Indian SAT+SMT school 2016

SMT Tutorial

2. Quantifier Elimination and Interpolation in SMT

Alberto Griggio
Fondazione Bruno Kessler – Trento, Italy

Outline

Introduction

Example Applications in Formal Verification

SMT-based Quantifier Elimination

Computing interpolants in SMT

Introduction

- (Existential) Quantifier Elimination problem: given a formula

$$\varphi := \exists X_1 . \forall X_2 \ldots \exists X_n . \phi(X_1, \ldots, X_n, Y)$$

find a quantifier-free formula $\psi(Y)$ that is equivalent to φ modulo *T*

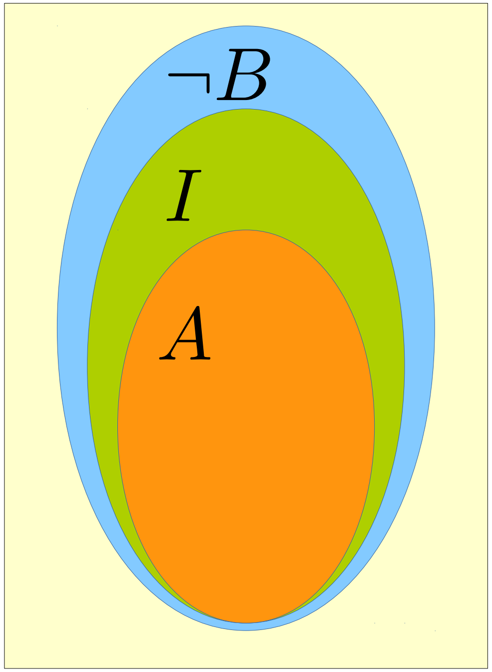
- Several important applications. E.g.
  - Image compuation
  - Parameter synthesis

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 $\exists_{\{x \not\in B\}} \vec{x}.A$
  - "Local" explanation of why A is inconsistent with B

Outline

Introduction

Example Applications in Formal Verification

SMT-based Quantifier Elimination

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:

$R(X) :=$

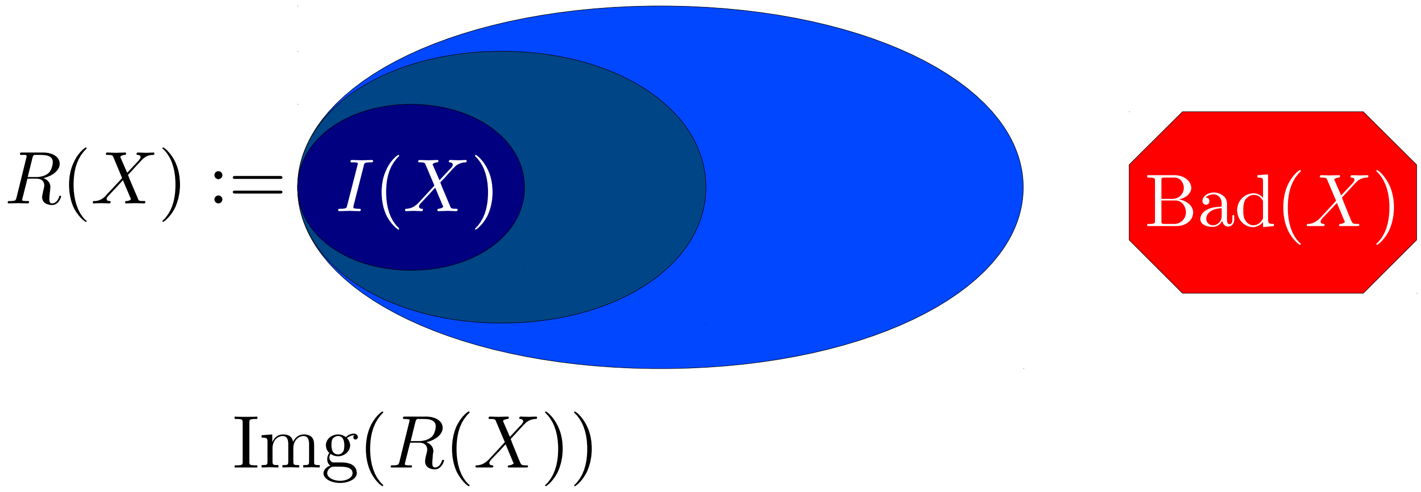
$\mathrm{Img}(R(X))$

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:

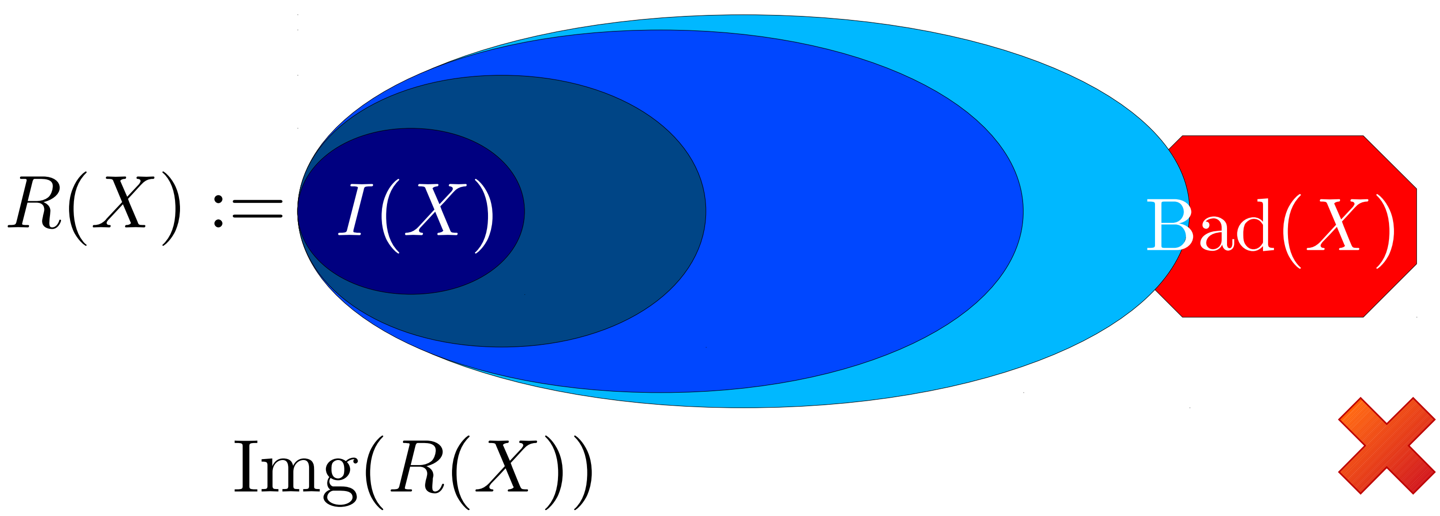


Forward reachability checking

- 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:

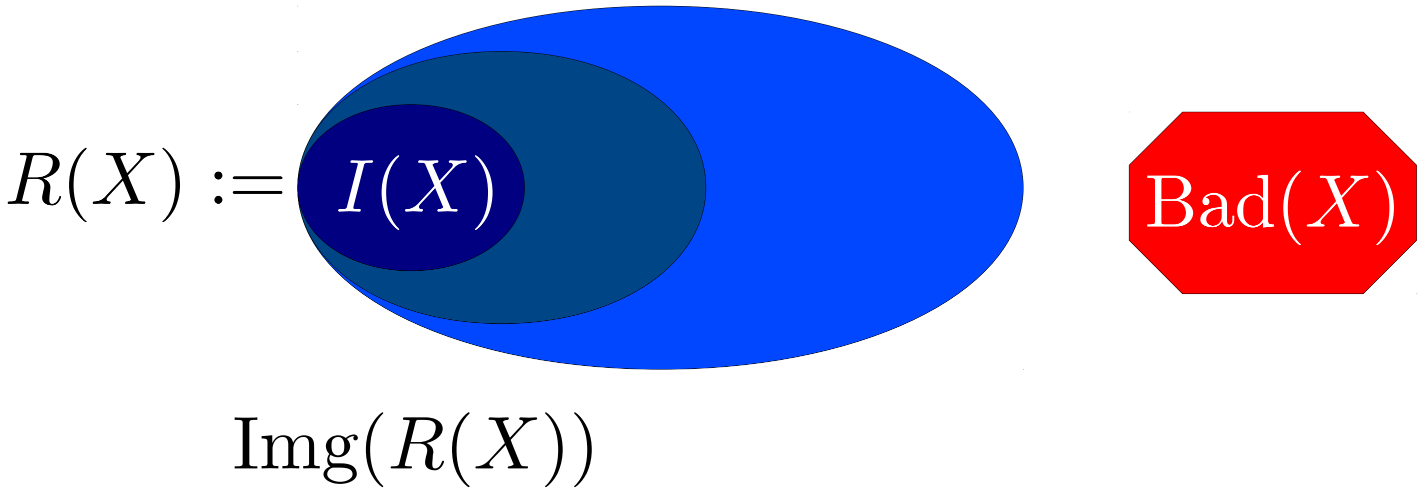


Forward reachability checking

- 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:

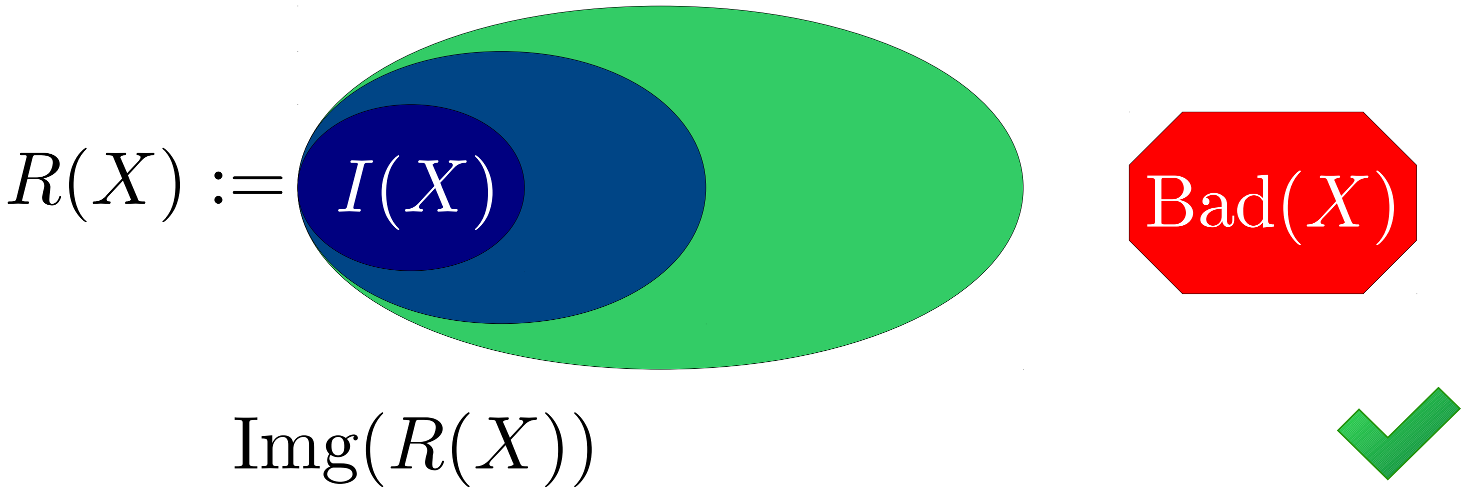


Forward reachability checking

- 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:



