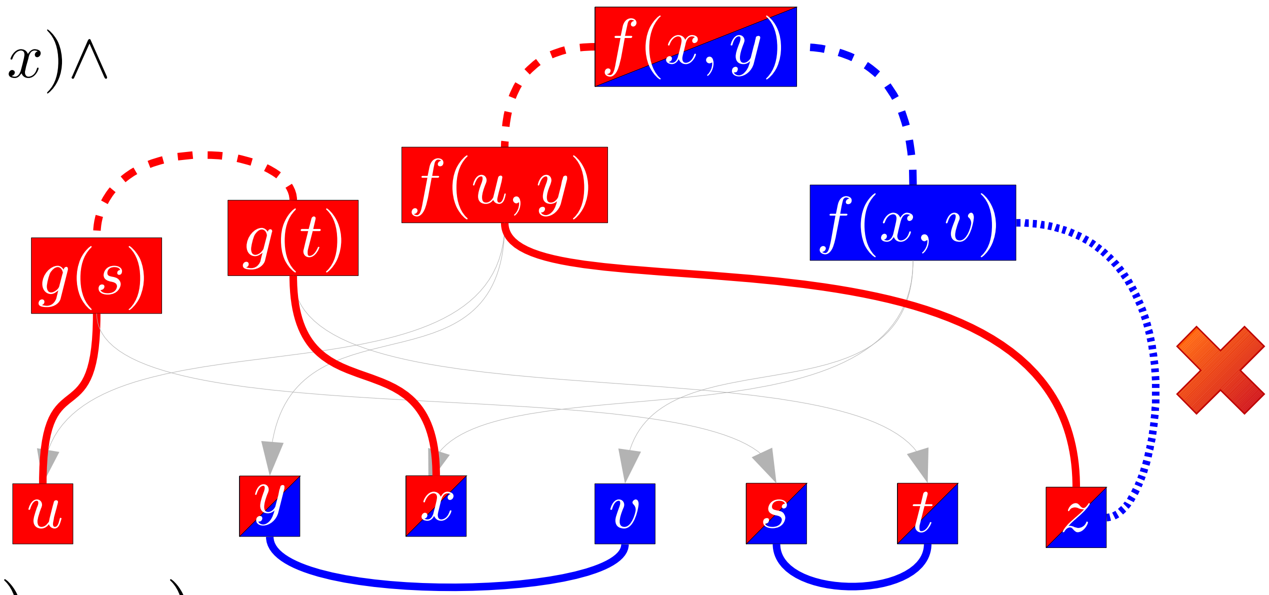
$B := (v = y) \wedge (s = t) \wedge$
$\quad \neg(f(x, v) = z)$

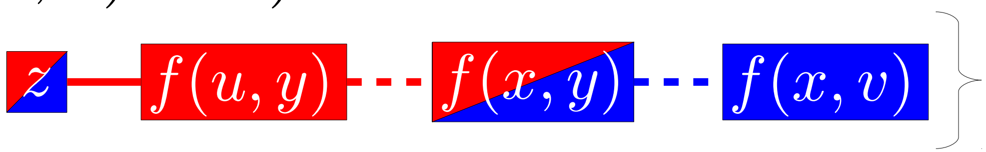


Example

$A := (u = g(s)) \wedge (g(t) = x) \wedge$
$\qquad (f(u, y) = z)$

$B := (v = y) \wedge (s = t) \wedge$
$\qquad \neg(f(x, v) = z)$



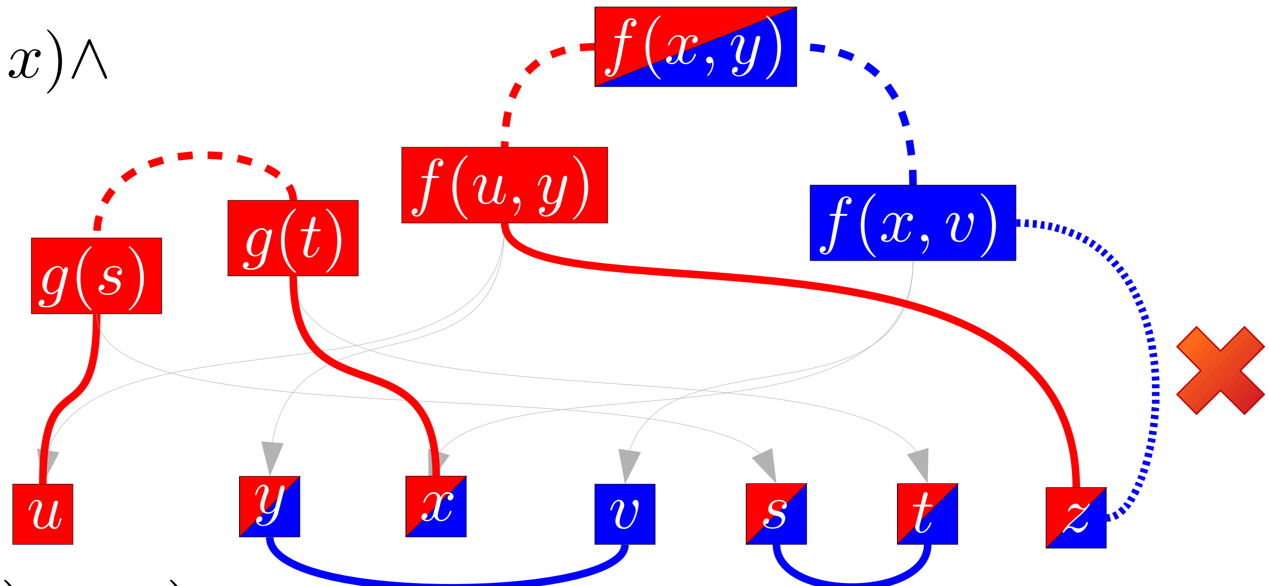
- Start from $\neg(f(x, v) = z)$
- *A*-summaries for 

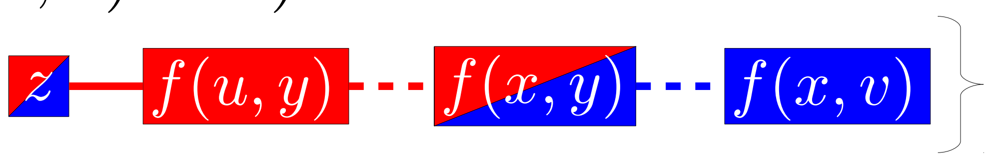
$z = f(x, y)$

Example

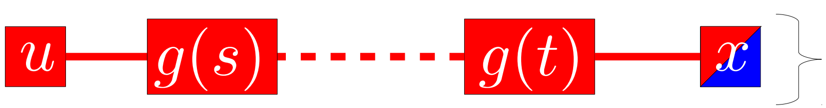
$A := (u = g(s)) \wedge (g(t) = x) \wedge$
$(f(u, y) = z)$

$B := (v = y) \wedge (s = t) \wedge$
$\neg(f(x, v) = z)$



- Start from $\neg(f(x, v) = z)$
- *A*-summaries for 

$z = f(x, y)$
- Recurse on edge 

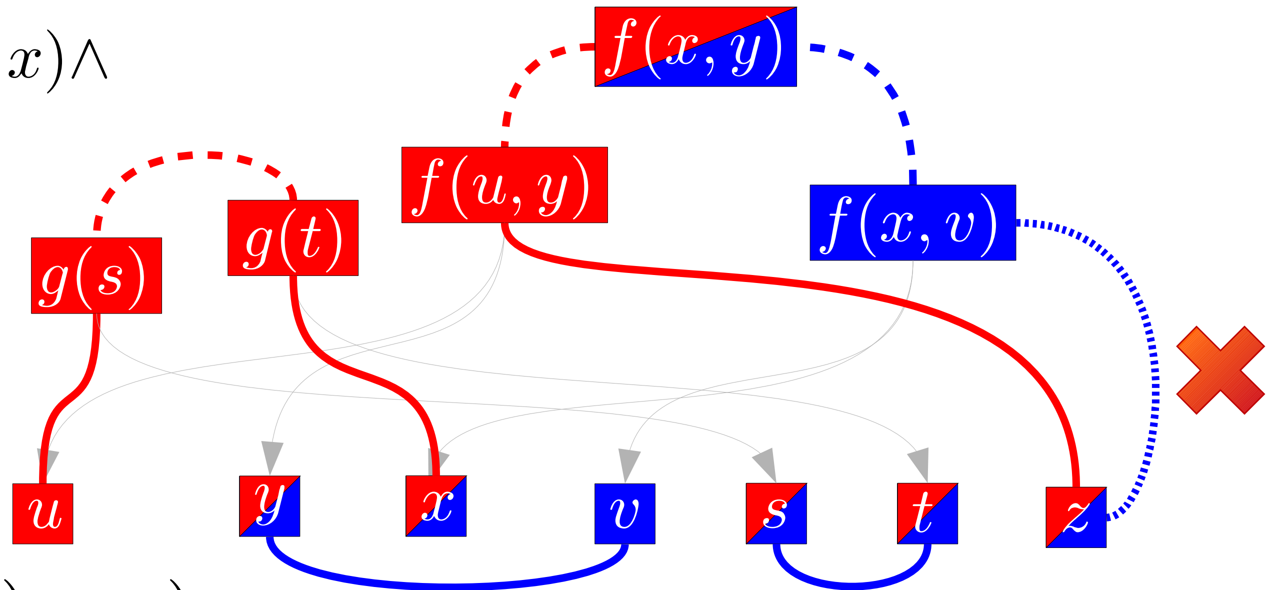
  - Path 

$\top$

Example

$A := (u = g(s)) \wedge (g(t) = x) \wedge$
$(f(u, y) = z)$

$B := (v = y) \wedge (s = t) \wedge$
$\neg(f(x, v) = z)$

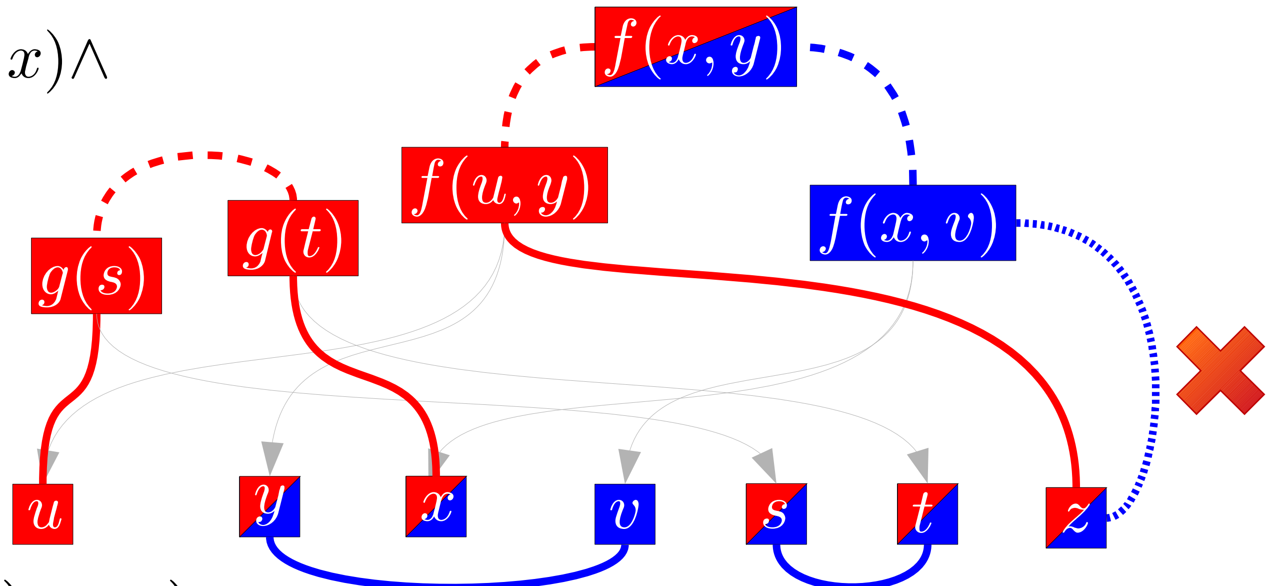


- Start from $\neg(f(x, v) = z)$
- *A*-summaries for z — $f(u, y)$ - - - $f(x, y)$ - - - $f(x, v)$ } $z = f(x, y)$
- Recurse on edge $f(u, y)$ - - - $f(x, y)$
  - Path u — $g(s)$ - - - $g(t)$ — x } $\top$
    - Recurse on edge $g(s)$ - - - $g(t)$
      - Path s — t , *B*-summary: $(s = t)$

Example

$A := (u = g(s)) \wedge (g(t) = x) \wedge$
$(f(u,y) = z)$

$B := (v = y) \wedge (s = t) \wedge$
$\neg(f(x,v) = z)$



- Start from $\neg(f(x,v) = z)$
- *A*-summaries for z — $f(u,y)$ - - - $f(x,y)$ - - - $f(x,v)$ } $z = f(x,y)$
- Recurse on edge $f(u,y)$ - - - $f(x,y)$
  - Path u — $g(s)$ - - - $g(t)$ — x } $\top$
  - Recurse on edge $g(s)$ - - - $g(t)$
    - Path s — t , *B*-summary: $(s = t)$
- Interpolant: $(s = t) \implies (z = f(x,y))$

Linear Rational Arithmetic (LRA)

- Interpolants from proofs of unsatisfiability of a system of inequalities $\sum_i a_i x_i \leq c$
- Proof of unsatisfiability: linear combination of inequalities with positive coefficients to derive a contradiction ($0 \leq c$ with $c < 0$)
- Interpolant obtained out of the proof by combining inequalities from *A* (using the same coefficients)
- Proof of unsatisfiability generated from the Simplex

Example

$$A := \underbrace{(3x_2 - x_1 \le 1)}_{s_1}, \underbrace{(0 \le x_1 + x_2)}_{s_2} \qquad B := \underbrace{(3 \le x_3 - 2x_1)}_{s_3}, \underbrace{(2x_3 \le 1)}_{s_4}$$

tableau

$$\begin{array}{rcl} s_1 & = & 3x_2 - x_1 \\ s_2 & = & x_1 + x_2 \\ s_3 & = & x_3 - 2x_1 \\ s_4 & = & 2x_3 \end{array}$$

bounds

$$\begin{array}{rcccl} -\infty & \le & s_1 & \le & 1 \\ 0 & \le & s_2 & \le & \infty \\ 3 & \le & s_3 & \le & \infty \\ -\infty & \le & s_4 & \le & 1 \end{array}$$

candidate solution β

$$\begin{array}{rcl} x_1 & \mapsto & 0 \\ x_2 & \mapsto & 0 \\ x_3 & \mapsto & 0 \\ s_1 & \mapsto & 0 \\ s_2 & \mapsto & 0 \\ s_3 & \mapsto & 0 \\ s_4 & \mapsto & 0 \end{array}$$

Example

$$\color{red}{A} := \underbrace{(3x_2 - x_1 \le 1)}_{s_1}, \underbrace{(0 \le x_1 + x_2)}_{s_2} \qquad \color{blue}{B} := \underbrace{(3 \le x_3 - 2x_1)}_{s_3}, \underbrace{(2x_3 \le 1)}_{s_4}$$

tableau

$$\begin{aligned} x_3 &= -\tfrac{1}{2}s_1 + \tfrac{3}{2}s_2 + s_3 \\ x_2 &= \tfrac{1}{4}s_1 + \tfrac{1}{4}s_2 \\ x_1 &= -\tfrac{1}{4}s_1 + \tfrac{3}{4}s_2 \\ s_4 &= -s_1 + 3s_2 + 2s_3 \end{aligned}$$

bounds

$$\begin{array}{ccccc} -\infty & \le & s_1 & \le & 1 \\ 0 & \le & s_2 & \le & \infty \\ 3 & \le & s_3 & \le & \infty \\ -\infty & \le & s_4 & \le & 1 \end{array}$$

candidate solution β

$$\begin{array}{rcr} x_1 & \mapsto & -\frac{1}{4} \\ x_2 & \mapsto & \frac{1}{4} \\ x_3 & \mapsto & \frac{5}{2} \\ s_1 & \mapsto & 1 \\ s_2 & \mapsto & 0 \\ s_3 & \mapsto & 3 \\ s_4 & \mapsto & 5 \end{array}$$

No suitable variable for pivoting!
Conflict

Example

$$A := \underbrace{(3x_2 - x_1 \leq 1)}_{s_1}, \underbrace{(0 \leq x_1 + x_2)}_{s_2} \qquad B := \underbrace{(3 \leq x_3 - 2x_1)}_{s_3}, \underbrace{(2x_3 \leq 1)}_{s_4}$$

tableau

$$\begin{aligned} x_3 &= -\tfrac{1}{2}s_1 + \tfrac{3}{2}s_2 + s_3 \\ x_2 &= \tfrac{1}{4}s_1 + \tfrac{1}{4}s_2 \\ x_1 &= -\tfrac{1}{4}s_1 + \tfrac{3}{4}s_2 \\ s_4 &= -s_1 + 3s_2 + 2s_3 \end{aligned}$$

bounds

$$\begin{array}{ccccc} -\infty & \leq & s_1 & \leq & 1 \\ 0 & \leq & s_2 & \leq & \infty \\ 3 & \leq & s_3 & \leq & \infty \\ -\infty & \leq & s_4 & \leq & 1 \end{array}$$

candidate solution β

$$\begin{array}{ccc} x_1 & \mapsto & -\frac{1}{4} \\ x_2 & \mapsto & \frac{1}{4} \\ x_3 & \mapsto & \frac{5}{2} \\ s_1 & \mapsto & 1 \\ s_2 & \mapsto & 0 \\ s_3 & \mapsto & 3 \\ s_4 & \mapsto & 5 \end{array}$$

Proof:

$$\dfrac{\dfrac{\dfrac{1 \cdot (2x_3 \leq 1) \quad 1 \cdot (3x_2 - x_1 \leq 1)}{(2x_3 + 3x_2 - x_1 \leq 2)} \quad 3 \cdot (0 \leq x_1 + x_2)}{(2x_3 - 4x_1 \leq 2)} \quad 2 \cdot (3 \leq x_3 - 2x_1)}{(0 \leq -4)}$$

Example

$A := \underbrace{(3x_2 - x_1 \leq 1)}_{s_1}, \underbrace{(0 \leq x_1 + x_2)}_{s_2}$ $\quad B := \underbrace{(3 \leq x_3 - 2x_1)}_{s_3}, \underbrace{(2x_3 \leq 1)}_{s_4}$

tableau

$$\begin{aligned} x_3 &= -\tfrac{1}{2}s_1 + \tfrac{3}{2}s_2 + s_3 \\ x_2 &= \tfrac{1}{4}s_1 + \tfrac{1}{4}s_2 \\ x_1 &= -\tfrac{1}{4}s_1 + \tfrac{3}{4}s_2 \\ \boxed{s_4} &\boxed{= -s_1 + 3s_2 + 2s_3} \end{aligned}$$

bounds

$$\begin{array}{ccccc} -\infty & \leq & \boxed{s_1 \leq 1} \\ \boxed{0 \leq s_2} & \leq & \infty \\ \boxed{3 \leq s_3} & \leq & \infty \\ -\infty & \leq & \boxed{s_4 \leq 1} \end{array}$$

candidate solution β

$$\begin{aligned} x_1 &\mapsto -\tfrac{1}{4} \\ x_2 &\mapsto \tfrac{1}{4} \\ x_3 &\mapsto \tfrac{5}{2} \\ s_1 &\mapsto 1 \\ s_2 &\mapsto 0 \\ s_3 &\mapsto 3 \\ s_4 &\mapsto 5 \end{aligned}$$

Interpolant:

$$\dfrac{\dfrac{\dfrac{-\quad 1\cdot(3x_2 - x_1 \leq 1)}{(3x_2 - x_1 \leq 1)} \quad 3\cdot(0 \leq x_1 + x_2)}{(-4x_1 \leq 1)} \quad -}{\boxed{(-4x_1 \leq 1)}}$$

Linear Integer Arithmetic (LIA)

- Constraints of the form

$$\sum_i c_i x_i + c \bowtie 0, \qquad \bowtie \in \{\leq, =\}$$

- In general, no quantifier-free interpolation for LIA

Example: $A := (y - 2x = 0) \quad B := (y - 2z - 1 = 0)$

The only interpolant is: $\exists w.(y = 2w)$

- Solution: extend the signature to include modular equations (divisibility predicates)

$$(t + c =_d 0) \equiv \exists w.(t + c = d \cdot w), \quad d \in \mathbb{Z}^{>0}$$

The interpolant now becomes: $(y =_2 0)$

SMT(LIA) with modular equations

- Modular equations can be eliminated via preprocessing:
  - Replace every atom $a := (t + c =_d 0)$ with a fresh Boolean variable p_a
  - Add the 4 clauses

$$p_a \rightarrow (t + c - dw_1 = 0)$$

$$\neg p_a \rightarrow (t + c - dw_1 - w_2 = 0)$$

$$(-w_2 + 1 \leq 0)$$

$$(w_2 - d + 1 \leq 0)$$

where w_1, w_2 are fresh integer variables

Interpolants from LIA-proofs

- Cutting-plane proof system: complete proof system for LIA

$$\text{Hyp}\ \frac{-}{t \leq 0} \qquad\qquad \text{Comb}\ \frac{t_1 \leq 0 \qquad t_2 \leq 0}{c_1 \cdot t_1 + c_2 \cdot t_2 \leq 0}, c_1, c_2 > 0$$

$$\text{Div}\ \frac{\sum_i c_i x_i + c \leq 0}{\sum_i \frac{c_i}{d} x_i + \lceil \frac{c}{d} \rceil \leq 0}, d > 0 \text{ divides the } c_i\text{'s}$$

Interpolants from LIA-proofs

- Cutting-plane proof system: complete proof system for LIA

$$\text{Hyp}\ \frac{-}{t \leq 0} \qquad \text{Comb}\ \frac{t_1 \leq 0 \qquad t_2 \leq 0}{c_1 \cdot t_1 + c_2 \cdot t_2 \leq 0}, c_1, c_2 > 0$$

LRA rules

$$\text{Div}\ \frac{\sum_i c_i x_i + c \leq 0}{\sum_i \frac{c_i}{d} x_i + \lceil \frac{c}{d} \rceil \leq 0}, d > 0 \text{ divides the } c_i\text{'s}$$

Interpolants from LIA-proofs

- Cutting-plane proof system: complete proof system for LIA

$$\text{Hyp}\ \frac{-}{t \leq 0} \qquad \text{Comb}\ \frac{t_1 \leq 0 \qquad t_2 \leq 0}{c_1 \cdot t_1 + c_2 \cdot t_2 \leq 0}, c_1, c_2 > 0$$

$$\text{Strenghten}\ \frac{\sum_i c_i x_i + c \leq 0}{\sum_i c_i x_i + d \cdot \lceil \frac{c}{d} \rceil \leq 0}, d > 0 \text{ divides the } c_i\text{'s}$$

Interpolants from LIA-proofs

- Cutting-plane proof system: complete proof system for LIA

$$\text{Hyp}\ \frac{-}{t \leq 0} \qquad \text{Comb}\ \frac{t_1 \leq 0 \qquad t_2 \leq 0}{c_1 \cdot t_1 + c_2 \cdot t_2 \leq 0}, c_1, c_2 > 0$$

$$\text{Strenghten}\ \frac{\sum_i c_i x_i + c \leq 0}{\sum_i c_i x_i + d \cdot \lceil \frac{c}{d} \rceil \leq 0}, d > 0 \text{ divides the } c_i\text{'s}$$

- Interpolation by annotating proof rules
  - Annotation: a set of pairs $\{\langle t_i \leq 0, \bigwedge_j (t_{ij} = 0)\rangle\}_i$
  - When $\bot$ is derived, then

$$I := \bigvee_i (t_i \leq 0 \wedge \bigwedge_j \text{ExistElim}(x_i \not\in B).(t_{ij} = 0))$$

is the computed interpolant

Interpolants from cutting-plane proofs

- Annotations for Hyp and Comb from McMillan (same as LRA)

$$\text{Hyp}\ \frac{-}{t \leq 0\ [\{\langle t \leq 0, \top\rangle\}]} \quad t' = \begin{cases} t & \text{if } t \leq 0 \in A \\ 0 & \text{if } t \leq 0 \in B \end{cases}$$

$$\text{Comb}\ \frac{t_1 \leq 0\ [I_1] \qquad t_2 \leq 0\ [I_2]}{c_1 \cdot t_1 + c_2 \cdot t_2 \leq 0\ [I]}$$

$$I := \{\langle c_1 t'_i + c_2 t'_j \leq 0, E_i \wedge E_j\rangle \mid \langle t'_i, E_i\rangle \in I_1, \langle t'_j, E_j\rangle \in I_2\}$$

- k-Strengthen rule of [Brillout et al. IJCAR'10]

$$\text{Str.}\ \frac{\sum_i c_i x_i + c \leq 0\ [\{\langle t \leq 0, \top\rangle\}]}{\sum_i c_i x_i + d \cdot \lceil \frac{c}{d} \rceil \leq 0\ [I]}, d > 0 \text{ divides the } c_i\text{'s}$$

$$I := \{\langle (t + n \leq 0),\ (t + n = 0)\rangle \mid 0 \leq n < d \cdot \lceil \tfrac{c}{d} \rceil - c\} \cup \{\langle (t + d \cdot \lceil \tfrac{c}{d} \rceil - c \leq 0), \top\rangle\}$$

Interpolants from cutting-plane proofs

- Annotations for Hyp and Comb from McMillan
  (same as LRA)

$$\text{Hyp}\ \frac{-}{t \leq 0\ [\{\langle 0 \leq 0, \top\rangle\}]} \quad t' = \begin{cases} t & \text{if } t \leq 0 \in A \\ 0 & \text{if } t \leq 0 \in B \end{cases}$$

$$\text{Comb}\ \frac{t_1 \leq 0\ [I_1] \qquad t_2 \leq 0\ [I_2]}{c_1 \cdot t_1 + c_2 \cdot t_2 \leq 0\ [I]}$$

$$I := \{\langle c_1 t'_i + c_2 t'_j \leq 0, E_i \wedge E_j\rangle \mid \langle t'_i, E_i\rangle \in I_1, \langle t'_j, E_j\rangle \in I_2\}$$

- k-Strengthen rule of [Brillout et al. IJCAR'10]

$$\text{Str.}\ \frac{\sum_i c_i x_i + c \leq 0\ [\{\langle t \leq 0, \top\rangle\}]}{\sum_i c_i x_i + d \cdot \lceil \frac{c}{d} \rceil \leq 0\ [I]}, d > 0 \text{ divides the } c_i\text{'s}$$

$$I := \{\langle (t + n \leq 0),\ (t + n = 0)\rangle \mid 0 \leq n < d \cdot \lceil \tfrac{c}{d} \rceil - c\} \cup \{\langle (t + d \cdot \lceil \tfrac{c}{d} \rceil - c \leq 0), \top\rangle\}$$