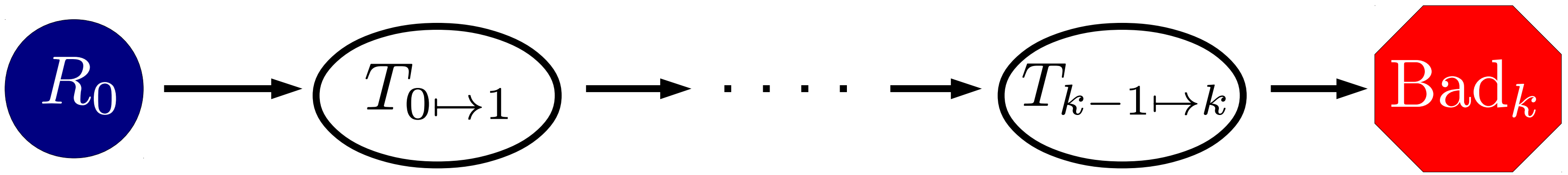
Interpolation-based reachability

- 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

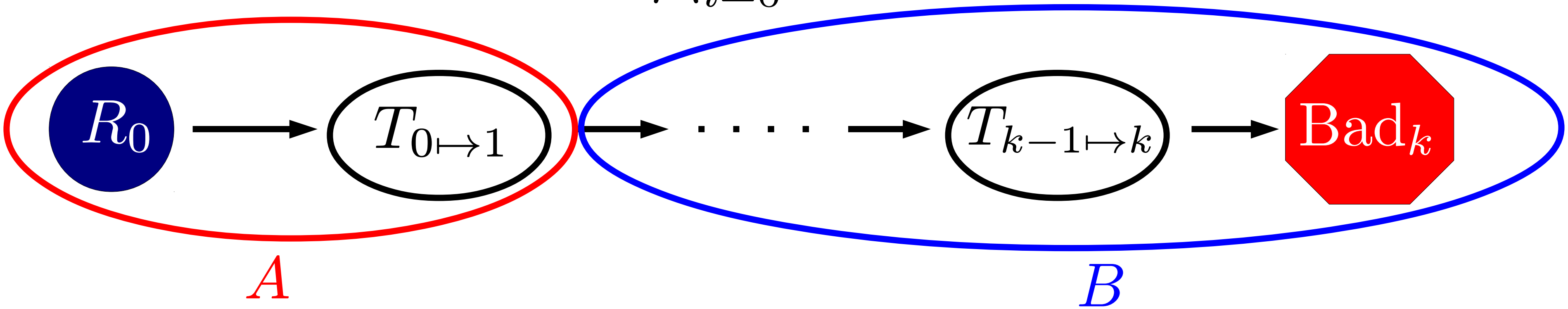
Interpolation-based reachability

- 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 $R_0 \wedge \bigwedge_{i=0}^{k-1} T_i \wedge \mathrm{Bad}_k$

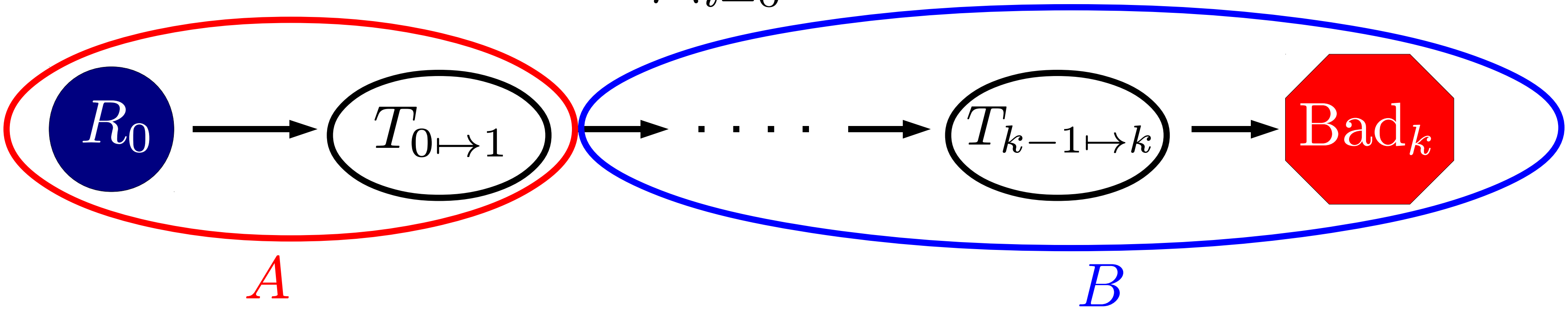


- If UNSAT:
  - Set $\varphi(X) := \mathrm{Interpolant}(A, B)[X_1/X]$

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

Interpolation-based reachability

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



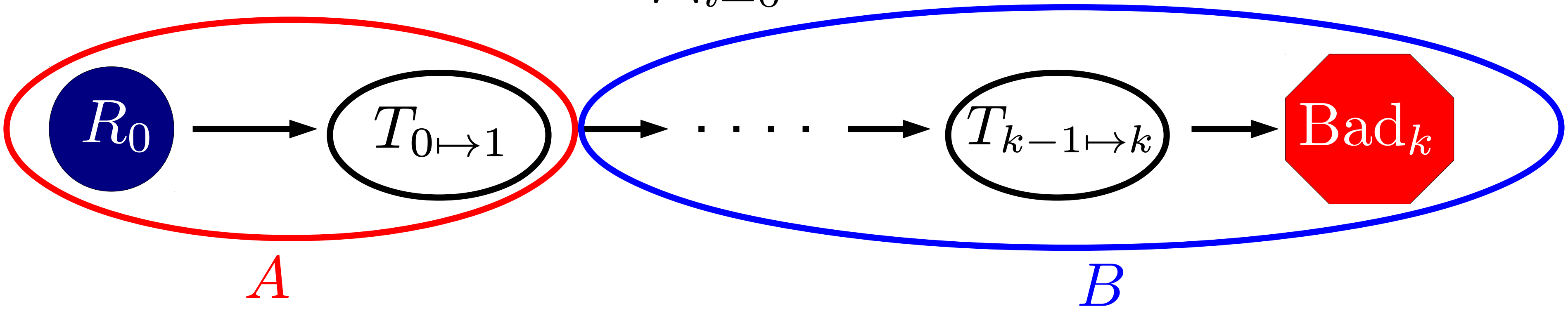
- If UNSAT:
  - Set $\varphi(X) := \mathrm{Interpolant}(A, B)[X_1/X]$

    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 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
    - The unrolling hits bad
  - Otherwise, we don't know
    - The path might be feasible due to the overapproximation
    - Increase *k* and try again

Outline

Introduction

Example Applications in Formal Verification

SMT-based Quantifier Elimination

Computing interpolants in SMT

Existential elimination for LRA

Fourier-Motzkin method

- Given a **conjunction** of linear inequalities C and one variable x to eliminate
  - Partition *C* into C^+ and C^-:

$$C^+ := \{a_i \cdot x \leq \sum_j b_{ij} \cdot y_j + c_i\}_i$$
$$C^- := \{-a_k \cdot x \leq \sum_j b_{kj} \cdot y_j + c_k\}_k$$

  - Return $\psi(Y) := \bigwedge_{ik}(0 \leq \sum_j \frac{a_k \cdot b_{ij} + a_i \cdot b_{kj}}{a_i \cdot a_k} y_j + \frac{a_i \cdot c_k + a_k \cdot c_i}{a_i \cdot a_k})$
- For multiple variables $x_1, \ldots, x_n$, apply the above in sequence
- For arbitrary formulae $\varphi(x, Y)$
  - First put φ in DNF, and then apply the above

Boosting Fourier-Motzkin with SMT

- The Fourier-Motzkin method is a purely syntactical one, which might generate a lot of redundant constraints
  - Blow-up when converting the input to DNF
  - Blow-up when eliminating a single variable
  - Blow-up when eliminating multiple variables
  - No reuse of information
- We can mitigate the blow-ups by using an SMT solver, to perform a “semantic-aware” existential elimination
  - Although it won't improve the (doubly-exponential) worst-case complexity, it will greatly improve performance in practice
    - See e.g. [Monniaux 2008]

SMT-based FM-elim

```
def FM_elim_SMT(formula, vars):
    res = FALSE()
    while True:
        m = get_model(formula)
        if m is None: break
        conj = { (a if entails(m, a) else Not(a))
                  for a in atoms(formula) }
        conj_e = FM_elim_conj(conj, vars)
        res = Or(res, conj_e)
        formula = And(formula, Not(conj_e))
    return res
```

```
def FM_elim_conj(conj, vars):
    for x in vars:
        c_p = { c for c in conj if coeff(c, x) > 0 }
        c_n = { c for c in conj if coeff(c, x) < 0 }
        conj = conj - (c_p | c_n)
        for a in c_p:
            for b in c_n:
                c = combine(a, b, x)
                if not implies(conj, c): conj.add(c)
    return And(*conj)
```

Alternative: Virtual Term Substitution

- Main bottleneck of the FME-based algorithm is the computation of the DNF
  - Even in the SMT case, still has to enumerate cubes of the input formula

- Virtual Term Substitution method doesn't require a DNF
  - Can work on a NNF, which can be computed in linear time

- Main idea: compute a set $S := \{\sigma_1, \ldots, \sigma_n\}$ such that $\exists x.\varphi$ is equivalent to $\bigvee_{i=1}^{n} \varphi[x := \sigma_i]$
  - S is computed syntactically, by only looking at the literals of φ
    - The Boolean structure doesn't matter

Virtual Term Substitution

- Collect all literals containing x
  - Put them in the form $(x \bowtie t_i), \bowtie \in \{<, =, >\}$
    - By rewriting $$(t_1 \leq t_2) \mapsto (t_1 = t_2) \vee (t_1 < t_2)$$ $$\neg(t_1 \leq t_2) \mapsto (t_2 < t_1)$$ (and so on)
- Build $S := \{t_i \mid (x = t_i) \in \varphi\} \cup \{t_i - \varepsilon \mid (x < t_i) \in \varphi\} \cup \{\infty\}$
  - ε is a symbolic infinitesimal parameter
- Apply the subsitutions as follows

$$(x \bowtie t_j)[x := \infty] = \begin{cases} \bot \text{ if } \bowtie \in \{=, <\} \\ \top \text{ if } \bowtie \text{ is } > \end{cases}$$

$$(x \bowtie t_j)[x := t_i - \varepsilon] = \begin{cases} \bot \text{ if } \bowtie \text{ is } = \\ (t_i \leq t_j) \text{ if } \bowtie \text{ is } < \\ (t_i > t_j) \text{ if } \bowtie \text{ is } > \end{cases}$$

Example

- $\varphi := \exists x.[(2y < 5) \wedge ((y > 0) \vee (x < 2y)) \wedge$
  $(((x \geq 0) \wedge (x < 4y)) \vee (y > -5))]$

Example

- $\varphi := \exists x.[(2y < 5) \wedge ((y > 0) \vee (x < 2y)) \wedge$
  $(((x \geq 0) \wedge (x < 4y)) \vee (y > -5))]$
- Collect literals $\{(x < 2y), \underbrace{(x > 0), (x = 0)}_{(x \geq 0)}, (x < 4y)\}$
- $S := \{0, 2y - \varepsilon, 4y - \varepsilon, \infty\}$

Example

- $\varphi := \exists x.[(2y < 5) \wedge ((y > 0) \vee (x < 2y))\wedge$
  $(((x \geq 0) \wedge (x < 4y)) \vee (y > -5))]$
- Collect literals $\{(x < 2y), \underbrace{(x > 0), (x = 0)}_{(x \geq 0)}, (x < 4y)\}$
- $S := \{0, 2y - \varepsilon, 4y - \varepsilon, \infty\}$
- $\varphi[x := 0] = (2y < 5) \wedge ((y > 0) \vee (0 < 2y)) \wedge ((0 < 4y) \vee (y > -5))$

Example

- $\varphi := \exists x.[(2y < 5) \wedge ((y > 0) \vee (x < 2y)) \wedge$
  $(((x \geq 0) \wedge (x < 4y)) \vee (y > -5))]$
- Collect literals $\{(x < 2y), \underbrace{(x > 0), (x = 0)}_{(x \geq 0)}, (x < 4y)\}$
- $S := \{0, 2y - \varepsilon, 4y - \varepsilon, \infty\}$
- $\varphi[x := 0] = (2y < 5) \wedge ((y > 0) \vee (0 < 2y)) \wedge ((0 < 4y) \vee (y > -5))$

  $\varphi[x := 2y - \varepsilon] = (2y < 5) \wedge (((2y > 0) \wedge (2y \leq 4y)) \vee (y > -5))$

Example

- $\varphi := \exists x.[(2y < 5) \wedge ((y > 0) \vee (x < 2y)) \wedge$
  $(((x \geq 0) \wedge (x < 4y)) \vee (y > -5))]$
- Collect literals $\{(x < 2y), \underbrace{(x > 0), (x = 0)}_{(x \geq 0)}, (x < 4y)\}$

- $S := \{0, 2y - \varepsilon, 4y - \varepsilon, \infty\}$
- $\varphi[x := 0] = (2y < 5) \wedge ((y > 0) \vee (0 < 2y)) \wedge ((0 < 4y) \vee (y > -5))$
  $\varphi[x := 2y - \varepsilon] = (2y < 5) \wedge (((2y > 0) \wedge (2y \leq 4y)) \vee (y > -5))$
  $\varphi[x := 4y - \varepsilon] = (2y < 5) \wedge ((y > 0) \vee (4y \leq 2y)) \wedge ((4y > 0) \vee (y > -5))$

Example

- $\varphi := \exists x.[(2y < 5) \wedge ((y > 0) \vee (x < 2y)) \wedge$
  $(((x \geq 0) \wedge (x < 4y)) \vee (y > -5))]$
- Collect literals $\{(x < 2y), \underbrace{(x > 0), (x = 0)}_{(x \geq 0)}, (x < 4y)\}$
- $S := \{0, 2y - \varepsilon, 4y - \varepsilon, \infty\}$
- $\varphi[x := 0] = (2y < 5) \wedge ((y > 0) \vee (0 < 2y)) \wedge ((0 < 4y) \vee (y > -5))$

  $\varphi[x := 2y - \varepsilon] = (2y < 5) \wedge (((2y > 0) \wedge (2y \leq 4y)) \vee (y > -5))$

  $\varphi[x := 4y - \varepsilon] = (2y < 5) \wedge ((y > 0) \vee (4y \leq 2y)) \wedge ((4y > 0) \vee (y > -5))$

  $\varphi[x := \infty] = (2y < 5) \wedge (y > 0) \wedge (y > -5)$

Example

- $\varphi := \exists x.[(2y < 5) \wedge ((y > 0) \vee (x < 2y)) \wedge$
  $(((x \geq 0) \wedge (x < 4y)) \vee (y > -5))]$
- Collect literals $\{(x < 2y), \underbrace{(x > 0), (x = 0)}_{(x \geq 0)}, (x < 4y)\}$
- $S := \{0, 2y - \varepsilon, 4y - \varepsilon, \infty\}$
- $\varphi[x := 0] = (2y < 5) \wedge ((y > 0) \vee (0 < 2y)) \wedge ((0 < 4y) \vee (y > -5))$

  $\varphi[x := 2y - \varepsilon] = (2y < 5) \wedge (((2y > 0) \wedge (2y \leq 4y)) \vee (y > -5))$

  $\varphi[x := 4y - \varepsilon] = (2y < 5) \wedge ((y > 0) \vee (4y \leq 2y)) \wedge ((4y > 0) \vee (y > -5))$

  $\varphi[x := \infty] = (2y < 5) \wedge (y > 0) \wedge (y > -5)$
- Result: $\varphi[x := 0] \vee \varphi[x := 2y - \varepsilon] \vee \varphi[x := 4y - \varepsilon] \vee \varphi[x := \infty]$

Virtual Term Substitution drawbacks

- Like the naive FM algorithm, the VTS method is purely syntactic
  - Doesn't consider the Boolean structure of the formula
  - Many cases might produce inconsistent disjunct, or duplicate and/or subsumed results

- In the previous example:
  - $\varphi[x := 0]$ and $\varphi[x := \infty]$ are equivalent
  - $\varphi[x := 2y - \varepsilon]$ and $\varphi[x := 4y - \varepsilon]$ are equivalent
  - $\varphi[x := 0]$ is implied by $\varphi[x := 2y - \varepsilon]$
- therefore, $\exists x.\varphi$ is equivalent to $\varphi[x := 2y - \varepsilon]$

- We can do better by exploiting SMT

SMT-based Virtual Term Substitution

```
def VTS_elim_SMT(formula, vars):
  f = to_nnf(formula)
  res = FALSE()
  while True:
    m = get_model(f)
    if m is None: break
    d = VTS_elim_model(f, m, vars)
    res = Or(res, d)
    f = And(f, Not(d))
  return res
```

```
def VTS_elim_model(f, vars, m):
  for x in vars:
    S = get_S(f, x)
    val = eval_S(S, x, m)
    f = apply_VTS(f, x, val)
  return f
```

```
def eval_S(S, x, m):
  cur = None
  for c in S:
      if c is (x = t) and m[x] == m[t]:
          return t
      elif c is (x < t) and
             (cur is None or m[t] < m[cur]):
          cur = t
  if cur is not None:
      return cur - EPSILON()
  return INF()
```

SMT-based Virtual Term Substitution

```
def VTS_elim_SMT(formula, vars):
  f = to_nnf(formula)
  res = FALSE()
  while True:
    m = get_model(f)
    if m is None: break
    d = VTS_elim_model(f, m, vars)
    res = Or(res, d)
    f = And(f, Not(d))
  return res
```

```
def VTS_elim_model(f, vars, m):
  for x in vars:
    S = get_S(f, x)
    val = eval_S(S, x, m)
    f = apply_VTS(f, x, val)
  return f
```

Find the virtual substitution that is consistent with the current model

```
def eval_S(S, x, m):
  cur = None
  for c in S:
      if c is (x = t) and m[x] == m[t]:
          return t
      elif c is (x < t) and
            (cur is None or m[t] < m[cur]):
          cur = t
  if cur is not None:
      return cur - EPSILON()
  return INF()
```

SMT-based Virtual Term Substitution

```
def VTS_elim_SMT(formula, vars):
  f = to_nnf(formula)
  res = FALSE()
  while True:
    m = get_model(f)
    if m is None: break
    d = VTS_elim_model(f, m, vars)
    res = Or(res, d)
    f = And(f, Not(d))
  return res
```

```
def VTS_elim_model(f, vars, m):
  for x in vars:
    S = get_S(f, x)
    val = eval_S(S, x, m)
    f = apply_VTS(f, x, val)
  return f
```

Find the virtual substitution that is consistent with the current model

Do not explore already-covered models

```
def eval_S(S, x, m):
  cur = None
  for c in S:
      if c is (x = t) and m[x] == m[t]:
          return t
      elif c is (x < t) and
            (cur is None or m[t] < m[cur]):
          cur = t
  if cur is not None:
      return cur - EPSILON()
  return INF()
```

Outline

Introduction

Example Applications in Formal Verification

SMT-based Quantifier Elimination

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

$$A := (x \vee \neg y_1) \wedge (\neg x \vee \neg y_2) \wedge y_1$$

$$B := (\neg y_1 \vee y_2) \wedge (y_1 \vee z) \wedge \neg z$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{x \vee \neg y_1 \qquad \neg x \vee \neg y_2}{\neg y_1 \vee \neg y_2} \qquad y_1}{\neg y_2} \qquad \neg y_1 \vee y_2}{y_1 \vee z \qquad \neg y_1}}{z \qquad \neg z}}{\bot}$$

Example

$$\color{red}{A} := (x \vee \neg y_1) \wedge (\neg x \vee \neg y_2) \wedge y_1$$

$$\color{blue}{B} := (\neg y_1 \vee y_2) \wedge (y_1 \vee z) \wedge \neg z$$

$$\frac{x \vee \neg y_1 \ \boxed{\neg y_1} \qquad \neg x \vee \neg y_2 \ \boxed{\neg y_2}}{\neg y_1 \vee \neg y_2 \ \boxed{\neg y_1 \vee \neg y_2}}$$

$$\frac{\neg y_1 \vee \neg y_2 \ \boxed{\neg y_1 \vee \neg y_2} \qquad y_1 \ \boxed{y_1}}{\neg y_2 \ \boxed{(\neg y_1 \vee \neg y_2) \wedge y_1}}$$

$$\frac{\neg y_2 \ \boxed{(\neg y_1 \vee \neg y_2) \wedge y_1} \qquad \neg y_1 \vee y_2 \ \boxed{\top}}{\neg y_1 \ \boxed{(\neg y_1 \vee \neg y_2) \wedge y_1}}$$

$$\frac{y_1 \vee z \ \boxed{\top} \qquad \neg y_1 \ \boxed{(\neg y_1 \vee \neg y_2) \wedge y_1}}{z \ \boxed{(\neg y_1 \vee \neg y_2) \wedge y_1}}$$

$$\frac{z \ \boxed{(\neg y_1 \vee \neg y_2) \wedge y_1} \qquad \neg z \ \boxed{\top}}{\bot \ \boxed{(\neg y_1 \vee \neg y_2) \wedge y_1}}$$

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*:

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

Equality (EUF)

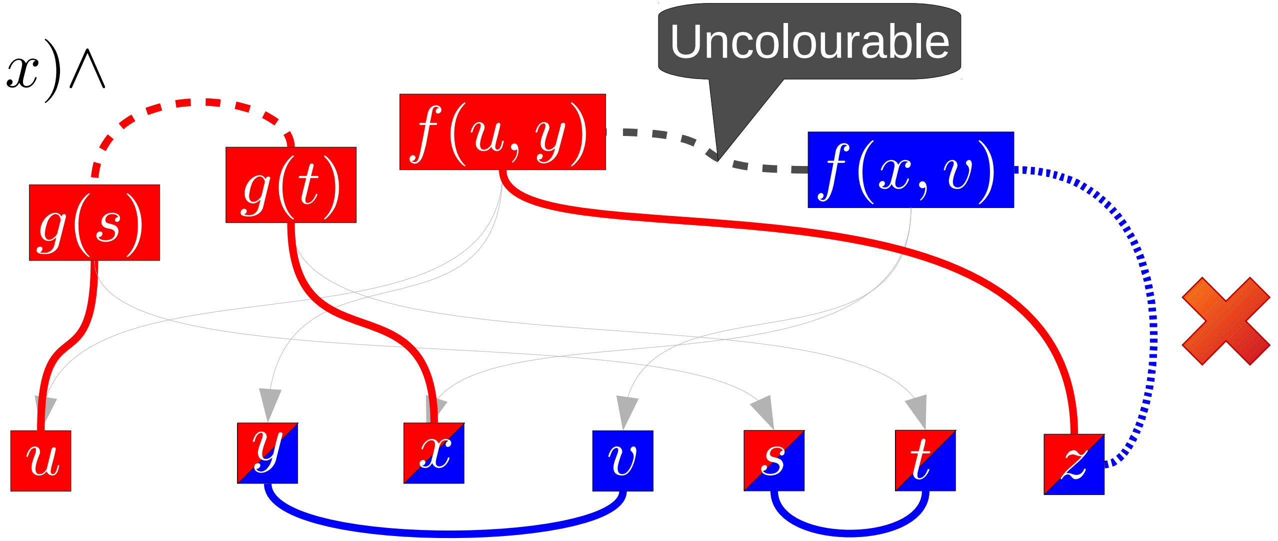
- Interpolants from coloured congruence graphs
  - Nodes with colours:
    - (red square) if term occurs in *A*
    - (blue square) if term occurs in *B*
    - (red/blue square) if term is shared
  - Edges with colours of the nodes they connect
    - Uncolorable edge: connects nodes of two different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