Example

$$A := \begin{cases} -y - 4x - 1 \leq 0 \\ y + 4x \leq 0 \end{cases} \qquad B := \begin{cases} -y - 4z + 1 \leq 0 \\ y + 4z - 2 \leq 0 \end{cases}$$

$$\frac{y + 4x \leq 0 \qquad -y - 4z + 1 \leq 0}{4x - 4z + 1 \leq 0}$$

$$\frac{4x - 4z + 1 \leq 0}{4x - 4z + 1 + 3 \leq 0}$$

$$\frac{-y - 4x - 1 \leq 0 \qquad y + 4z - 2 \leq 0}{-4x + 4z - 3 \leq 0}$$

$$\frac{4x - 4z + 1 + 3 \leq 0 \qquad -4x + 4z - 3 \leq 0}{(1 \leq 0) \equiv \bot}$$

Example – with annotations

$$A := \begin{cases} -y - 4x - 1 \leq 0 \\ y + 4x \leq 0 \end{cases} \qquad B := \begin{cases} -y - 4z + 1 \leq 0 \\ y + 4z - 2 \leq 0 \end{cases}$$

$$\dfrac{\dfrac{\dfrac{\begin{matrix} y + 4x \leq 0 & -y - 4z + 1 \leq 0 \\ [\{\langle y + 4x \leq 0, \top\rangle\}] & [\{\langle 0 \leq 0, \top\rangle\}] \end{matrix}}{\begin{matrix} 4x - 4z + 1 \leq 0 \\ [\{\langle y + 4x \leq 0, \top\rangle\}] \end{matrix}}}{\begin{matrix} 4x - 4z + 1 + 3 \leq 0 \\ [\{\langle y + 4x + n \leq 0, y + 4x + n = 0\rangle \mid \\ 0 \leq n < 3\} \cup \{\langle y + 4x + 2 \leq 0, \top\rangle\}] \end{matrix}} \quad \dfrac{\begin{matrix} -y - 4x - 1 \leq 0 & y + 4z - 2 \leq 0 \\ [\{\langle -y - 4x - 1 \leq 0, \top\rangle\}] & [\{\langle 0 \leq 0, \top\rangle\}] \end{matrix}}{\begin{matrix} -4x + 4z - 3 \leq 0 \\ [\{\langle -y - 4x - 1 \leq 0, \top\rangle\}] \end{matrix}}}{\begin{matrix} (1 \leq 0) \equiv \bot \\ [\{\langle n - 1 \leq 0, y + 4x + n = 0\rangle \mid 0 \leq n < 3\} \cup \{\langle 2 - 1 \leq 0, \top\rangle\}] \end{matrix}}$$

Example – with annotations

$$A := \begin{cases} -y - 4x - 1 \leq 0 \\ y + 4x \leq 0 \end{cases} \qquad B := \begin{cases} -y - 4z + 1 \leq 0 \\ y + 4z - 2 \leq 0 \end{cases}$$

$$\frac{\dfrac{\begin{array}{cc} y + 4x \leq 0 & -y - 4z + 1 \leq 0 \\ [\{\langle y + 4x \leq 0, \top\rangle\}] & [\{\langle 0 \leq 0, \top\rangle\}] \end{array}}{\begin{array}{c} 4x - 4z + 1 \leq 0 \\ [\{\langle y + 4x \leq 0, \top\rangle\}] \end{array}}}{\begin{array}{c} 4x - 4z + 1 + 3 \leq 0 \\ [\{\langle y + 4x + n \leq 0, y + 4x + n = 0\rangle \mid 0 \leq n < 3\} \cup \{\langle y + 4x + 2 \leq 0, \top\rangle\}] \end{array}} \qquad \frac{\begin{array}{cc} -y - 4x - 1 \leq 0 & y + 4z - 2 \leq 0 \\ [\{\langle -y - 4x - 1 \leq 0, \top\rangle\}] & [\{\langle 0 \leq 0, \top\rangle\}] \end{array}}{\begin{array}{c} -4x + 4z - 3 \leq 0 \\ [\{\langle -y - 4x - 1 \leq 0, \top\rangle\}] \end{array}}$$

$$\overline{\begin{array}{c} (1 \leq 0) \equiv \bot \\ [\{\langle n - 1 \leq 0, y + 4x + n = 0\rangle \mid 0 \leq n < 3\} \cup \{\langle 2 - 1 \leq 0, \top\rangle\}] \end{array}}$$

Interpolant: $(y =_4 0) \vee (y + 1 =_4 0)$

Drawback of Strengthen

- Interpolation of Strengthen creates potentially very big disjunctions
  - Linear in the strengthening factor $k := d\lceil \frac{c}{d} \rceil - c$
  - Can be exponential in the size of the proof

Example:

$$A := \begin{cases} -y - 4x - 1 \leq 0 \\ y + 4x \leq 0 \end{cases} \qquad B := \begin{cases} -y - 4z + 1 \leq 0 \\ y + 4z - 2 \leq 0 \end{cases}$$

Interpolant: $(y =_4 0) \vee (y + 1 =_4 0)$

Drawback of Strengthen

- Interpolation of Strengthen creates potentially very big disjunctions
  - Linear in the strengthening factor $k := d\lceil \frac{c}{d} \rceil - c$
  - Can be exponential in the size of the proof

Example:
$$A := \begin{cases} -y - 2nx - n + 1 \leq 0 \\ y + 2nx \leq 0 \end{cases} \quad B := \begin{cases} -y - 2nz + 1 \leq 0 \\ y + 2nz - n \leq 0 \end{cases}$$

Interpolant: $(y =_{2n} 0) \vee (y + 1 =_{2n} 0) \vee \ldots \vee (y =_{2n} n - 1)$

Drawback of Strengthen

- Interpolation of Strengthen creates potentially very big disjunctions
  - Linear in the strengthening factor $k := d\lceil \frac{c}{d} \rceil - c$
  - Can be exponential in the size of the proof

Example: $A := \begin{cases} -y - 2nx - n + 1 \leq 0 \\ y + 2nx \leq 0 \end{cases}$ $B := \begin{cases} -y - 2nz + 1 \leq 0 \\ y + 2nz - n \leq 0 \end{cases}$

Interpolant: $(y =_{2n} 0) \vee (y + 1 =_{2n} 0) \vee \ldots \vee (y =_{2n} n - 1)$

- The problem are AB-mixed cuts:

$$\text{Strengthen}\ \frac{\sum_{x_i \notin B} c_i x_i + \sum_{y_j \notin A} c_j y_j + c \leq 0}{\sum_{x_i \notin B} c_i x_i + \sum_{y_j \notin A} c_j y_j + d \cdot \lceil \frac{c}{d} \rceil \leq 0}$$

Interpolation with ceilings

- Idea: use a different extension of the signature of LIA, and extend also its domain
  - Introduce the ceiling function $\lceil \cdot \rceil$ [Pudlák '97]
  - Allow non-variable terms to be non-integers (e.g. $\frac{x}{2}$)
- Much simpler interpolation procedure
  - Proof annotations are single inequalities $(t \leq 0)$

Interpolation with ceilings

- Idea: use a different extension of the signature of LIA, and extend also its domain
  - Introduce the ceiling function $\lceil \cdot \rceil$ [Pudlák '97]
  - Allow non-variable terms to be non-integers (e.g. $\frac{x}{2}$)
- Much simpler interpolation procedure
  - Proof annotations are single inequalities $(t \leq 0)$

$$\text{Hyp}\ \frac{-}{t \leq 0\ [t' \leq 0]} \qquad \text{Comb}\ \frac{t_1 \leq 0\ [t_1' \leq 0] \qquad t_2 \leq 0\ [t_2' \leq 0]}{c_1 \cdot t_1 + c_2 \cdot t_2 \leq 0\ [c_1 \cdot t_1' + c_2 \cdot t_2' \leq 0]}$$

$$\text{Div}\ \frac{\begin{array}{c}\sum_{y_j \notin B} a_j y_j + \sum_{z_k \notin A} b_k z_k + \sum_{x_i \in A \cap B} c_i x_i + c \\ \left[\sum_{y_j \notin B} a_j y_j + \sum_{x_i \in A \cap B} c_i' x_i + t'\right]\end{array}}{\begin{array}{c}\sum_{y_j \notin B} \frac{a_j}{d} y_j + \sum_{z_k \in B} \frac{b_k}{d} z_k + \sum_{x_i \in A \cap B} \frac{c_i}{d} x_i + \lceil \frac{c}{d} \rceil \\ \left[\sum_{y_j \notin B} \frac{a_j}{d} y_j + \left\lceil \frac{\sum_{x_i \in A \cap B} c_i' x_i + t'}{d} \right\rceil\right]\end{array}} \qquad d > 0 \text{ divides } a_j, b_k, c_i$$

Interpolation with ceilings - example

- No blowup of interpolants wrt. the size of the proofs

$$A := \begin{cases} -y - 2nx - n + 1 \leq 0 \\ y + 2nx \leq 0 \end{cases} \qquad B := \begin{cases} -y - 2nz + 1 \leq 0 \\ y + 2nz - n \leq 0 \end{cases}$$

$$\frac{y + 2nx \leq 0 \qquad -y - 2nz + 1 \leq 0}{2nx - 2nz + 1 \leq 0}$$

$$\frac{2nx - 2nz + 1 \leq 0}{2n \cdot (x - z + 1 \leq 0)}$$

$$\frac{-y - 2nx - n + 1 \leq 0 \qquad y + 2nz - n \leq 0}{-2nx + 2nz - 2n + 1 \leq 0}$$

$$\frac{2n \cdot (x - z + 1 \leq 0) \qquad -2nx + 2nz - 2n + 1 \leq 0}{(1 \leq 0) \equiv \bot}$$

Interpolation with ceilings - example

- No blowup of interpolants wrt. the size of the proofs

$$A := \begin{cases} -y - 2nx - n + 1 \leq 0 \\ y + 2nx \leq 0 \end{cases} \qquad B := \begin{cases} -y - 2nz + 1 \leq 0 \\ y + 2nz - n \leq 0 \end{cases}$$

$$\frac{\dfrac{\dfrac{y + 2nx \leq 0 \quad [y + 2nx \leq 0] \qquad -y - 2nz + 1 \leq 0 \quad [0 \leq 0]}{2nx - 2nz + 1 \leq 0 \quad [y + 2nx \leq 0]}}{2n \cdot (x - z + 1 \leq 0) \quad [x + \lceil \frac{y}{2n} \rceil \leq 0]} \qquad \dfrac{-y - 2nx - n + 1 \leq 0 \quad [-y - 2nx - n + 1 \leq 0] \qquad y + 2nz - n \leq 0 \quad [0 \leq 0]}{-2nx + 2nz - 2n + 1 \leq 0 \quad [-y - 2nx - n + 1 \leq 0]}}{(1 \leq 0) \equiv \bot \quad [2n\lceil \frac{y}{2n} \rceil - y - n + 1 \leq 0]}$$

Interpolation with ceilings - example

- No blowup of interpolants wrt. the size of the proofs

$$A := \begin{cases} -y - 2nx - n + 1 \leq 0 \\ y + 2nx \leq 0 \end{cases} \qquad B := \begin{cases} -y - 2nz + 1 \leq 0 \\ y + 2nz - n \leq 0 \end{cases}$$

$$\dfrac{\dfrac{\dfrac{\begin{matrix} y + 2nx \leq 0 & -y - 2nz + 1 \leq 0 \\ [y + 2nx \leq 0] & [0 \leq 0] \end{matrix}}{\begin{matrix} 2nx - 2nz + 1 \leq 0 \\ [y + 2nx \leq 0] \end{matrix}}}{\begin{matrix} 2n \cdot (x - z + 1 \leq 0) \\ [x + \lceil \frac{y}{2n} \rceil \leq 0] \end{matrix}} \qquad \dfrac{\begin{matrix} -y - 2nx - n + 1 \leq 0 & y + 2nz - n \leq 0 \\ [-y - 2nx - n + 1 \leq 0] & [0 \leq 0] \end{matrix}}{\begin{matrix} -2nx + 2nz - 2n + 1 \leq 0 \\ [-y - 2nx - n + 1 \leq 0] \end{matrix}}}{(1 \leq 0) \equiv \bot}$$

Interpolant: $[2n\lceil \frac{y}{2n} \rceil - y - n + 1 \leq 0]$

SMT(LIA) with ceilings

- Like modular equations, also ceilings can be eliminated via preprocessing
  - Replace every term $\lceil t \rceil$ with a fresh integer variable $x_{\lceil t \rceil}$
  - Add the 2 unit clauses (encoding the meaning of ceiling: $\lceil t \rceil - 1 < t \leq \lceil t \rceil$)

$$(l \cdot x_{\lceil t \rceil} - l \cdot t + l \leq 0)$$

$$(l \cdot t - l \cdot x_{\lceil t \rceil} \leq 0)$$

where l is the least common multiple of the denominators of the coefficients in t

Bit-vectors (BV)

- Interpolation for bit-vectors is hard
  - Only some limited work done so far
- Most efficient solvers use eager encoding into SAT, which is efficient but not good for interpolation
  - Easy in principle, but not very useful interpolants
- Try to exploit lazy bit-blasting to incorporate BV into DPLL(T)

Interpolation via Bit-Blasting

- Interpolation via bit-blasting is easy…
  - From A_{BV} and B_{BV} generate $A_{\texttt{Bool}}$ and $B_{\texttt{Bool}}$

    Each var x of width *n* encoded with *n* Boolean vars $b_1^x \ldots b_n^x$
  - Generate a Boolean interpolant $I_{\texttt{Bool}}$ for $(A_{\texttt{Bool}}, B_{\texttt{Bool}})$
  - Replace every variable b_i^x in $I_{\texttt{Bool}}$ with the bit-selection $x[i]$ and every Boolean connective with the corresponding bit-wise connective: $\wedge \mapsto \&, \quad \vee \mapsto |, \quad \neg \mapsto \sim$

- ...but quite impractical
  - Generates “ugly” interpolants
  - Word-level structure of the original problem completely lost
    - How to apply word-level simplifications?

Interpolation via Bit-Blasting - Example

$$A \stackrel{\text{def}}{=} (a_{[8]} * b_{[8]} = 15_{[8]}) \wedge (a_{[8]} = 3_{[8]})$$

$$B \stackrel{\text{def}}{=} \neg(b_{[8]} \%_u c_{[8]} = 1_{[8]}) \wedge (c_{[8]} = 2_{[8]})$$

A word-level interpolant is:

$$I \stackrel{\text{def}}{=} (b_{[8]} * 3_{[8]} = 15_{[8]})$$

...but with bit-blasting we get:

$$I' \stackrel{\text{def}}{=} (b_{[8]}[0] = 1_{[1]}) \wedge ((b_{[8]}[0] \& \sim ((((((\sim b_{[8]}[7] \& \sim b_{[8]}[6]) \& \sim b_{[8]}[5]) \& \sim b_{[8]}[4]) \& \sim b_{[8]}[3]) \& b_{[8]}[2]) \& \sim b_{[8]}[1])) = 0_{[1]})$$

Alternative: lazy bit-blasting and DPLL(T)

- Exploit *lazy bit-blasting*
  - Bit-blast only BV-atoms, not the whole formula
  - Boolean skeleton of the formula handled by the “main” DPLL, like in DPLL(T)
  - Conjunctions of BV-atoms handled (via bit-blasting) by a “sub”-DPLL (DPLL-BV) that acts as a BV-solver

Standard Boolean Interpolation	+	BV-specific Interpolation for *conjunctions of constraints*

Interpolation for BV constraints

- A layered approach
- Apply in sequence a chain of procedures of increasing generality and cost
  - Interpolation in EUF
  - Interpolation via equality inlining
  - Interpolation via Linear Integer Arithmetic encoding
  - Interpolation via bit-blasting

Interpolation in EUF

- Treat all the BV-operators as uninterpreted functions
- Exploit cheap, efficient algorithms for solving and interpolating modulo EUF
  - Possible because we avoid bit-blasting upront!

Example:

$$A \stackrel{\text{def}}{=} (x_{1[32]} = 3_{[32]}) \wedge (x_{3[32]} = x_{1[32]} \cdot x_{2[32]})$$

$$B \stackrel{\text{def}}{=} (x_{4[32]} = x_{2[32]}) \wedge (x_{5[32]} = 3_{[32]} \cdot x_{4[32]}) \wedge \neg(x_{3[32]} = x_{5[32]})$$

$$I_{\mathsf{UF}} \stackrel{\text{def}}{=} x_3 = f^{\cdot}(f^3, x_2)$$

$$I_{\mathsf{BV}} \stackrel{\text{def}}{=} x_{3[32]} = 3_{[32]} \cdot x_{2[32]}$$

Interpolation via Equality Inlining

- Interpolation via quantifier elimination: given (A, B), an interpolant can be computed by eliminating quantifiers from $\exists_{x \notin B} A$ or from $\exists_{x \notin A} \neg B$

- In general, this can be very expensive for BV
  - Might require bit-blasting and can cause blow-up of the formula

- Cheap case: non-common variables occurring in “definitional” equalities

  Example: $(x = e) \wedge \varphi$ and x does not occur in e, then

$$\exists_x((x = e) \wedge \varphi) \implies \varphi[x \mapsto e]$$

Interpolation via Equality Inlining

- Inline definitional equalities until either all all non-common variables are removed, or a fixpoint is reached
  - Try both from A and $\neg B$
  - If one of them succeeds, we have an interpolant

Example: $A \stackrel{\text{def}}{=} (0_{[24]} :: (x_{4[8]} \cdot x_{5[8]}) \leq_s (0_{[24]} :: x_{1[8]} - 1_{[32]})) \wedge$
$(x_{2[8]} = x_{1[8]}) \wedge (x_{4[8]} = 192_{[8]}) \wedge (x_{5[8]} = 128_{[8]})$

$B \stackrel{\text{def}}{=} ((x_{3[8]} \cdot x_{6[8]}) = (-(0_{[24]} :: x_{2[8]}))[7:0]) \wedge$
$(x_{3[8]} <_u 1_{[8]}) \wedge (0_{[8]} \leq_u x_{3[8]}) \wedge (x_{6[8]} = 1_{[8]})$

Interpolation via Equality Inlining

- Inline definitional equalities until either all all non-common variables are removed, or a fixpoint is reached
  - Try both from A and $\neg B$
  - If one of them succeeds, we have an interpolant

Example: $$A \stackrel{\text{def}}{=} (0_{[24]} :: (x_{4\,[8]} \cdot x_{5\,[8]}) \leq_s (0_{[24]} :: x_{1\,[8]} - 1_{[32]})) \wedge (x_{2\,[8]} = x_{1\,[8]}) \wedge (x_{4\,[8]} = 192_{[8]}) \wedge (x_{5\,[8]} = 128_{[8]})$$

Definitional equalities

$$B \stackrel{\text{def}}{=} ((x_{3\,[8]} \cdot x_{6\,[8]}) = (-(0_{[24]} :: x_{2\,[8]}))[7:0]) \wedge (x_{3\,[8]} <_u 1_{[8]}) \wedge (0_{[8]} \leq_u x_{3\,[8]}) \wedge (x_{6\,[8]} = 1_{[8]})$$

Interpolation via Equality Inlining

- Inline definitional equalities until either all all non-common variables are removed, or a fixpoint is reached
  - Try both from A and $\neg B$
  - If one of them succeeds, we have an interpolant

Example:

$$A \stackrel{\text{def}}{=} (0_{[24]} :: (x_{4\,[8]} \cdot x_{5\,[8]}) \leq_s (0_{[24]} :: x_{1\,[8]} - 1_{[32]})) \wedge$$
$$(x_{2\,[8]} = x_{1\,[8]}) \wedge (x_{4\,[8]} = 192_{[8]}) \wedge (x_{5\,[8]} = 128_{[8]})$$

$$B \stackrel{\text{def}}{=} ((x_{3\,[8]} \cdot x_{6\,[8]}) = (-(0_{[24]} :: x_{2\,[8]}))[7:0]) \wedge$$
$$(x_{3\,[8]} <_u 1_{[8]}) \wedge (0_{[8]} \leq_u x_{3\,[8]}) \wedge (x_{6\,[8]} = 1_{[8]})$$

Interpolation via Equality Inlining

- Inline definitional equalities until either all all non-common variables are removed, or a fixpoint is reached
  - Try both from A and $\neg B$
  - If one of them succeeds, we have an interpolant

Example: $$A \stackrel{\text{def}}{=} (0_{[24]} :: ({x_4}_{[8]} \cdot {x_5}_{[8]}) \leq_s (0_{[24]} :: {x_2}_{[8]} - 1_{[32]})) \wedge$$
$$\wedge ({x_4}_{[8]} = 192_{[8]}) \wedge ({x_5}_{[8]} = 128_{[8]})$$

$$B \stackrel{\text{def}}{=} (({x_3}_{[8]} \cdot {x_6}_{[8]}) = (-(0_{[24]} :: {x_2}_{[8]}))[7:0]) \wedge$$
$$({x_3}_{[8]} <_u 1_{[8]}) \wedge (0_{[8]} \leq_u {x_3}_{[8]}) \wedge ({x_6}_{[8]} = 1_{[8]})$$

Interpolation via Equality Inlining

- Inline definitional equalities until either all all non-common variables are removed, or a fixpoint is reached
  - Try both from A and $\neg B$
  - If one of them succeeds, we have an interpolant

Example: $A \stackrel{\text{def}}{=} (0_{[24]} :: (x_{4\,[8]} \cdot x_{5\,[8]}) \leq_s (0_{[24]} :: x_{2\,[8]} - 1_{[32]}))\wedge$

$\wedge (x_{4\,[8]} = 192_{[8]}) \wedge (x_{5\,[8]} = 128_{[8]})$

$$B \stackrel{\text{def}}{=} ((x_{3\,[8]} \cdot x_{6\,[8]}) = (-(0_{[24]} :: x_{2\,[8]}))[7:0])\wedge$$

$$(x_{3\,[8]} <_u 1_{[8]}) \wedge (0_{[8]} \leq_u x_{3\,[8]}) \wedge (x_{6\,[8]} = 1_{[8]})$$

Interpolation via Equality Inlining

- Inline definitional equalities until either all all non-common variables are removed, or a fixpoint is reached
  - Try both from A and $\neg B$
  - If one of them succeeds, we have an interpolant

Example: $A \stackrel{\text{def}}{=} (0_{[24]} :: (192_{[8]} \cdot 128_{[8]}) \leq_s (0_{[24]} :: x_{2[8]} - 1_{[32]}))$

$\wedge \qquad \wedge$

$B \stackrel{\text{def}}{=} ((x_{3[8]} \cdot x_{6[8]}) = (-(0_{[24]} :: x_{2[8]}))[7:0]) \wedge$
$(x_{3[8]} <_u 1_{[8]}) \wedge (0_{[8]} \leq_u x_{3[8]}) \wedge (x_{6[8]} = 1_{[8]})$

Interpolation via Equality Inlining

- Inline definitional equalities until either all all non-common variables are removed, or a fixpoint is reached
  - Try both from A and $\neg B$
  - If one of them succeeds, we have an interpolant

Example: $A \stackrel{\text{def}}{=} (0_{[24]} :: (192_{[8]} \cdot 128_{[8]}) \leq_s (0_{[24]} :: x_{2[8]} - 1_{[32]}))$

$$\wedge \qquad \wedge$$

$$I \stackrel{\text{def}}{=} (0_{32} \leq_s (0_{24} :: x_{2[8]} - 1_{[32]})$$

$$B \stackrel{\text{def}}{=} ((x_{3[8]} \cdot x_{6[8]}) = (-(0_{[24]} :: x_{2[8]}))[7:0]) \wedge$$
$$(x_{3[8]} <_u 1_{[8]}) \wedge (0_{[8]} \leq_u x_{3[8]}) \wedge (x_{6[8]} = 1_{[8]})$$

Interpolation via LIA Encoding

- Simple idea (in principle):
  - Encode a set of BV-constraints into an SMT(LIA)-formula
  - Generate a LIA-interpolant using existing algorithms
  - Map back to a BV-interpolant

- However, several problems to solve:
  - Efficiency
  - More importantly, soundness

Encoding BV into LIA

- Use well-known encodings from BV to SMT(LIA)
  - Encode each BV term $t_{[n]}$ as an integer variable x_t and the constraints $(0 \leq x_t) \wedge (x_t \leq 2^n - 1)$
  - Encode each BV operation as a LIA-formula.

Examples:

$$t_{[i-j+1]} \stackrel{\text{def}}{=} t_{1[n]}[i:j] \Rightarrow (x_t = m) \wedge (x_{t_1} = 2^{i+1}h + 2^j m + l) \wedge$$
$$l \in [0, 2^i) \wedge m \in [0, 2^{i-j+1}) \wedge h \in [0, 2^{n-i-1})$$

$$t_{[n]} \stackrel{\text{def}}{=} t_{1[n]} + t_{2[n]} \Rightarrow (x_t = x_{t_1} + x_{t_2} - 2^n \sigma) \wedge (0 \leq \sigma \leq 1)$$

$$t_{[n]} \stackrel{\text{def}}{=} t_{1[n]} \cdot k \Rightarrow (x_t = k \cdot x_{t_1} - 2^n \sigma) \wedge (0 \leq \sigma \leq k)$$

From LIA-interpolants to BV-interpolants

- “Invert” the LIA encoding to get a BV interpolant
- *Unsound* in general
  - Issues due to overflow and (un)signedness of operations
- Our (very simple) solution: *check the interpolants*
  - Given a candidate interpolant $\hat{I}$, use our SMT(BV) solver to check the unsatisfiability of $(A \wedge \neg\hat{I}) \vee (B \wedge \hat{I})$
  - If successful, then $\hat{I}$ is an interpolant

From LIA- to BV-interpolants: examples

$$A \stackrel{\text{def}}{=} ({y_1}_{[8]} = {y_5}_{[4]} :: {y_5}_{[4]}) \wedge ({y_1}_{[8]} = {y_2}_{[8]}) \wedge ({y_5}_{[4]} = 1_{[4]})$$

$$B \stackrel{\text{def}}{=} \neg({y_4}_{[8]} + 1_{[8]} \leq_u {y_2}_{[8]}) \wedge ({y_4}_{[8]} = 1_{[8]})$$

Encoding into LIA:

$$\begin{aligned} A_{\text{LIA}} \stackrel{\text{def}}{=} & (x_{y_2} = 16x_{y_5} + x_{y_5}) \wedge (x_{y_1} = x_{y_2}) \wedge (x_{y_5} = 1) \wedge \\ & (x_{y_1} \in [0, 2^8)) \wedge (x_{y_2} \in [0, 2^8)) \wedge (x_{y_5} \in [0, 2^4)) \end{aligned}$$

$$\begin{aligned} B_{\text{LIA}} \stackrel{\text{def}}{=} & \neg(x_{y_4+1} \leq x_{y_2}) \wedge (x_{y_4+1} = x_{y_4} + 1 - 2^8\sigma) \wedge \\ & (x_{y_4} = 1) \wedge \\ & (x_{y_4+1} \in [0, 2^8)) \wedge (x_{y_4} \in [0, 2^8)) \wedge (0 \leq \sigma \leq 1) \end{aligned}$$

From LIA- to BV-interpolants: examples

$$A \stackrel{\text{def}}{=} ({y_1}_{[8]} = {y_5}_{[4]} :: {y_5}_{[4]}) \wedge ({y_1}_{[8]} = {y_2}_{[8]}) \wedge ({y_5}_{[4]} = 1_{[4]})$$

$$B \stackrel{\text{def}}{=} \neg({y_4}_{[8]} + 1_{[8]} \leq_u {y_2}_{[8]}) \wedge ({y_4}_{[8]} = 1_{[8]})$$

LIA-Interpolant:

$$I_{\text{LIA}} \stackrel{\text{def}}{=} (17 \leq x_{y_2})$$

BV-interpolant:

$$I \stackrel{\text{def}}{=} (17_{[8]} \leq_u {y_2}_{[8]})$$

Good!