Equality (EUF)

- Interpolants from coloured congruence graphs
  - Nodes with colours: [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)

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

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

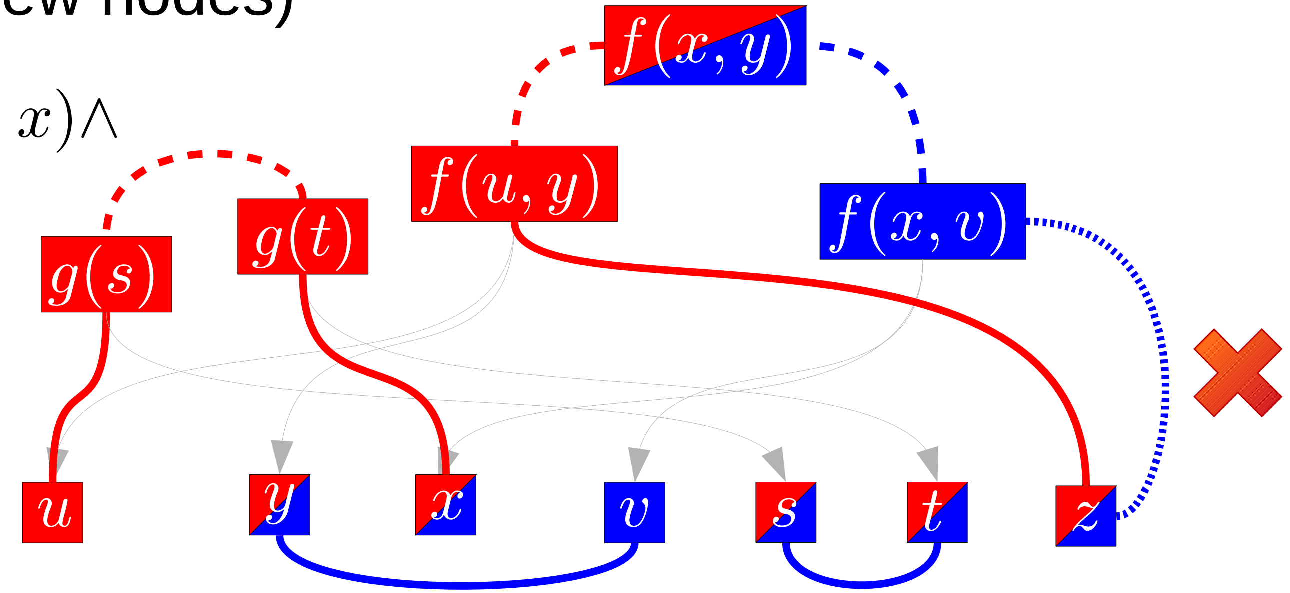


Equality (EUF)

- Interpolants from coloured congruence graphs
  - Nodes with colours:
    - Red square: if term occurs in A
    - Blue square: if term occurs in B
    - Red/blue square: 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)

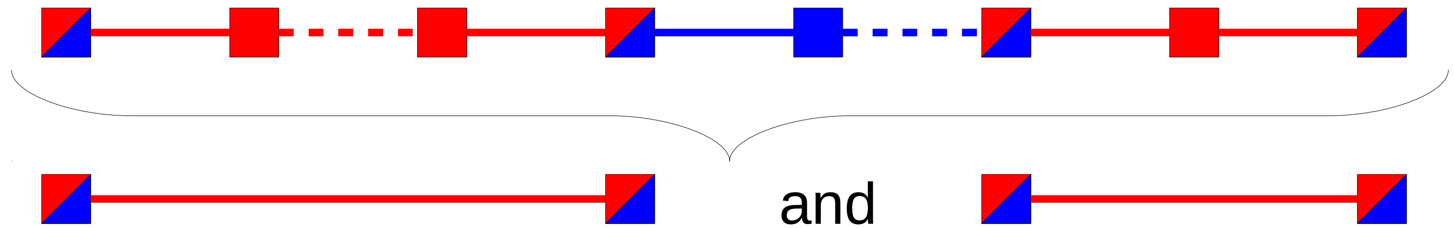
$$A := (u = g(s)) \wedge (g(t) = x) \wedge (f(u, y) = z)$$

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



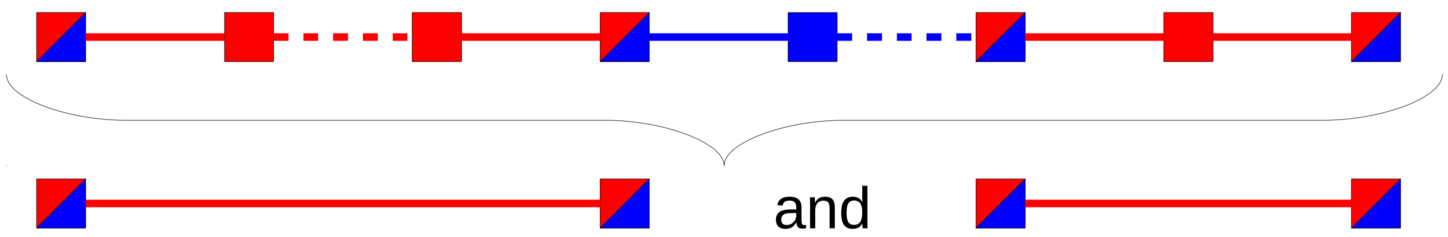
Interpolation algorithm (sketch)

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

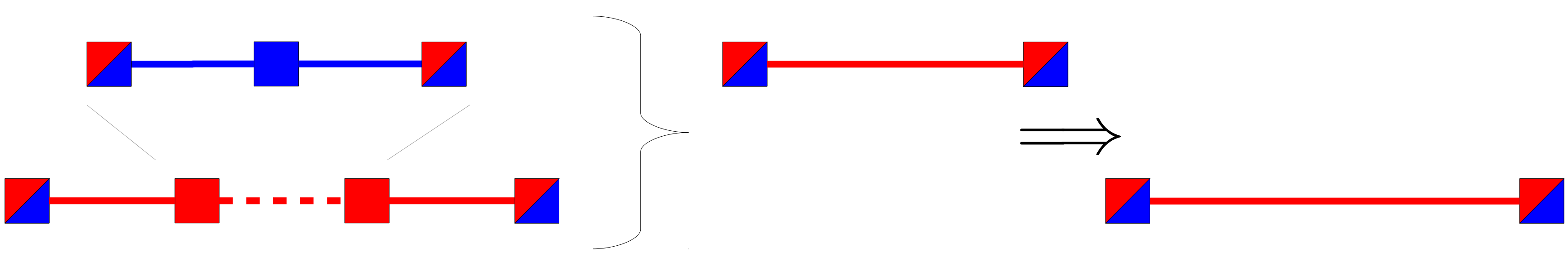


Interpolation algorithm (sketch)

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

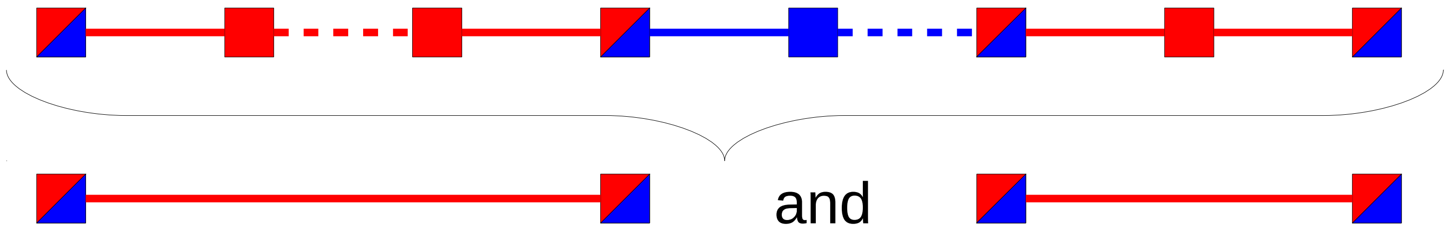


- If an *A*-summary involves a congruence edge, compute summaries recursively on function arguments
  - Use *B*-summaries as premises for the *A*-summary

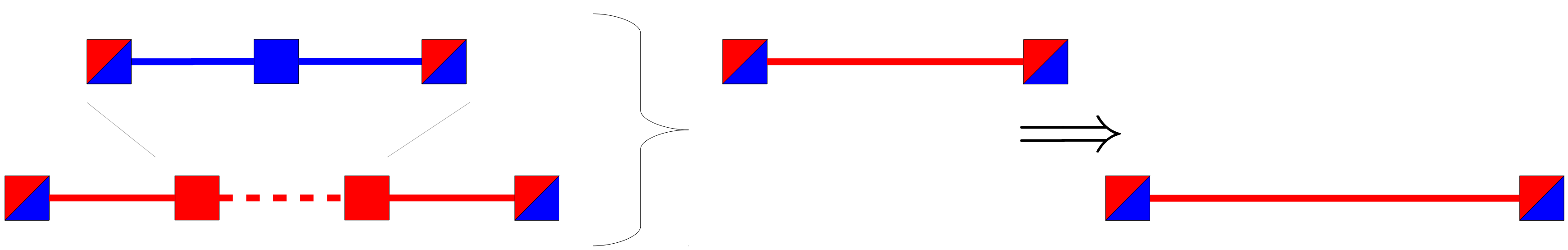


Interpolation algorithm (sketch)

- 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

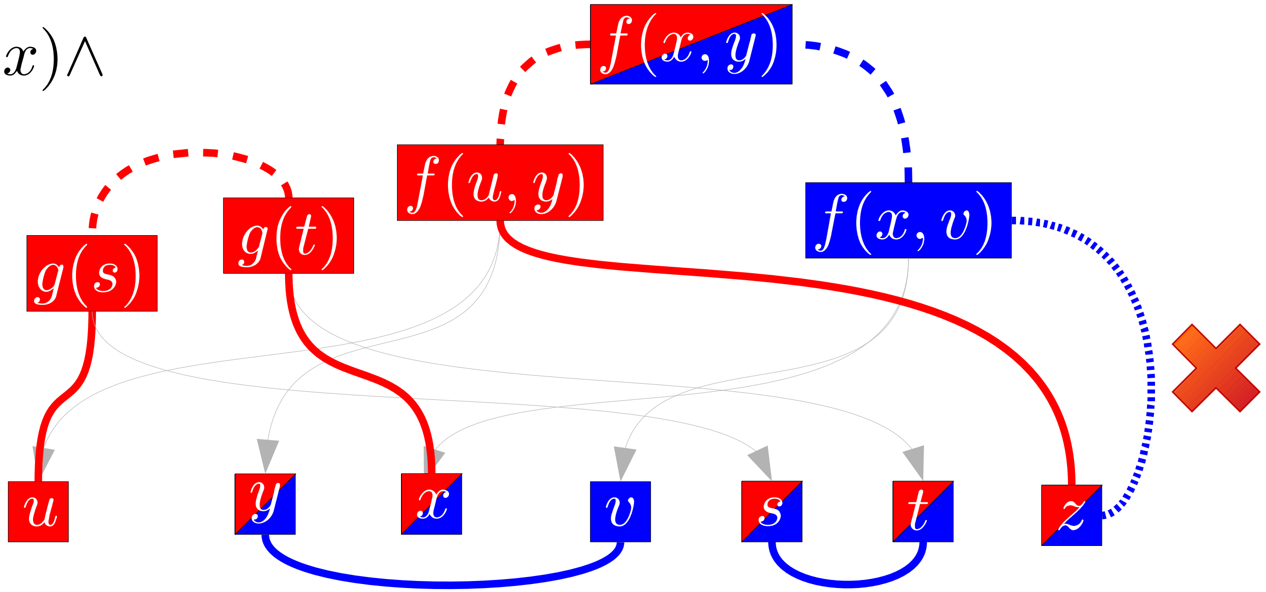


- (Several cases to consider)

Example

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

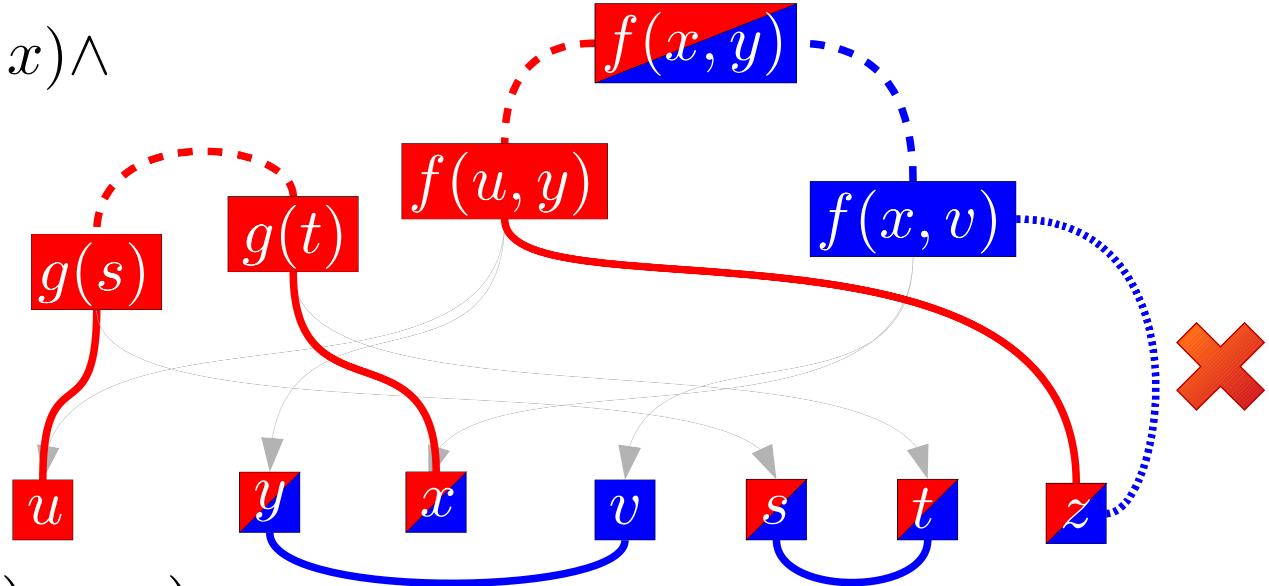
$B := (v = y) \wedge (s = t) \wedge$
$\neg(f(x, v) = z)$



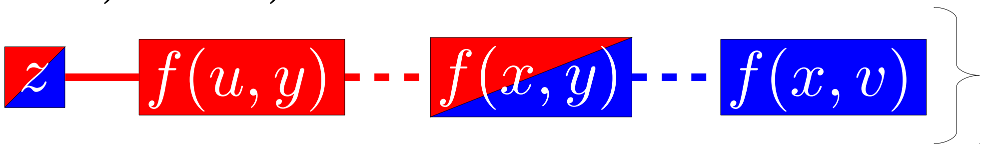
Example

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

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



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

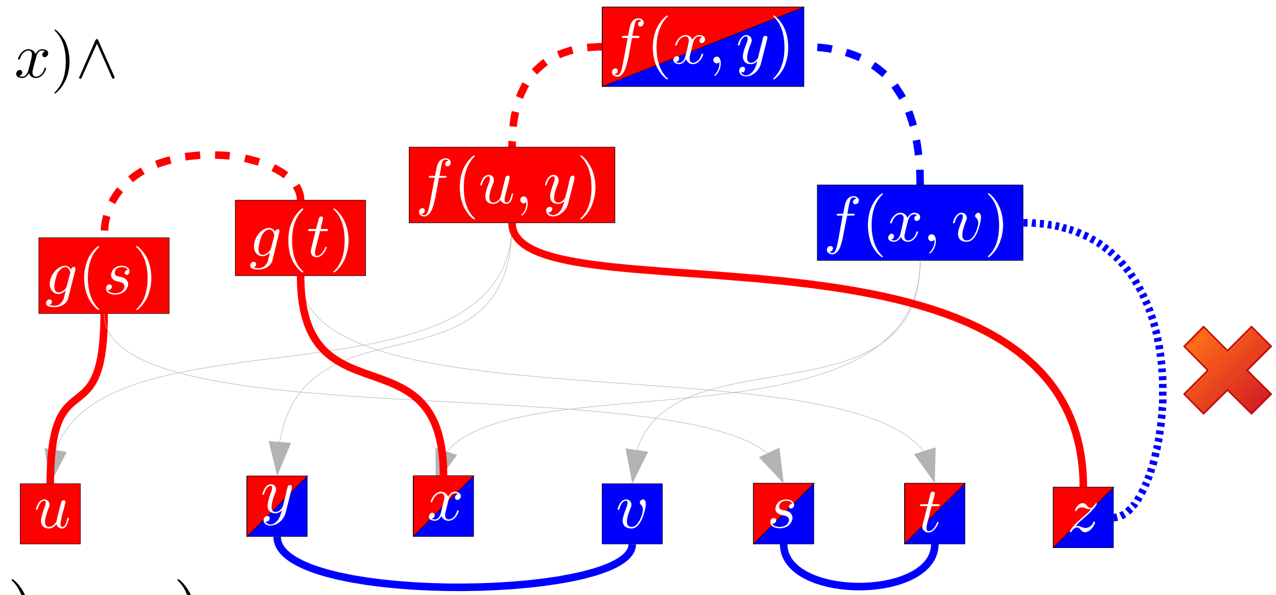


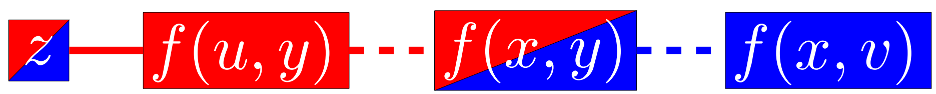
$z = f(x, y)$

Example

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

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



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

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

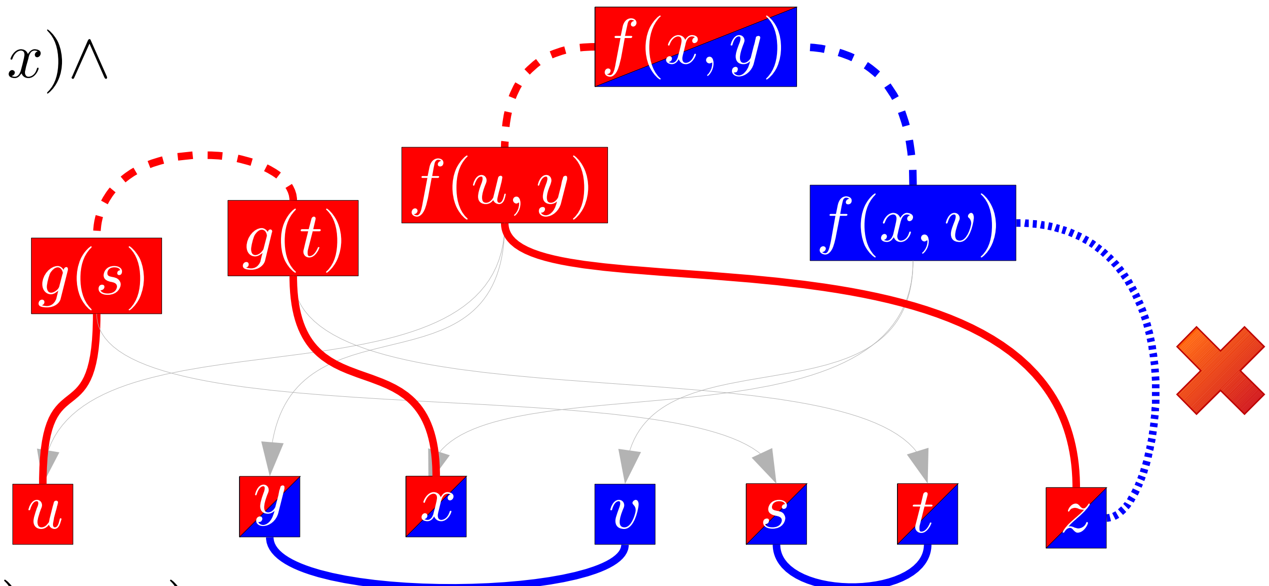
  - Path 

$\} \; \top$

Example

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

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

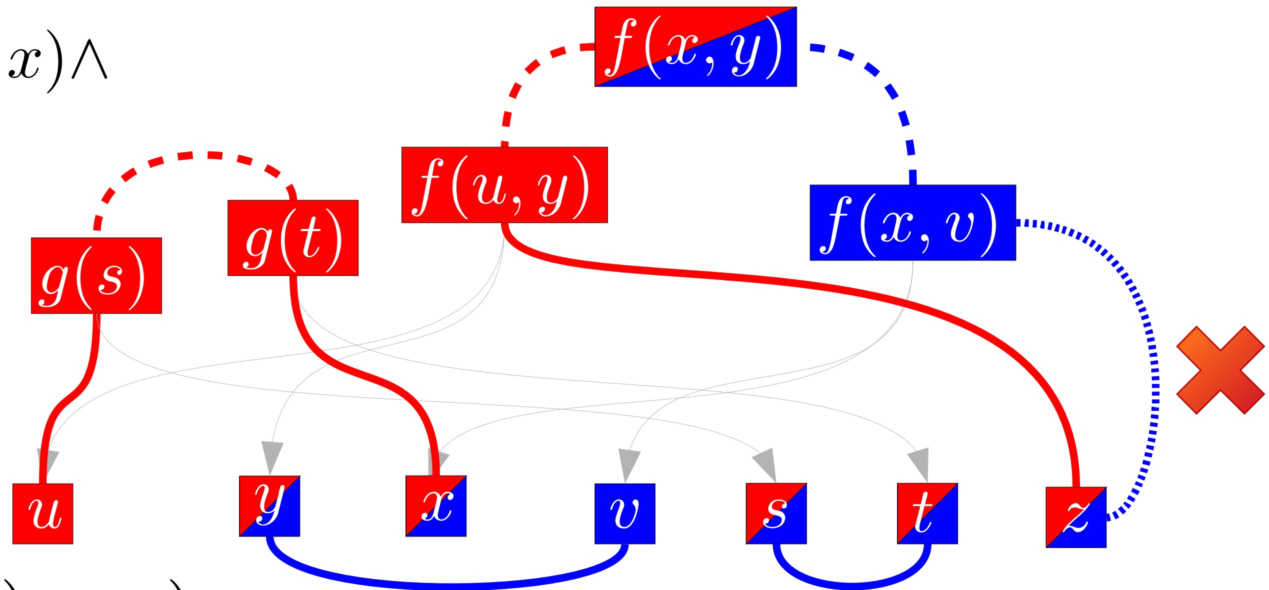


- Start from $\neg(f(x,v) = z)$
- *A*-summaries for z — $f(u,y)$ - - - $f(x,y)$ - - - $f(x,v)$ $\}$ $z = f(x,y)$
- Recurse on edge $f(u,y)$ - - - $f(x,y)$
  - Path u — $g(s)$ - - - $g(t)$ — x $\}$ $\top$
    - Recurse on edge $g(s)$ - - - $g(t)$
      - Path s — t , *B*-summary: $(s = t)$