From LIA- to BV-interpolants: examples

$$A \stackrel{\text{def}}{=} (y_{2[8]} = 81_{[8]}) \wedge (y_{3[8]} = 0_{[8]}) \wedge (y_{4[8]} = y_{2[8]})$$

$$B \stackrel{\text{def}}{=} (y_{13[16]} = 0_{[8]} :: y_{4[8]}) \wedge (255_{[16]} \leq_u y_{13[16]} + (0_{[8]} :: y_{3[8]}))$$

Encoding into LIA:

$$\begin{aligned} A_{\text{LIA}} \stackrel{\text{def}}{=} & (x_{y_2} = 81) \wedge (x_{y_3} = 0) \wedge (x_{y_4} = x_{y_2}) \wedge \\ & (x_{y_2} \in [0, 2^8)) \wedge (x_{y_3} \in [0, 2^8)) \wedge (x_{y_4} \in [0, 2^8)) \end{aligned}$$

$$\begin{aligned} B_{\text{LIA}} \stackrel{\text{def}}{=} & (x_{y_{13}} = 2^8 \cdot 0 + x_{y_4}) \wedge (255 \leq x_{y_{13}+(0::y_3)}) \wedge \\ & (x_{y_{13}+(0::y_3)} = x_{y_{13}} + 2^8 \cdot 0 + x_{y_3} - 2^{16}\sigma) \wedge \\ & (x_{y_{13}} \in [0, 2^{16})) \wedge (x_{y_{13}+(0::y_3)} \in [0, 2^{16})) \wedge \\ & (0 \leq \sigma \leq 1) \end{aligned}$$

From LIA- to BV-interpolants: examples

$$A \stackrel{\text{def}}{=} (y_{2[8]} = 81_{[8]}) \wedge (y_{3[8]} = 0_{[8]}) \wedge (y_{4[8]} = y_{2[8]})$$

$$B \stackrel{\text{def}}{=} (y_{13[16]} = 0_{[8]} :: y_{4[8]}) \wedge (255_{[16]} \leq_u y_{13[16]} + (0_{[8]} :: y_{3[8]}))$$

LIA-interpolant:

$$I_{\text{LIA}} \stackrel{\text{def}}{=} (x_{y_3} + x_{y_4} \leq 81)$$

BV-interpolant:

$$\hat{I} \stackrel{\text{def}}{=} (y_{3[8]} + y_{4[8]} \leq_u 81_{[8]})$$

Wrong!

$B \wedge \hat{I} \not\models \bot$

From LIA- to BV-interpolants: examples

$$A \stackrel{\text{def}}{=} (y_{2[8]} = 81_{[8]}) \land (y_{3[8]} = 0_{[8]}) \land (y_{4[8]} = y_{2[8]})$$

$$B \stackrel{\text{def}}{=} (y_{13[16]} = 0_{[8]} :: y_{4[8]}) \land (255_{[16]} \leq_u y_{13[16]} + (0_{[8]} :: y_{3[8]}))$$

LIA-interpolant:

$$I_{\text{LIA}} \stackrel{\text{def}}{=} (x_{y_3} + x_{y_4} \leq 81)$$

BV-interpolant:

$$\hat{I} \stackrel{\text{def}}{=} (y_{3[8]} + y_{4[8]} \leq_u 81_{[8]})$$

Addition might overflow in BV!

Wrong!

$B \land \hat{I} \not\models \bot$

From LIA- to BV-interpolants: examples

$$A \stackrel{\text{def}}{=} (y_{2[8]} = 81_{[8]}) \wedge (y_{3[8]} = 0_{[8]}) \wedge (y_{4[8]} = y_{2[8]})$$

$$B \stackrel{\text{def}}{=} (y_{13[16]} = 0_{[8]} :: y_{4[8]}) \wedge (255_{[16]} \leq_u y_{13[16]} + (0_{[8]} :: y_{3[8]}))$$

LIA-interpolant:

$$I_{\text{LIA}} \stackrel{\text{def}}{=} (x_{y_3} + x_{y_4} \leq 81)$$

Addition might overflow in BV!

BV-interpolant:

A correct interpolant would be

$$I \stackrel{\text{def}}{=} (0_{[1]} :: y_{3[8]} + 0_{[1]} :: y_{4[8]} \leq_u 81_{[9]})$$

Wrong!

$B \wedge \hat{I} \not\models \bot$

From LIA- to BV-interpolants: examples

$A \stackrel{\text{def}}{=} \neg({y_4}_{[8]} + 1_{[8]} \leq_u {y_3}_{[8]}) \wedge ({y_2}_{[8]} = {y_4}_{[8]} + 1_{[8]})$

$B \stackrel{\text{def}}{=} ({y_2}_{[8]} + 1_{[8]} \leq_u {y_3}_{[8]}) \wedge ({y_7}_{[8]} = 3_{[8]}) \wedge ({y_7}_{[8]} = {y_2}_{[8]} + 1_{[8]})$

Encoding into LIA:

$$\begin{aligned} A_{\text{LIA}} \stackrel{\text{def}}{=} & \neg(x_{y_4+1} \leq x_{y_3}) \wedge (x_{y_2} = x_{y_4+1}) \wedge \\ & (x_{y_4+1} = x_{y_4} + 1 - 2^8\sigma_1) \wedge \\ & (x_{y_2} \in [0, 2^8)) \wedge (x_{y_3} \in [0, 2^8)) \wedge (x_{y_4} \in [0, 2^8)) \wedge \\ & (x_{y_4+1} \in [0, 2^8)) \wedge (0 \leq \sigma_1 \leq 1) \end{aligned}$$

$$\begin{aligned} B_{\text{LIA}} \stackrel{\text{def}}{=} & (x_{y_2+1} \leq x_{y_3}) \wedge (x_{y_7} = 3) \wedge (x_{y_7} = x_{y_2+1}) \wedge \\ & (x_{y_2+1} = x_{y_2} + 1 - 2^8\sigma_2) \wedge \\ & (x_{y_7} \in [0, 2^8)) \wedge (x_{y_2+1} \in [0, 2^8)) \wedge (0 \leq \sigma_2 \leq 1) \end{aligned}$$

From LIA- to BV-interpolants: examples

$$A \stackrel{\text{def}}{=} \neg(y_{4[8]} + 1_{[8]} \leq_u y_{3[8]}) \wedge (y_{2[8]} = y_{4[8]} + 1_{[8]})$$

$$B \stackrel{\text{def}}{=} (y_{2[8]} + 1_{[8]} \leq_u y_{3[8]}) \wedge (y_{7[8]} = 3_{[8]}) \wedge (y_{7[8]} = y_{2[8]} + 1_{[8]})$$

LIA-interpolant:

$$I_{\text{LIA}} \stackrel{\text{def}}{=} (-255 \leq x_{y_2} - x_{y_3} + 256\lfloor -1\frac{x_{y_2}}{256}\rfloor)$$

BV-interpolant: (after fixing overflows)

$$\hat{I}' \stackrel{\text{def}}{=} (65281_{[16]} \leq_u (0_{[8]} :: y_{2[8]}) - (0_{[8]} :: y_{3[8]}) + 256_{[16]} \cdot (65535_{[16]} \cdot (0_{[8]} :: y_{2[8]}) /_u 256_{[16]}))$$

From LIA- to BV-interpolants: examples

$$A \stackrel{\text{def}}{=} \neg(y_{4[8]} + 1_{[8]} \leq_u y_{3[8]}) \wedge (y_{2[8]} = y_{4[8]} + 1_{[8]})$$

$$B \stackrel{\text{def}}{=} (y_{2[8]} + 1_{[8]} \leq_u y_{3[8]}) \wedge (y_{7[8]} = 3_{[8]}) \wedge (y_{7[8]} = y_{2[8]} + 1_{[8]})$$

LIA-interpolant:

$$I_{\text{LIA}} \stackrel{\text{def}}{=} (-255 \leq x_{y_2} - x_{y_3} + 256\lfloor -1\frac{x_{y_2}}{256}\rfloor)$$

BV-interpolant: (after fixing overflows)

$$\hat{I}' \stackrel{\text{def}}{=} (65281_{[16]} \leq_u (0_{[8]} :: y_{2[8]}) - (0_{[8]} :: y_{3[8]}) + 256_{[16]} \cdot (65535_{[16]} \cdot (0_{[8]} :: y_{2[8]}) /_u 256_{[16]}))$$

In this case, the problem is also the sign

Still Wrong!

From LIA- to BV-interpolants: examples

$A \stackrel{\text{def}}{=} \neg({y_4}_{[8]} + 1_{[8]} \leq_u {y_3}_{[8]}) \wedge ({y_2}_{[8]} = {y_4}_{[8]} + 1_{[8]})$

$B \stackrel{\text{def}}{=} ({y_2}_{[8]} + 1_{[8]} \leq_u {y_3}_{[8]}) \wedge ({y_7}_{[8]} = 3_{[8]}) \wedge ({y_7}_{[8]} = {y_2}_{[8]} + 1_{[8]})$

LIA-interpolant:

$$I_{\text{LIA}} \stackrel{\text{def}}{=} (-255 \leq x_{y_2} - x_{y_3} + 256\lfloor -1\frac{x_{y_2}}{256} \rfloor)$$

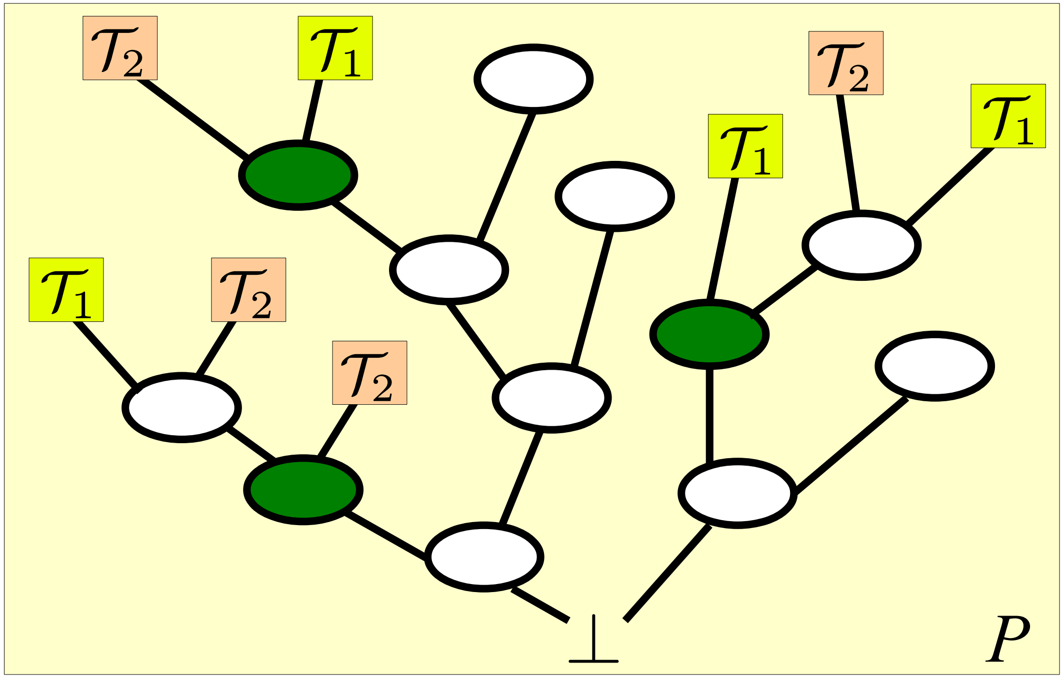
BV-interpolant:

$$I \stackrel{\text{def}}{=} (65281_{[16]} \leq_s (0_{[8]} :: {y_2}_{[8]}) - (0_{[8]} :: {y_3}_{[8]}) + 256_{[16]} \cdot (65535_{[16]} \cdot (0_{[8]} :: {y_2}_{[8]}) /_u 256_{[16]}))$$

Correct interpolant

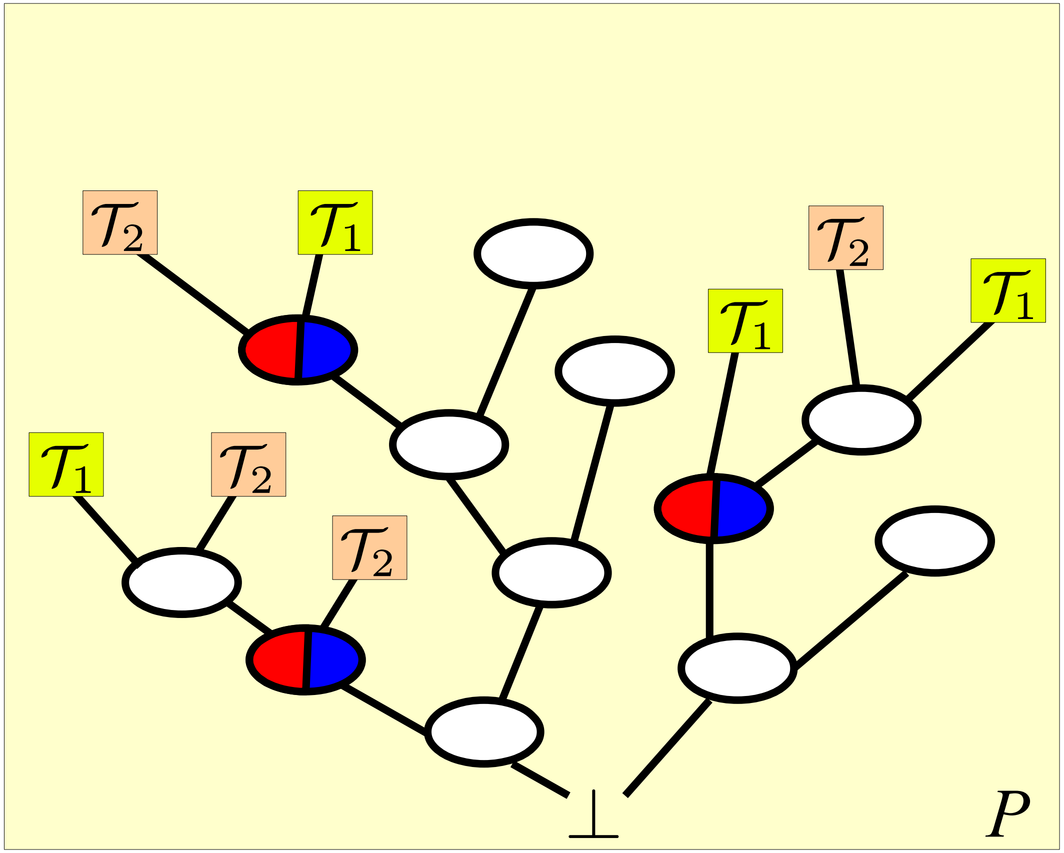
Interpolation in combined theories

- Delayed Theory Combination (DTC): use the DPLL engine to perform theory combination
  - Independent $\mathcal{T}_i$-solvers, that interact only with DPLL
  - How: Boolean search space augmented with interface equalities
    - Equalities between variables shared by the two theories
- Combination of theories encoded directly in the proof of unsatisfiability P
  - $\mathcal{T}_i$-lemmas for the individual theories
  - ***P*** contains interface equalities



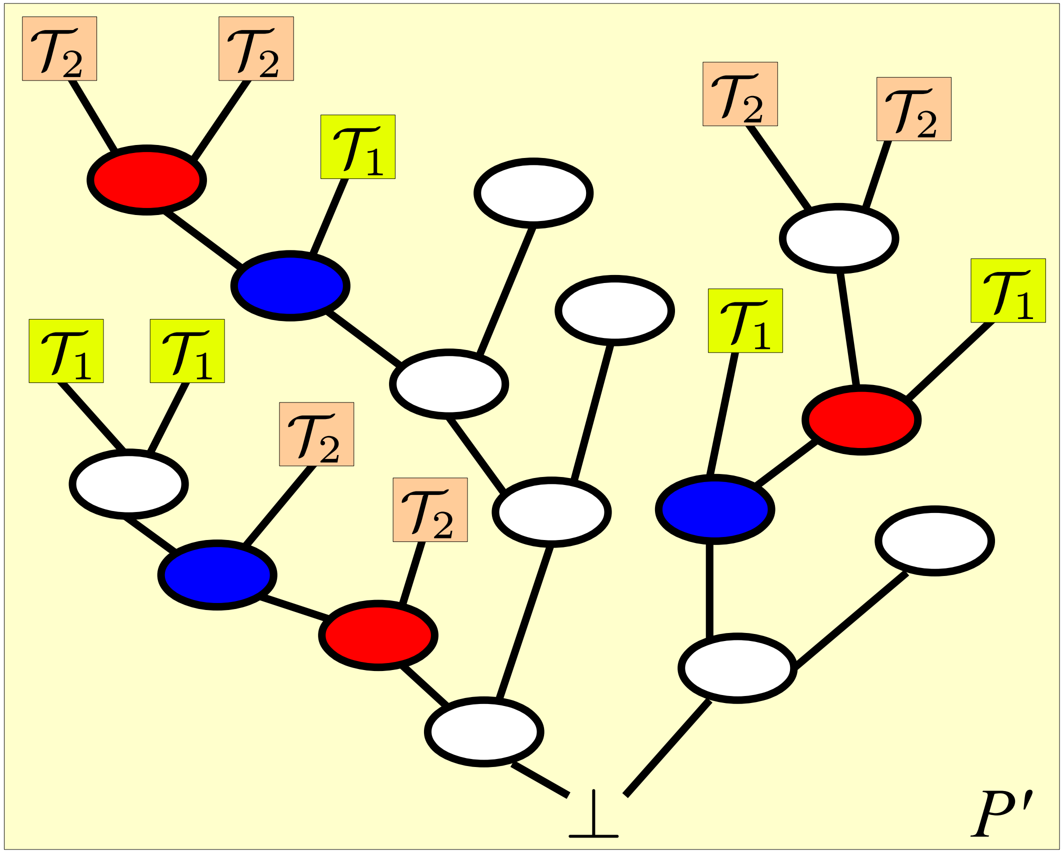
Interpolation in combined theories

- Problem for interpolation:
  - Some interface equalities ($x = y$) are AB-mixed: $x \not\in B,\ y \not\in A$
  - *Interpolation procedures don't work with AB-mixed terms*
- Solution: *Split AB-mixed equalities occurring in P, and fix the proof*
  - How: Split each $\mathcal{T}$-lemma $\eta \vee (x = y)$ into $(\eta \vee (x = t)) \wedge \eta \vee (t = y)$ with $t \in A \cap B$ using available algorithms
  - $\mathcal{T}_i$'s must be equality-interpolating and convex
  - Propagate the changes throughout P



Interpolation in combined theories

- Problem for interpolation:
  - Some interface equalities ($x = y$) are AB-mixed: $x \notin B,\ y \notin A$
  - *Interpolation procedures don't work with AB-mixed terms*
- Solution: *Split AB-mixed equalities occurring in P, and fix the proof*
  - How: Split each $\mathcal{T}$-lemma $\eta \vee (x = y)$ into $(\eta \vee (x = t)) \wedge \eta \vee (t = y)$ with $t \in A \cap B$ using available algorithms
  - $\mathcal{T}_i$'s must be equality-interpolating and convex
  - Propagate the changes throughout P

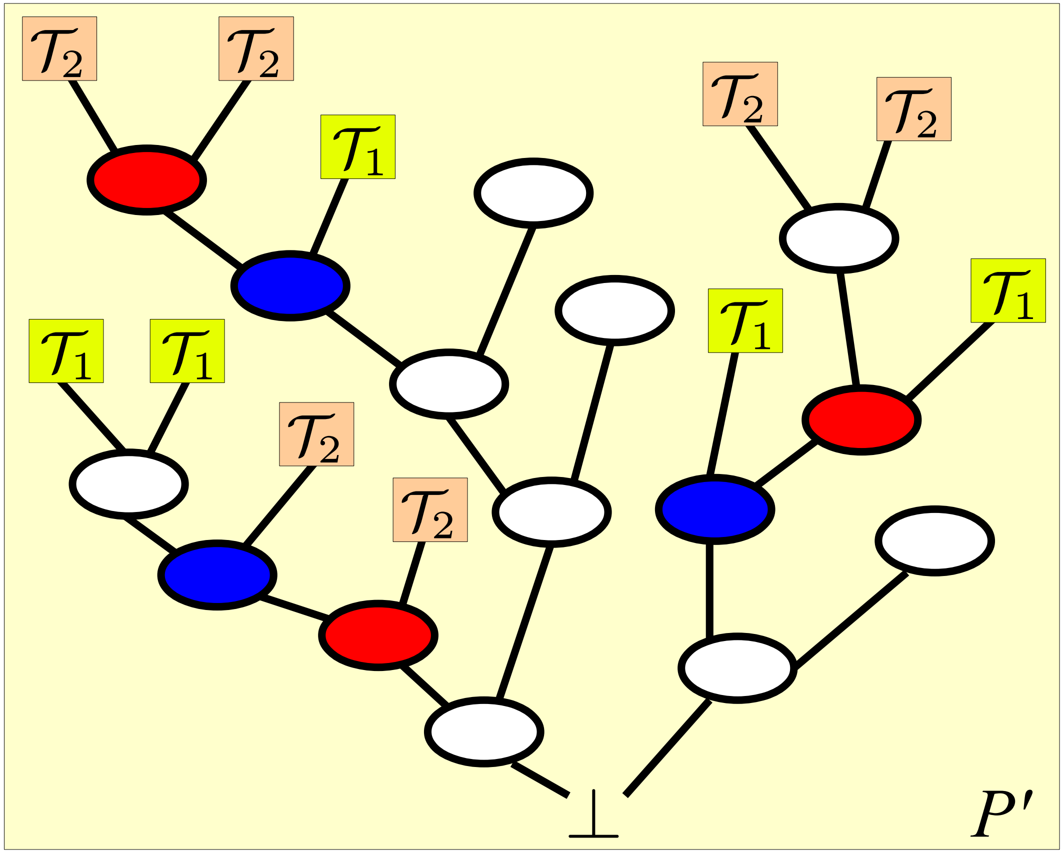


Interpolation in combined theories

- Problem for interpolation:
  - Some interface equalities ($x = y$) are AB-mixed: $x \notin B,\ y \notin A$
  - *Interpolation procedures don't work with AB-mixed terms*
- Solution: *Split AB-mixed equalities occurring in P, and fix the proof*
  - How: Split each $\mathcal{T}$-lemma

Problem: splitting can cause exponential blow-up in P

Solution: control the kind of proofs generated by DPLL, so that the splitting can be performed **efficiently** (ie-local proofs)



Interpolation in combined theories

- After splitting AB-mixed equalities, we can compute an interpolant as usual
  - *Nothing special needed for theory combination!*
  - Because theory combination is encoded in the proof, we can reuse the Boolean interpolation algorithm

- Features:
  - No need of ad-hoc interpolant combination procedures
  - Exploit state-of-the-art SMT solvers, based on (variants of) DTC
  - Split only when necessary

Example

$$A := (a_1 = f(x_1)) \wedge (z - x_1 = 1) \wedge (a_1 + z = 0)$$

$$B := (a_2 = f(x_2)) \wedge (z - x_2 = 1) \wedge (a_2 + z = 1)$$

Example

$$A := (a_1 = f(x_1)) \wedge (z - x_1 = 1) \wedge (a_1 + z = 0)$$

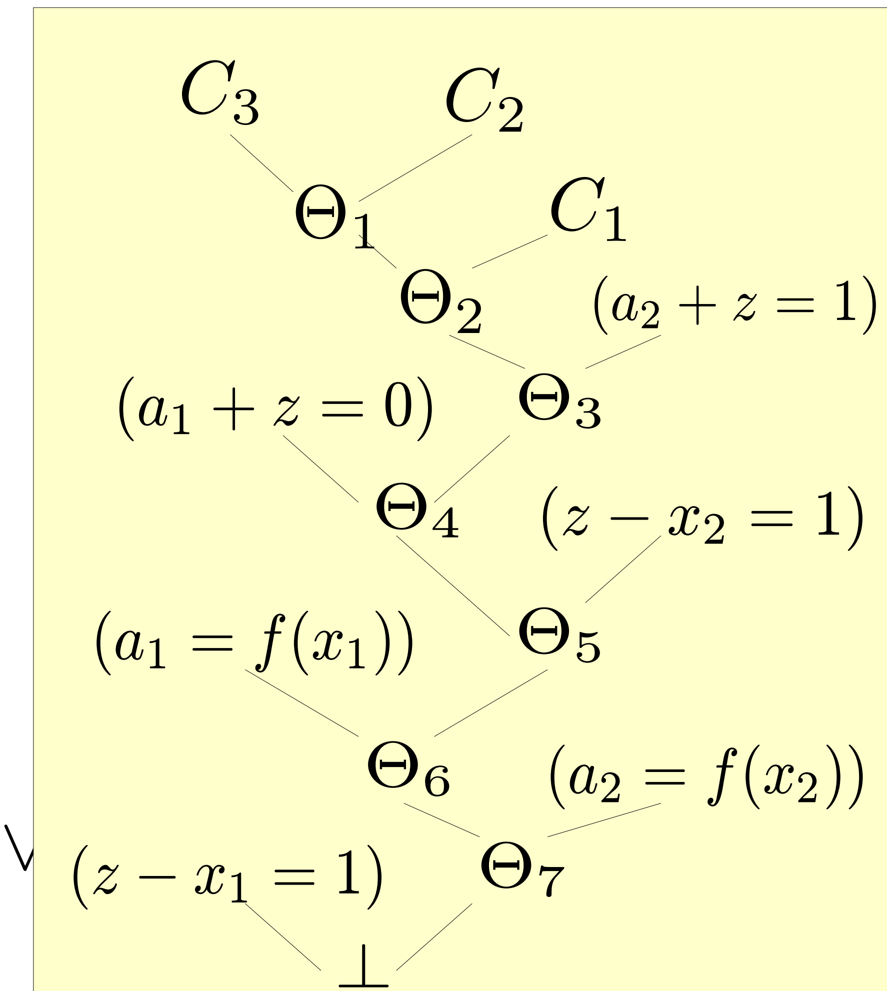
$$B := (a_2 = f(x_2)) \wedge (z - x_2 = 1) \wedge (a_2 + z = 1)$$