Example

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

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



- 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

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

tableau

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

bounds

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

candidate solution β

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

Example

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

tableau

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

bounds

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

candidate solution β

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

No suitable variable for pivoting!
Conflict

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 | bounds | candidate solution β

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

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

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

Proof:

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

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

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

Interpolant:

$$\cfrac{\cfrac{\cfrac{-\qquad 1 \cdot (3x_2 - x_1 \leq 1)}{(3x_2 - x_1 \leq 1)} \qquad 3 \cdot (0 \leq x_1 + x_2)}{(-4x_1 \leq 1)} \qquad -}{(-4x_1 \leq 1)}$$

Linear Integer Arithmetic (LIA)

- Constraints of the form

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

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

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

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

Linear Integer Arithmetic (LIA)

- Constraints of the form

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

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

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

LRA rules

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

Linear Integer Arithmetic (LIA)

- Constraints of the form

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

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

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

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

- Interpolation by annotating proof rules with inequalities
  - When $\bot$ is derived, the associated annotation is the computed interpolant

Interpolation with ceilings

- Need to extend the signature of LIA to allow interpolation
  - Introduce the ceiling function $\lceil\cdot\rceil$ [Pudlák '97]
  - Allow non-variable terms to be non-integers (e.g. $\frac{x}{2}$)

Interpolation with ceilings

- Need to extend the signature of LIA to allow interpolation
  - Introduce the ceiling function $\lceil \cdot \rceil$ [Pudlák '97]
  - Allow non-variable terms to be non-integers (e.g. $\frac{x}{2}$)

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

$$\text{Comb}\ \frac{t_1 \leq 0\ [t'_1 \leq 0] \qquad t_2 \leq 0\ [t'_2 \leq 0]}{c_1 \cdot t_1 + c_2 \cdot t_2 \leq 0\ [c_1 \cdot t'_1 + c_2 \cdot t'_2 \leq 0]}$$

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

Interpolation with ceilings - example

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

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

$$\frac{4x - 4z + 1 \leq 0}{4 \cdot (x - z + 1 \leq 0)}$$

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

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

Interpolation with ceilings - example

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

$$\dfrac{\dfrac{\dfrac{y + 4x \leq 0 \quad -y - 4z + 1 \leq 0}{[y + 4x \leq 0] \quad [0 \leq 0]}}{4x - 4z + 1 \leq 0 \quad [y + 2nx \leq 0]}}{4 \cdot (x - z + 1 \leq 0) \quad [x + \lceil \frac{y}{4} \rceil \leq 0]}$$

$$\dfrac{\dfrac{-y - 4x - 1 \leq 0 \quad y + 4z - 2 \leq 0}{[-y - 4x - 1 \leq 0] \quad [0 \leq 0]}}{-4x + 4z - 3 \leq 0 \quad [-y - 4x - 1 \leq 0]}$$

$$\dfrac{4 \cdot (x - z + 1 \leq 0) \; [x + \lceil \frac{y}{4} \rceil \leq 0] \qquad -4x + 4z - 3 \leq 0 \; [-y - 4x - 1 \leq 0]}{(1 \leq 0) \equiv \bot \quad [4\lceil \frac{y}{4} \rceil - y - 1 \leq 0]}$$

Interpolation with ceilings - example

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

$$\dfrac{\dfrac{\dfrac{\begin{matrix} y + 4x \leq 0 & -y - 4z + 1 \leq 0 \\ [y + 4x \leq 0] & [0 \leq 0] \end{matrix}}{\begin{matrix} 4x - 4z + 1 \leq 0 \\ [y + 2nx \leq 0] \end{matrix}}}{\begin{matrix} 4 \cdot (x - z + 1 \leq 0) \\ [x + \lceil \frac{y}{4} \rceil \leq 0] \end{matrix}} \qquad \dfrac{\begin{matrix} -y - 4x - 1 \leq 0 & y + 4z - 2 \leq 0 \\ [-y - 4x - 1 \leq 0] & [0 \leq 0] \end{matrix}}{\begin{matrix} -4x + 4z - 3 \leq 0 \\ [-y - 4x - 1 \leq 0] \end{matrix}}}{(1 \leq 0) \equiv \bot}$$

Interpolant: $[4\lceil \frac{y}{4} \rceil - y - 1 \leq 0]$

SMT(LIA) with ceilings

- 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 $b_1^x \dots b_n^x$
  - 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:

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

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

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

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

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

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

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

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

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

$\wedge \quad \wedge$

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

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

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 $(A \land \neg\hat{I}) \lor (B \land \hat{I})$
  - 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:

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

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

From LIA- to BV-interpolants: examples

$$A \stackrel{\text{def}}{=} (y_{1[8]} = y_{5[4]} :: y_{5[4]}) \land (y_{1[8]} = y_{2[8]}) \land (y_{5[4]} = 1_{[4]})$$

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

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

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

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

LIA-interpolant:

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

BV-interpolant:

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

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:

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

Wrong!

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

From LIA- to BV-interpolants: examples

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

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

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:

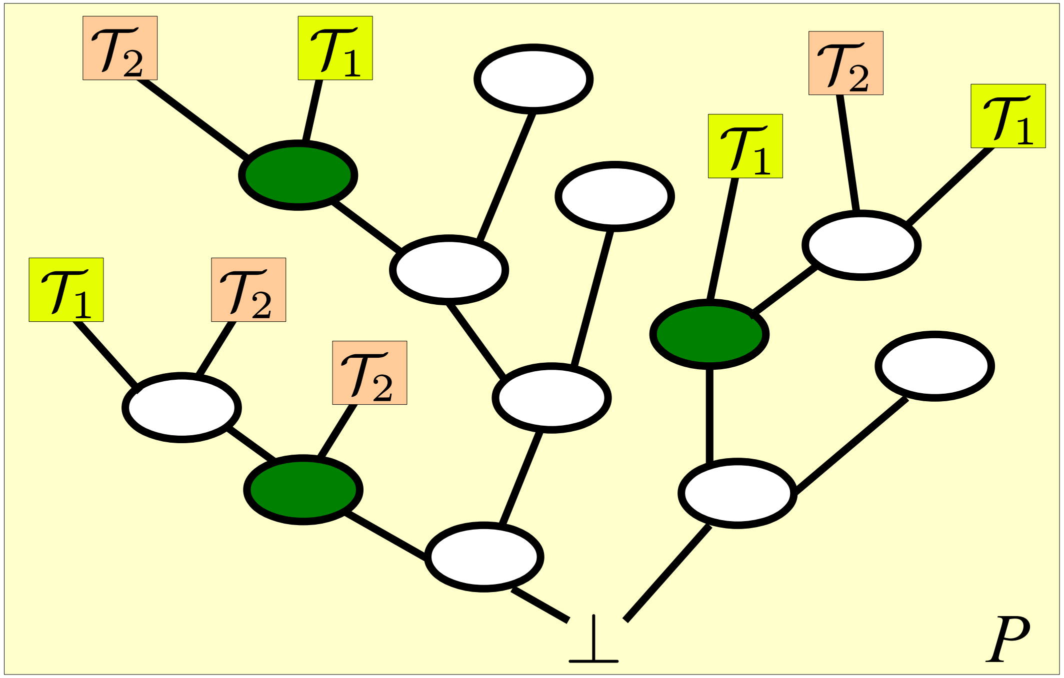
$$I \stackrel{\text{def}}{=} (65281_{[16]} \leq_s (0_{[8]} :: {y_2}_{[8]}) - (0_{[8]} :: {y_3}_{[8]}) + 256_{[16]} \cdot (65535_{[16]} \cdot (0_{[8]} :: {y_2}_{[8]}) /_u 256_{[16]}))$$

Correct interpolant

Interpolation in combined theories

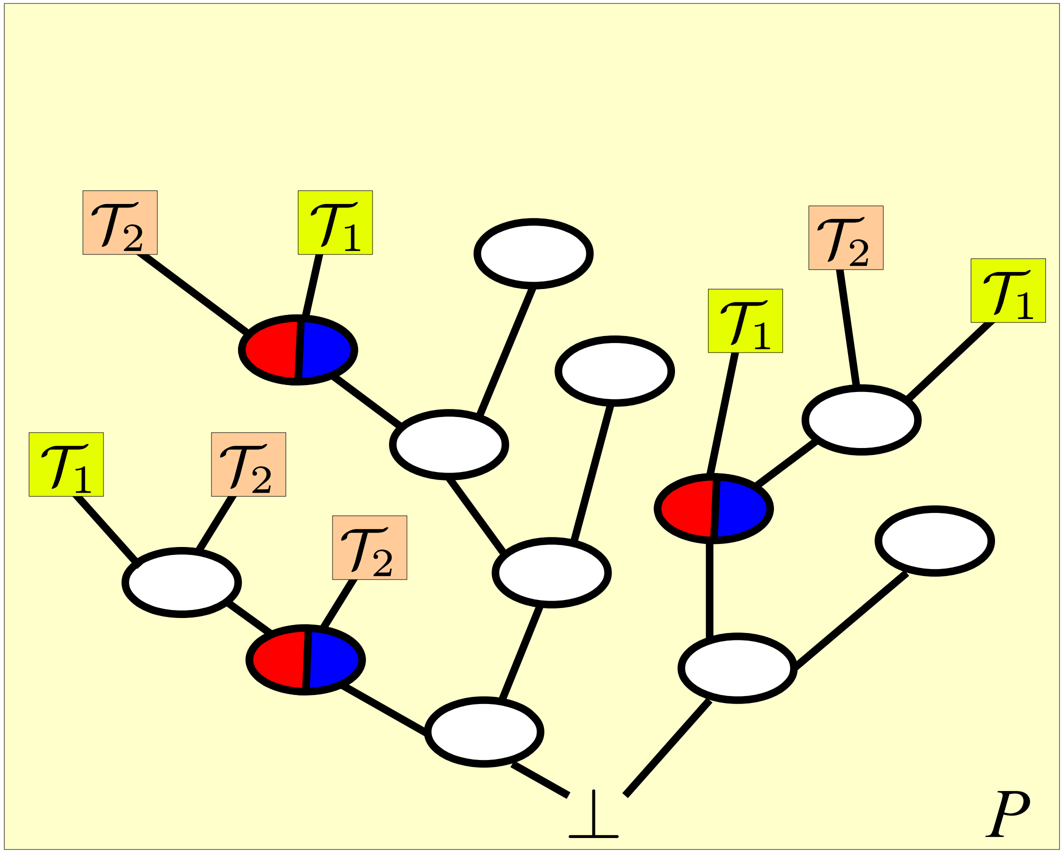
- Delayed Theory Combination (DTC): use the DPLL engine to perform theory combination
  - Independent $\mathcal{T}_i$-solvers, that interact only with DPLL
  - How: Boolean search space augmented with interface equalities
    - Equalities between variables shared by the two theories

- Combination of theories encoded directly in the proof of unsatisfiability P
  - $\mathcal{T}_i$-lemmas for the individual theories
  - $\boldsymbol{P}$ contains interface equalities