T-lemmas:

$$C_1 \equiv (x_1 = x_2) \vee \neg(z - x_1 = 1) \vee \neg(z - x_2 = 1)$$

$$C_2 \equiv (a_1 = a_2) \vee \neg(a_2 = f(x_2)) \vee \neg(a_1 = f(x_1)) \vee \neg(x_1 = x_2)$$

$$C_3 \equiv \neg(a_1 + z = 0) \vee \neg(a_2 + z = 1) \vee \neg(a_1 = a_2)$$



Example

$A := (a_1 = f(x_1)) \wedge (z - x_1 = 1) \wedge (a_1 + z = 0)$

$B := (a_2 = f(x_2)) \wedge (z - x_2 = 1) \wedge (a_2 + z = 1)$

Pivot: $(a_1 = a_2)$

Pivot: $(x_1 = x_2)$

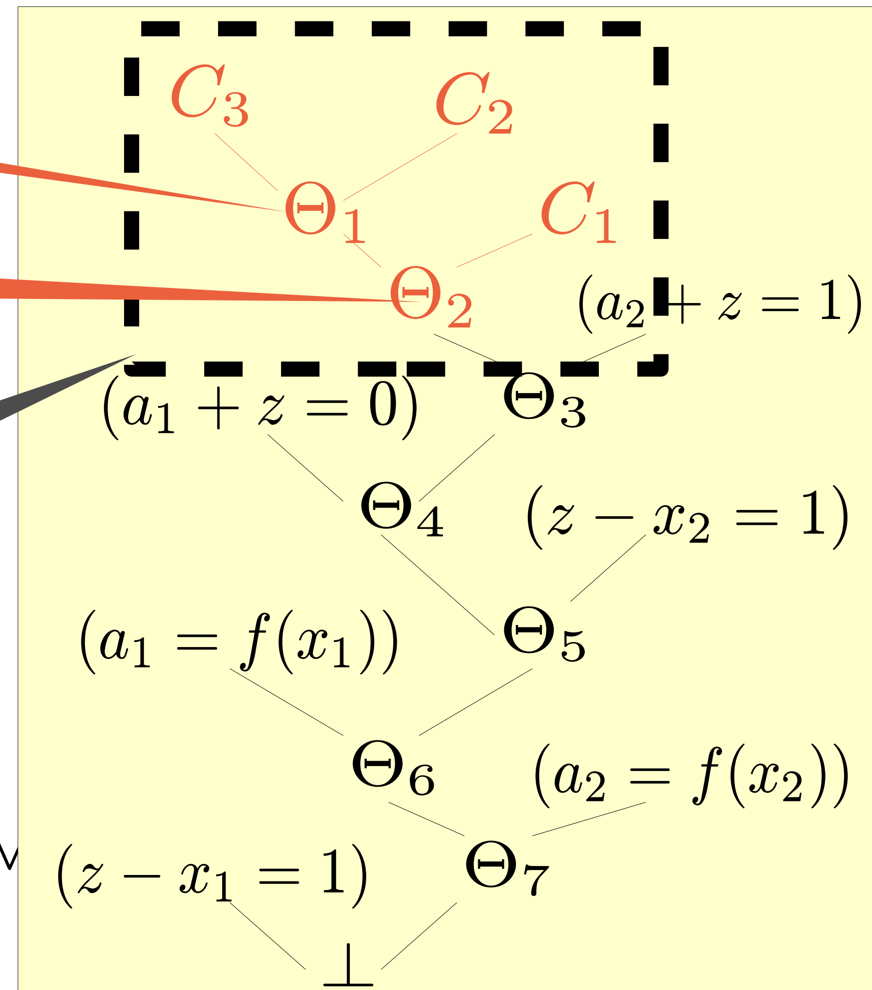
subproof with int.eqs.

T-lemmas:

$C_1 \equiv (x_1 = x_2) \vee \neg(z - x_1 = 1) \vee \neg(z - x_2 = 1)$

$C_2 \equiv (a_1 = a_2) \vee \neg(a_2 = f(x_2)) \vee \neg(a_1 = f(x_1)) \vee \neg(x_1 = x_2)$

$C_3 \equiv \neg(a_1 + z = 0) \vee \neg(a_2 + z = 1) \vee \neg(a_1 = a_2)$

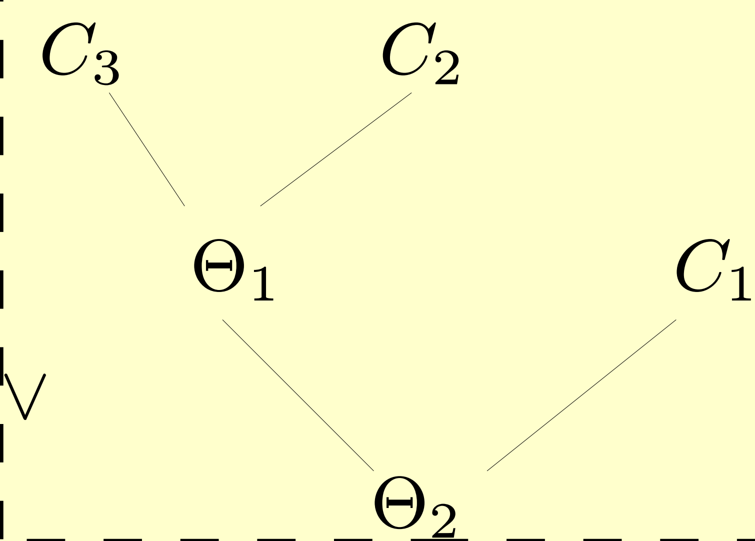


Example

$$A := (a_1 = f(x_1)) \wedge (z - x_1 = 1) \wedge (a_1 + z = 0)$$

$$B := (a_2 = f(x_2)) \wedge (z - x_2 = 1) \wedge (a_2 + z = 1)$$

P^{ie} subproof:



T-lemmas:

$$C_1 \equiv (x_1 = x_2) \vee \neg(z - x_1 = 1) \vee \neg(z - x_2 = 1)$$

$$C_2 \equiv (a_1 = a_2) \vee \neg(a_2 = f(x_2)) \vee \neg(a_1 = f(x_1)) \vee \neg(x_1 = x_2)$$

$$C_3 \equiv \neg(a_1 + z = 0) \vee \neg(a_2 + z = 1) \vee \neg(a_1 = a_2)$$

Example

$$A := (a_1 = f(x_1)) \wedge (z - x_1 = 1) \wedge (a_1 + z = 0)$$

$$B := (a_2 = f(x_2)) \wedge (z - x_2 = 1) \wedge (a_2 + z = 1)$$

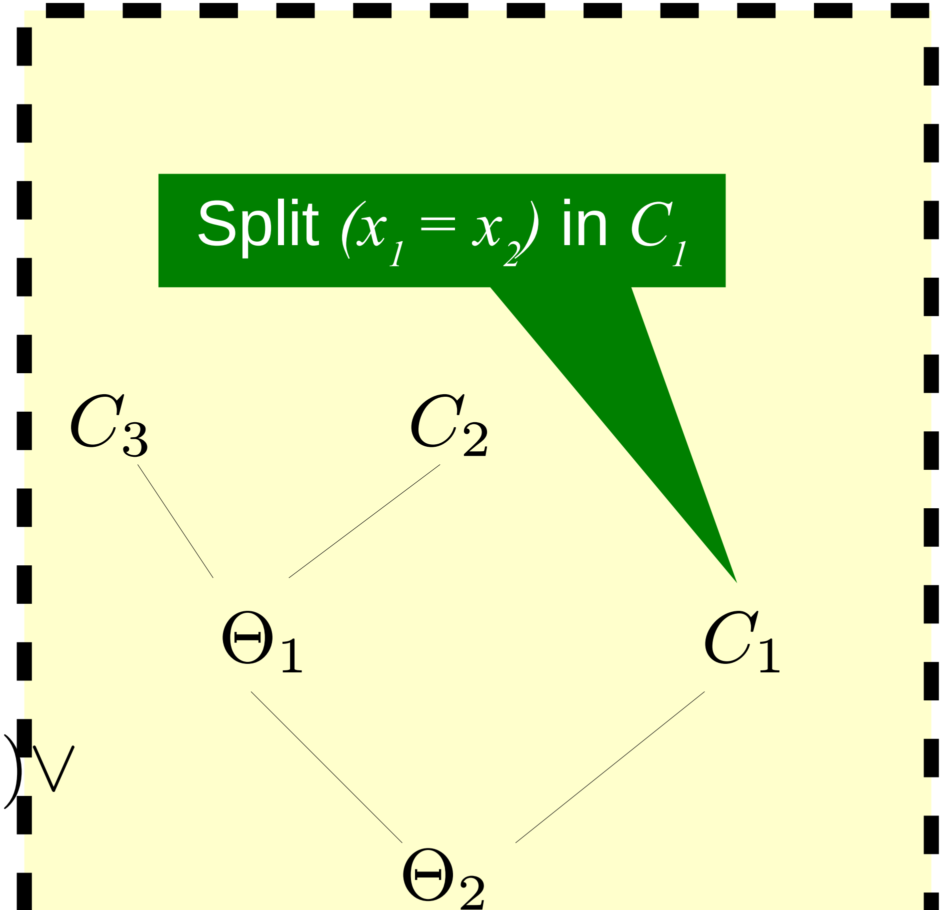
P^{ie} subproof:

T-lemmas:

$$C_1 \equiv (x_1 = x_2) \vee \neg(z - x_1 = 1) \vee \neg(z - x_2 = 1)$$

$$C_2 \equiv (a_1 = a_2) \vee \neg(a_2 = f(x_2)) \vee \neg(a_1 = f(x_1)) \vee \neg(x_1 = x_2)$$

$$C_3 \equiv \neg(a_1 + z = 0) \vee \neg(a_2 + z = 1) \vee \neg(a_1 = a_2)$$



Example

$$A := (a_1 = f(x_1)) \wedge (z - x_1 = 1) \wedge (a_1 + z = 0)$$

$$B := (a_2 = f(x_2)) \wedge (z - x_2 = 1) \wedge (a_2 + z = 1)$$

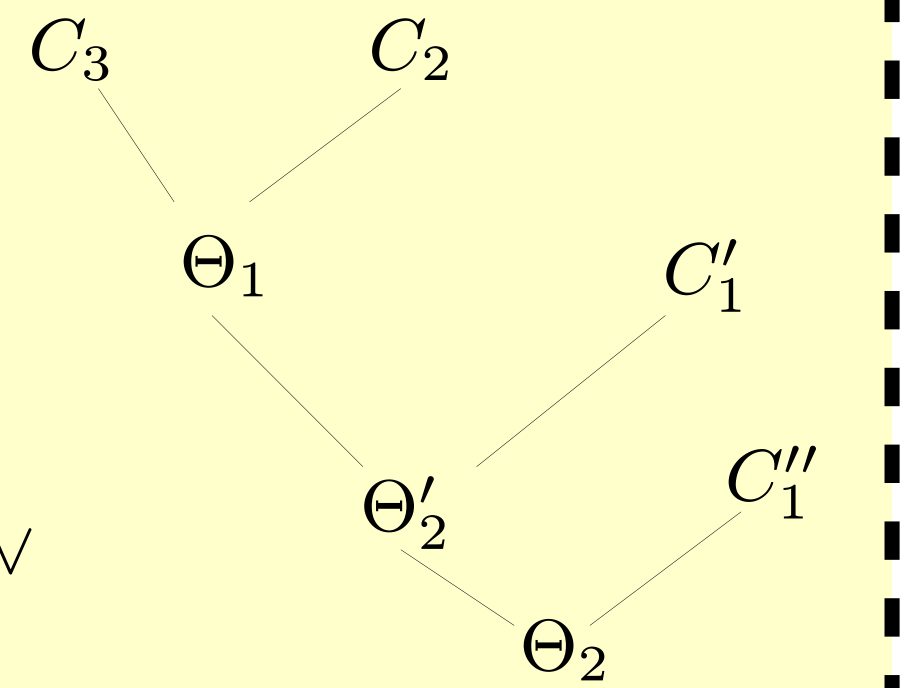
P^{ie} subproof:

$$C_1' \equiv (x_1 = z - 1) \vee \neg(z - x_1 = 1) \vee \neg(z - x_2 = 1)$$

$$C_1'' \equiv (z - 1 = x_2) \vee \neg(z - x_1 = 1) \vee \neg(z - x_2 = 1)$$

$$C_2 \equiv (a_1 = a_2) \vee \neg(a_2 = f(x_2)) \vee \neg(a_1 = f(x_1)) \vee \neg(x_1 = x_2)$$