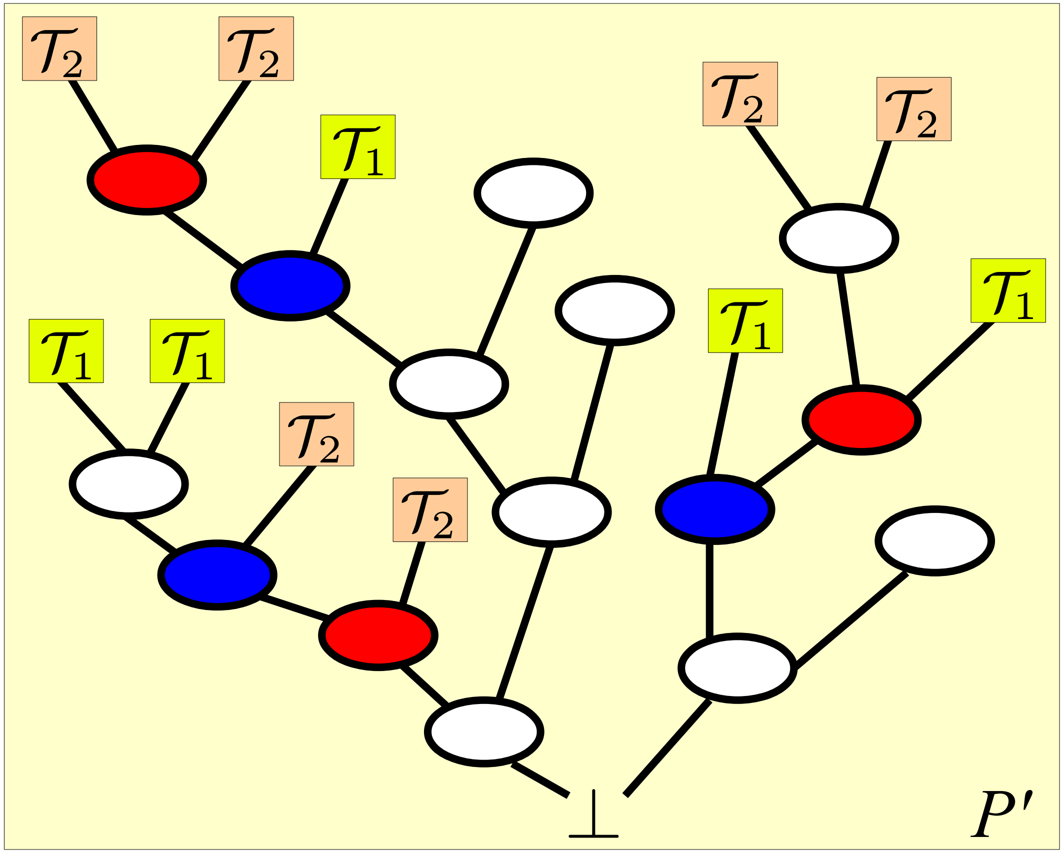
Interpolation in combined theories

- 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

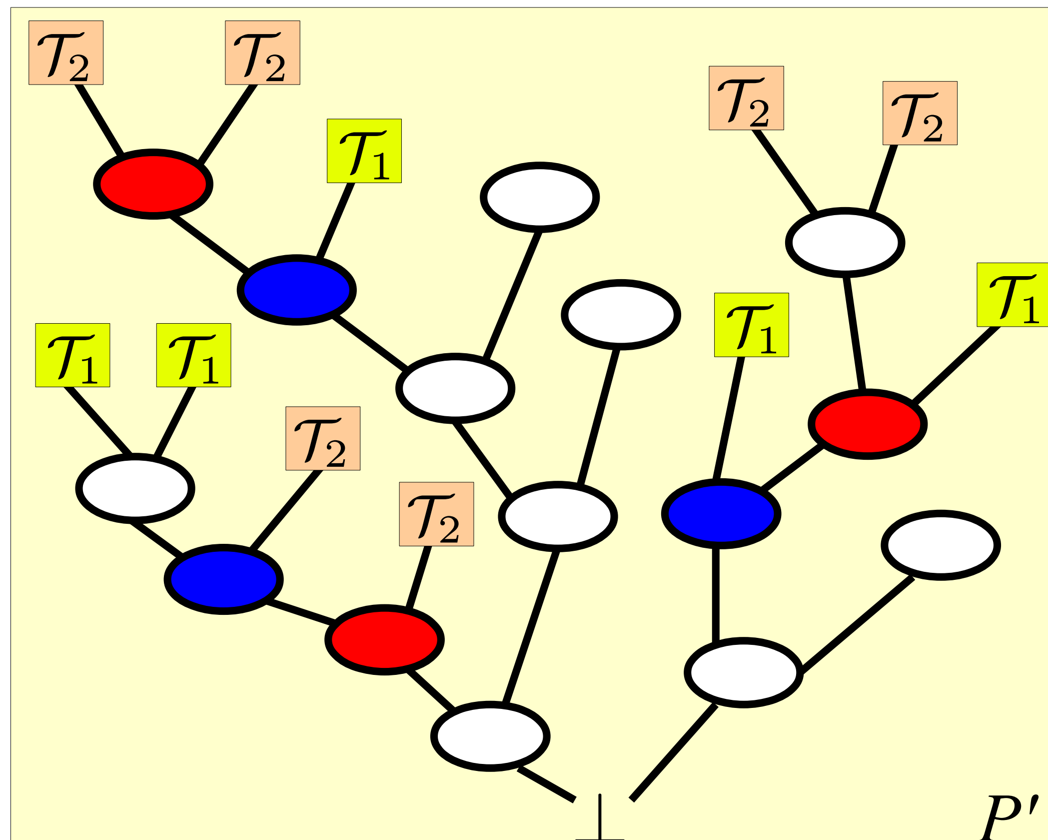


Interpolation in combined theories

- 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

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

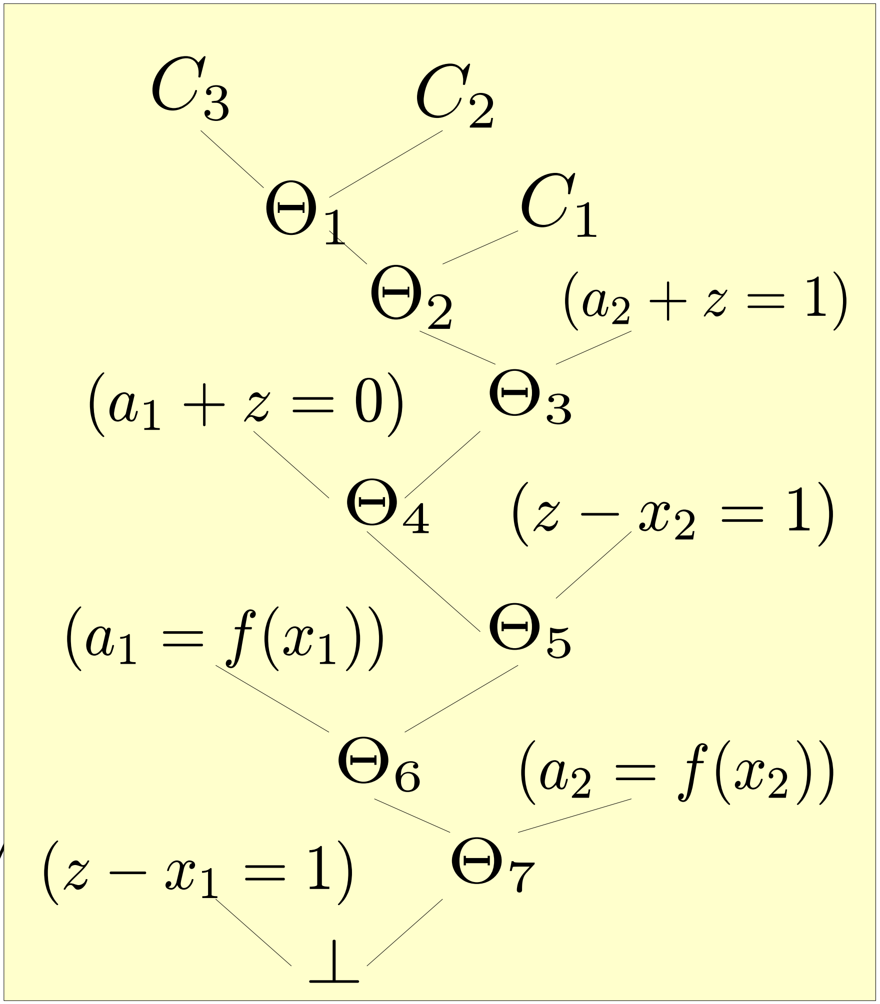
$$B := (a_2 = f(x_2)) \wedge (z - x_2 = 1) \wedge (a_2 + z = 1)$$

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: $(x_1 = x_2)$

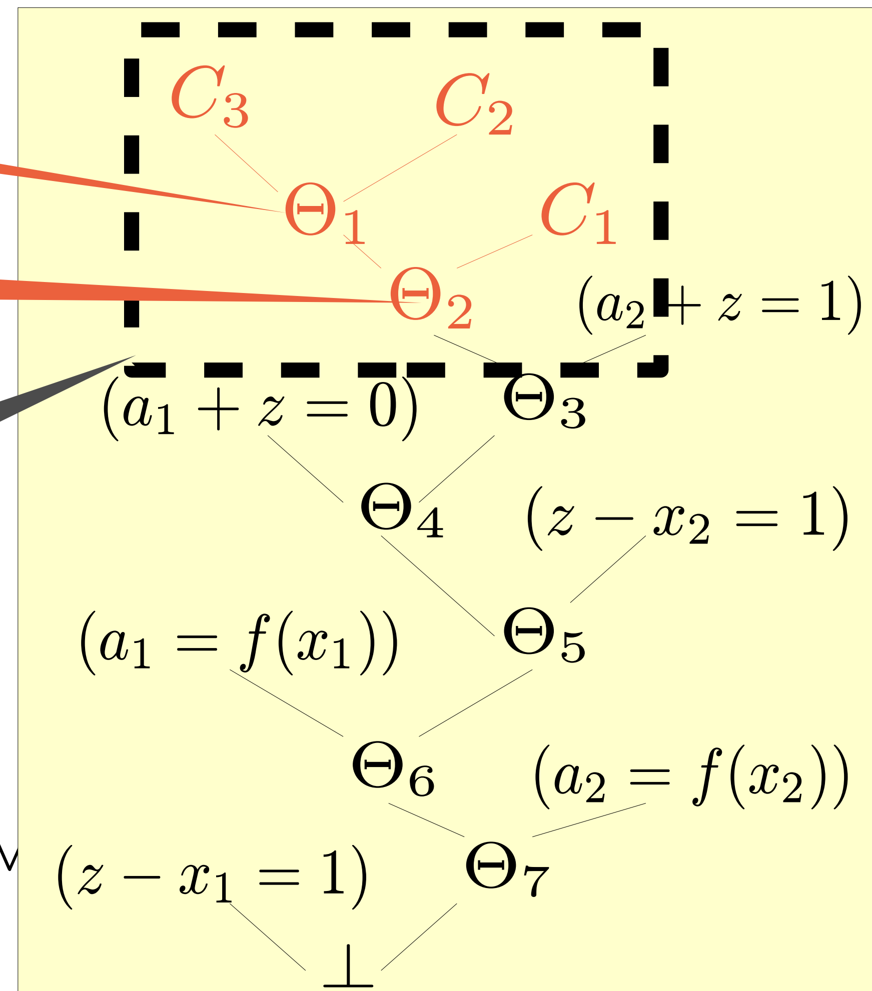
subproof with int.eqs.