$$C_3 \equiv \neg(a_1 + z = 0) \vee \neg(a_2 + z = 1) \vee \neg(a_1 = a_2)$$



Example

$$A := (a_1 = f(x_1)) \wedge (z - x_1 = 1) \wedge (a_1 + z = 0)$$

$$B := (a_2 = f(x_2)) \wedge (z - x_2 = 1) \wedge (a_2 + z = 1)$$

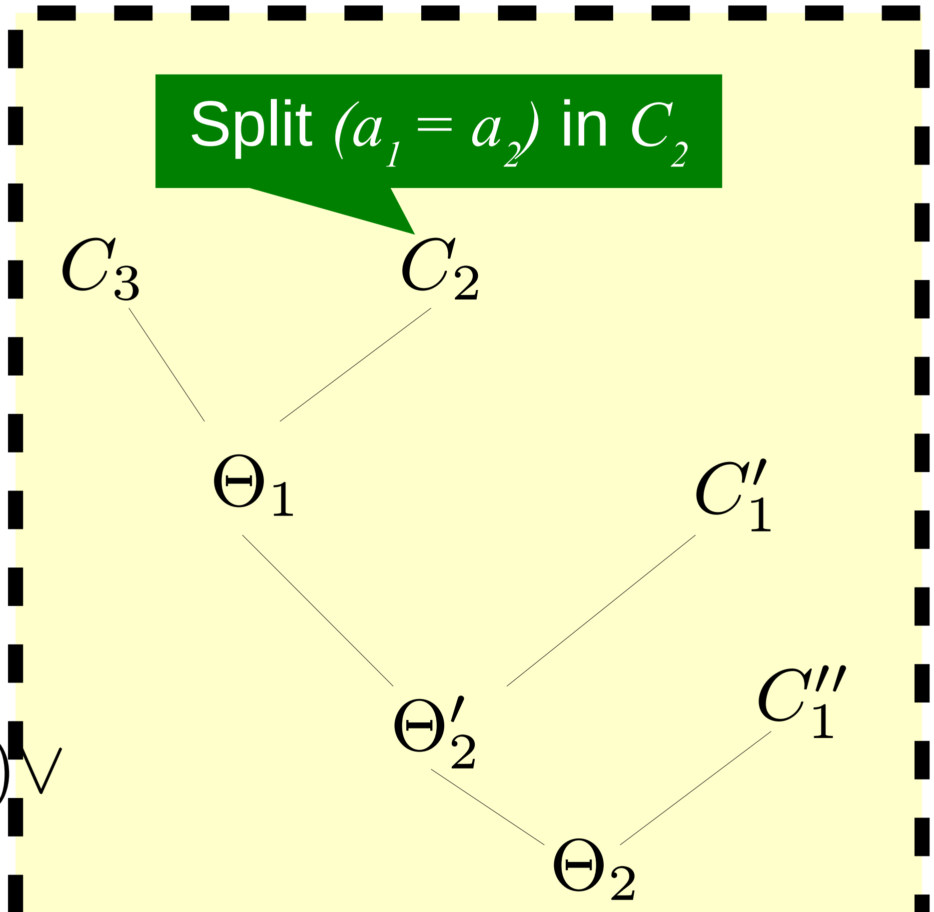
P^{ie} subproof:

$$C_1' \equiv (x_1 = z - 1) \vee \neg(z - x_1 = 1) \vee \neg(z - x_2 = 1)$$

$$C_1'' \equiv (z - 1 = x_2) \vee \neg(z - x_1 = 1) \vee \neg(z - x_2 = 1)$$

$$C_2 \equiv (a_1 = a_2) \vee \neg(a_2 = f(x_2)) \vee \neg(a_1 = f(x_1)) \vee \neg(x_1 = x_2)$$

$$C_3 \equiv \neg(a_1 + z = 0) \vee \neg(a_2 + z = 1) \vee \neg(a_1 = a_2)$$



Example

$$A := (a_1 = f(x_1)) \wedge (z - x_1 = 1) \wedge (a_1 + z = 0)$$

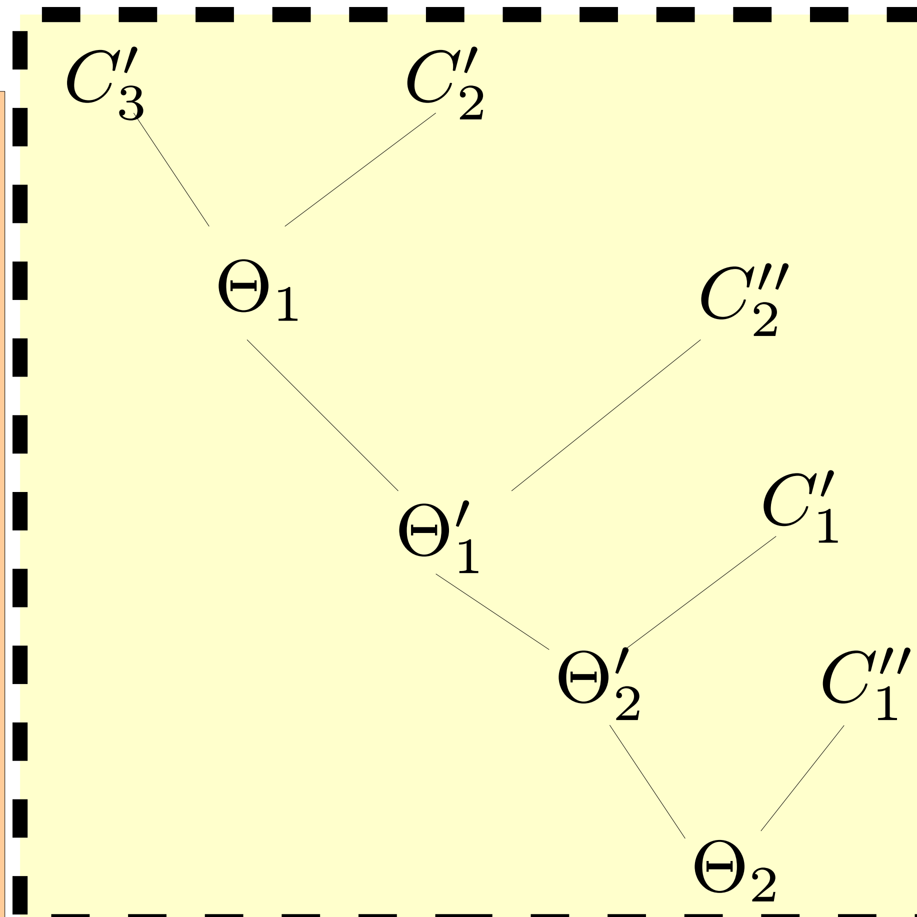
$$B := (a_2 = f(x_2)) \wedge (z - x_2 = 1) \wedge (a_2 + z = 1)$$

P^{ie} subproof:

$$C_2' \equiv (a_1 = f(z-1)) \vee \neg(a_2 = f(x_2)) \vee \neg(a_1 = f(x_1)) \vee \neg(x_1 = z-1) \vee \neg(z-1 = x_2)$$

$$C_2'' \equiv (f(z-1) = a_2) \vee \neg(a_2 = f(x_2)) \vee \neg(a_1 = f(x_1)) \vee \neg(x_1 = z-1) \vee \neg(z-1 = x_2)$$

$$C_3' \equiv \neg(a_1 + z = 0) \vee \neg(a_2 + z = 1) \vee \neg(a_1 = f(z-1)) \vee \neg(f(z-1) = a_2)$$



Proof Tree Preserving Interpolation

- [Christ, Hoenicke and Nutz, TACAS 2013]
- Interpolants with AB-mixed literals without proof rewriting
  - Replace AB-mixed terms $(s \leq t)$ with $(s \leq x) \wedge (x \leq t)$ in leaves, where x is a fresh purification variable
  - Eliminate the purification variable when resolving on $(s \leq t)$

$$\frac{C_1 \vee (s \leq t)\ [I_1(x)] \qquad\qquad C_2 \vee \neg(s \leq t)\ [I_2(x)]}{C_1 \vee C_2\ [I_3]}$$

- Advantages:
  - no need of proof rewriting
  - handles also for non-convex theories
- Drawbacks:
  - need *T*-specific interpolation rules for resolution steps
  - more complex interpolation system

From Binary to Sequence Interpolants

- An ordered sequence of formulae $F_1, \ldots, F_n$ such that $\bigwedge_i F_i \models \bot$
- We want a sequence of interpolants $I_1, \ldots, I_{n-1}$ such that
  - I_k is an interpolant for $(\bigwedge_{i=1}^{k} F_i, \bigwedge_{j=k+1}^{n} F_j)$
  - $F_k \wedge I_{k-1} \models I_k$ for all $k \in [2, n-1]$
- Needed in various applications (e.g. abstraction refinement)
- How to compute them?
  - In general, if we compute arbitrary binary interpolants for $(\bigwedge_{i=1}^{k} F_i, \bigwedge_{j=k+1}^{n} F_j)$, the second condition will not hold

A simple solution

- Compute I_1 as an interpolant of $(F_1, \bigwedge_{j=2}^{n} F_j)$
- Compute I_k as an interpolant of $(I_{k-1} \wedge F_k, \bigwedge_{j=k+1}^{n} F_j)$

- **Claim:** I_k is an interpolant for $(\bigwedge_{i=1}^{k} F_i, \bigwedge_{j=k+1}^{n} F_j)$
- Proof (sketch):
  - By ind.hyp. I_{k-1} is an interpolant for $(\bigwedge_{i=1}^{k-1} F_i, \bigwedge_{j=k}^{n} F_j)$ so $\bigwedge_{i=1}^{k-1} F_i \models I_{k-1}$ and $I_{k-1} \wedge F_k \wedge \bigwedge_{j=k+1}^{n} F_j \models \bot$

- Advantages:
  - simple to implement
  - can use any off-the-shelf binary interpolation
- Drawback: requires *n-1* SMT calls

A more efficient algorithm

- Compute an SMT proof of unsatisfiablity *P* for $\bigwedge_{i=1}^{n} F_i$
- Compute each $I_k := \mathrm{Interpolant}(\bigwedge_{i=1}^{k} F_i, \bigwedge_{j=k+1}^{n} F_j)$ from the same proof *P*

- **Theorem:** $F_k \wedge I_{k-1} \models I_k$

A more efficient algorithm

- Compute an SMT proof of unsatisfiablity *P* for $\bigwedge_{i=1}^{n} F_i$
- Compute each $I_k := \mathrm{Interpolant}(\bigwedge_{i=1}^{k} F_i, \bigwedge_{j=k+1}^{n} F_j)$ from the same proof *P*

- **Theorem:** $F_k \wedge I_{k-1} \models I_k$
- Proof (sketch) – case *n=3*:
  - Let *C* be a node of *P* with partial interpolants *I'* and *I''* for the partitionings $(F_1, F_2 \wedge F_3)$ and $(F_1 \wedge F_2, F_3)$ resp. Then we can prove, by induction on the structure of *P*, that:

$$I' \wedge F_2 \models I'' \vee \bigvee \{l \in C \mid \mathrm{var}(l) \not\in F_3\}$$

  - The theorem then follows as a corollary
  - Works also for DTC-rewritten proofs

Selected bibliography

*DISCLAIMER: this is **very** incomplete. Apologies to missing authors/works*

- Interpolants in Formal Verification

  - McMillan. **Interpolation and SAT-based Model Checking**. CAV 2003
  - Henzinger, Jhala, Majumdar, McMillan. **Abstractions from Proofs**. POPL 2004
  - McMillan. **Lazy Abstraction with Interpolants**. CAV 2006
  - Vizel, Grumberg. **Interpolation-Sequence based model checking**. FMCAD 2009
  - Albargouthi, Gurfinkel, Chechick. **Whale: an interpolation-based algorithm for inter-procedural verification**. VMCAI 2012

Selected bibliography

- Interpolants in SAT and SMT

  - McMillan. **An Interpolating Theorem Prover**. TCS 2005.
  - Yorsh, Musuvathi. **A Combination Method for Generating Interpolants**. CADE 2005
  - Cimatti, Griggio, Sebastiani. **Efficient Generation of Craig Interpolants in SMT**. TOCL 2010
  - Rybalchenko, Sofronie-Stokkermans. **Constraint solving for interpolation**. J. Symb. Comput. 45(11): 1212-1233 (2010)
  - Griggio. **Effective Word-Level Interpolation for Software Verification**. FMCAD 2011
  - Brillout, Kroening, Rümmer, Wahl. **An Interpolating Sequent Calculus for Quantifier-Free Presburger Arithmetic**. J. Autom. Reasoning 47(4): 341-367 (2011)

Selected bibliography

- Interpolants in SAT and SMT

  - D'Silva, Kroening, Purandare, Weissenbacher. **Interpolant strength**. VMCAI 2010
  - Goel, Krstic, Tinelli. **Ground interpolation for the theory of equality**. Logical Methods in Computer Science 8(1) 2012
  - Bruttomesso, Ghilardi, Ranise. **Quantifier-free interpolation of a Theory of Arrays**. Logical Methods in Comp. Sci. 8(2) 2012
  - Totla, Wies. **Complete instantiation-based interpolation**. POPL 2013
  - Christ, Hoenicke, Nutz. **Proof Tree Preserving Interpolation**. TACAS 2013
  - Ruemmer, Subotic. **Exploring Interpolants**. FMCAD 2013
  - Bruttomesso, Ghilardi, Ranise. **Quantifier-free interpolation in combinations of equality interpolating theories**. TOCL 2014

Thank You