T-lemmas:

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

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

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

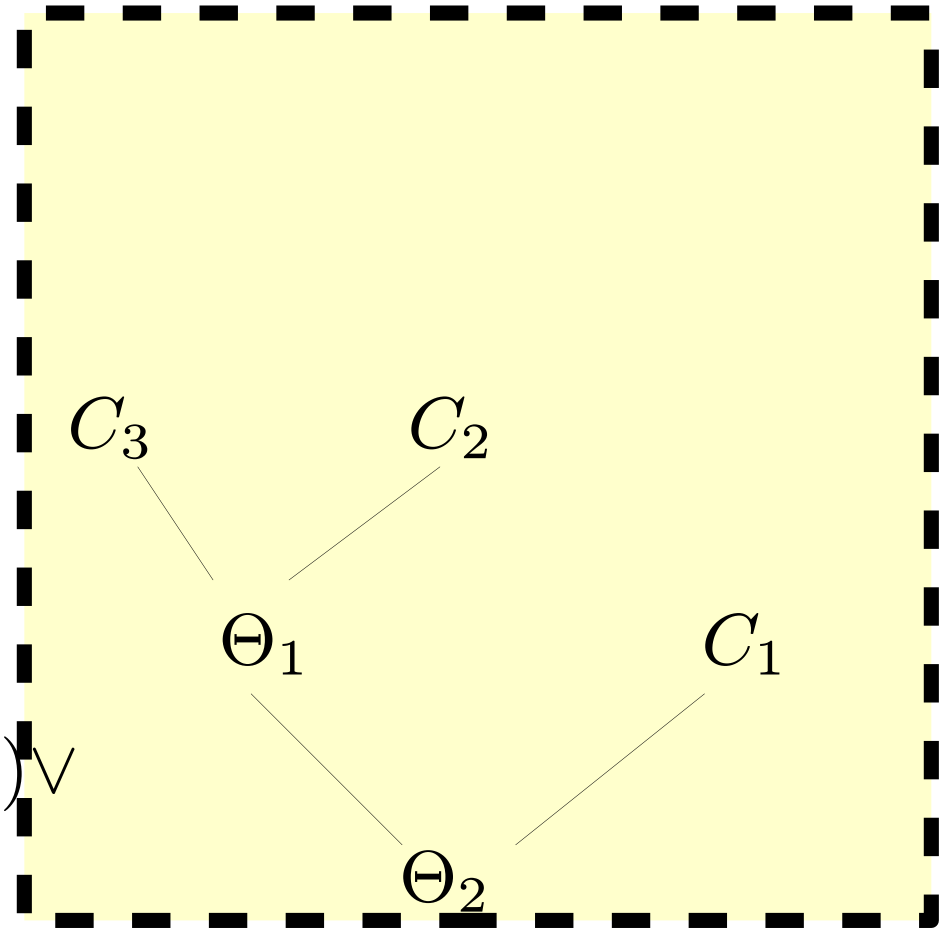


Example

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

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

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

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

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

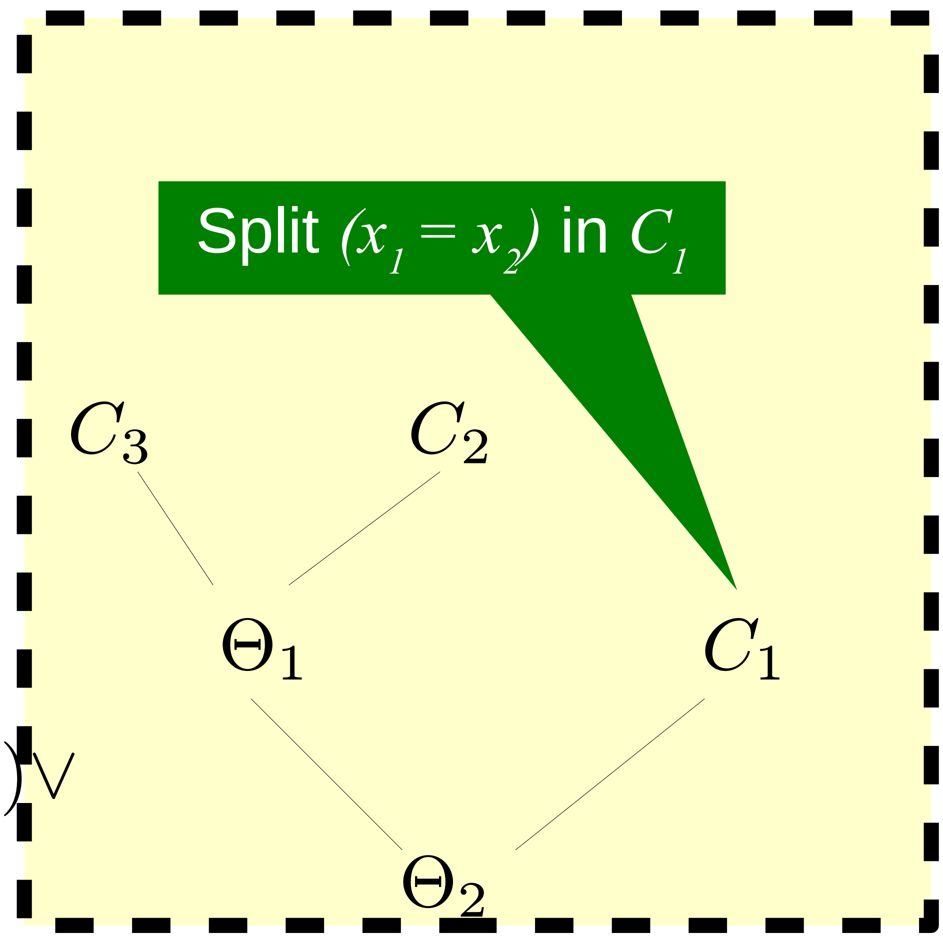
P^{ie} subproof:

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

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

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

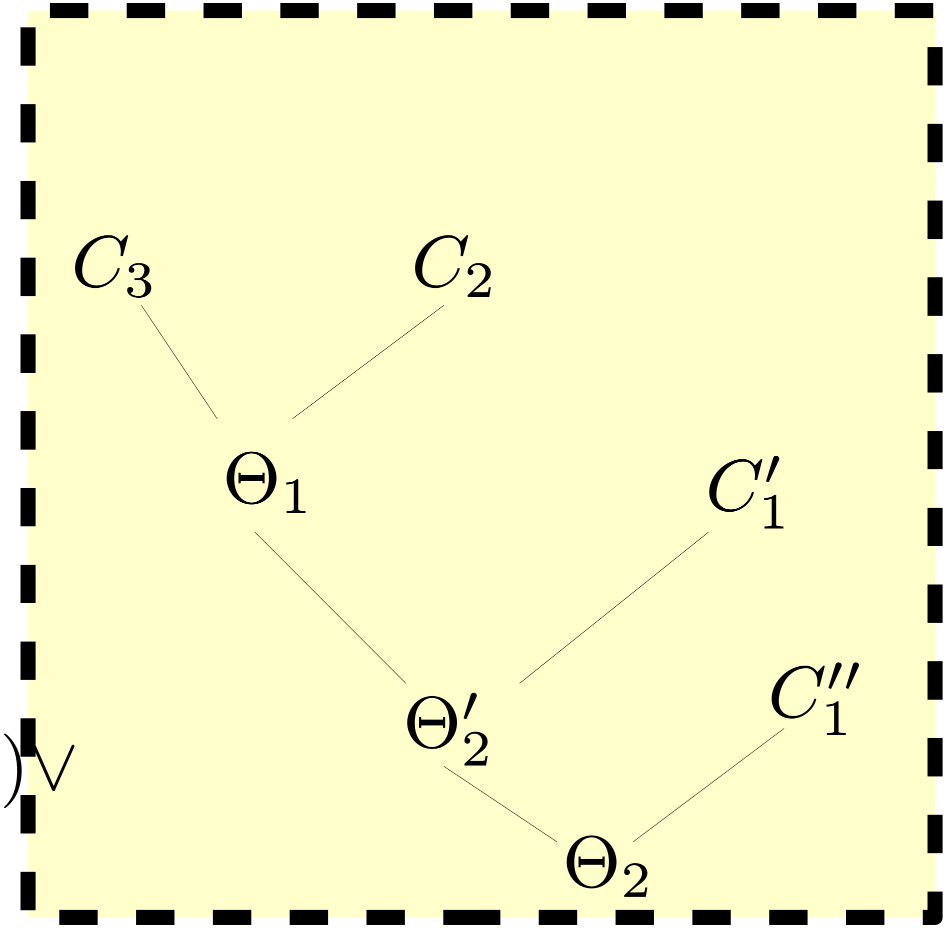
P^{ie} subproof:

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

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

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

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



Example

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

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

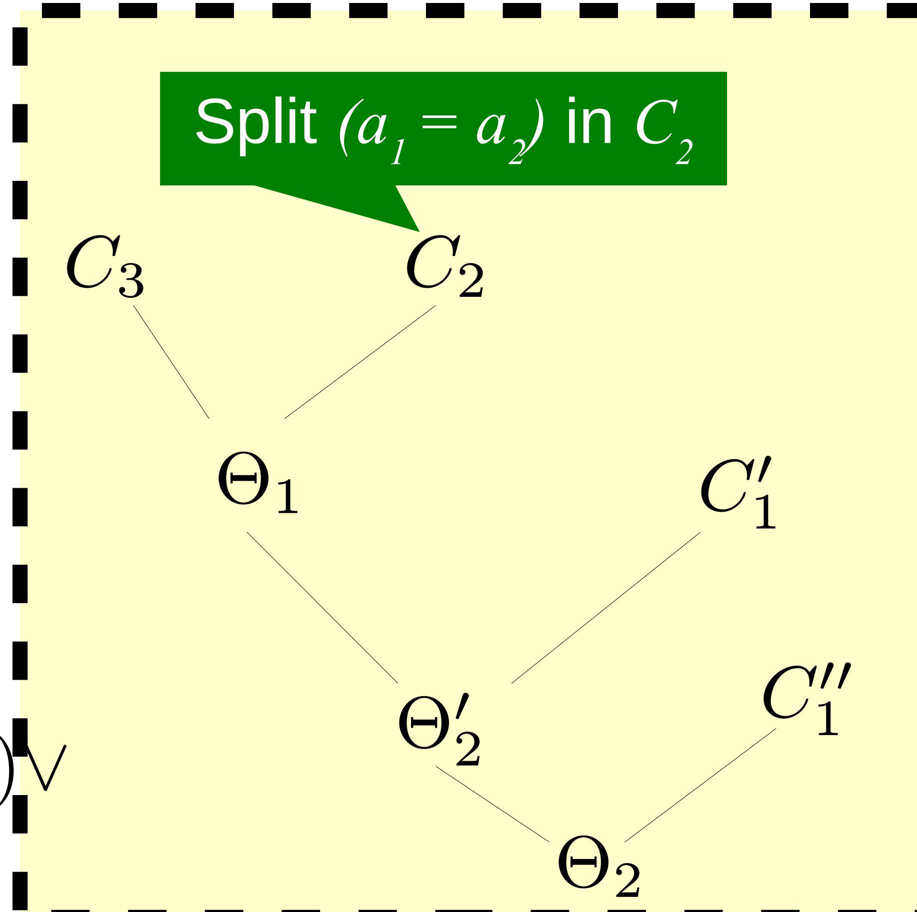
P^{ie} subproof:

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

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

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

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



Example

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

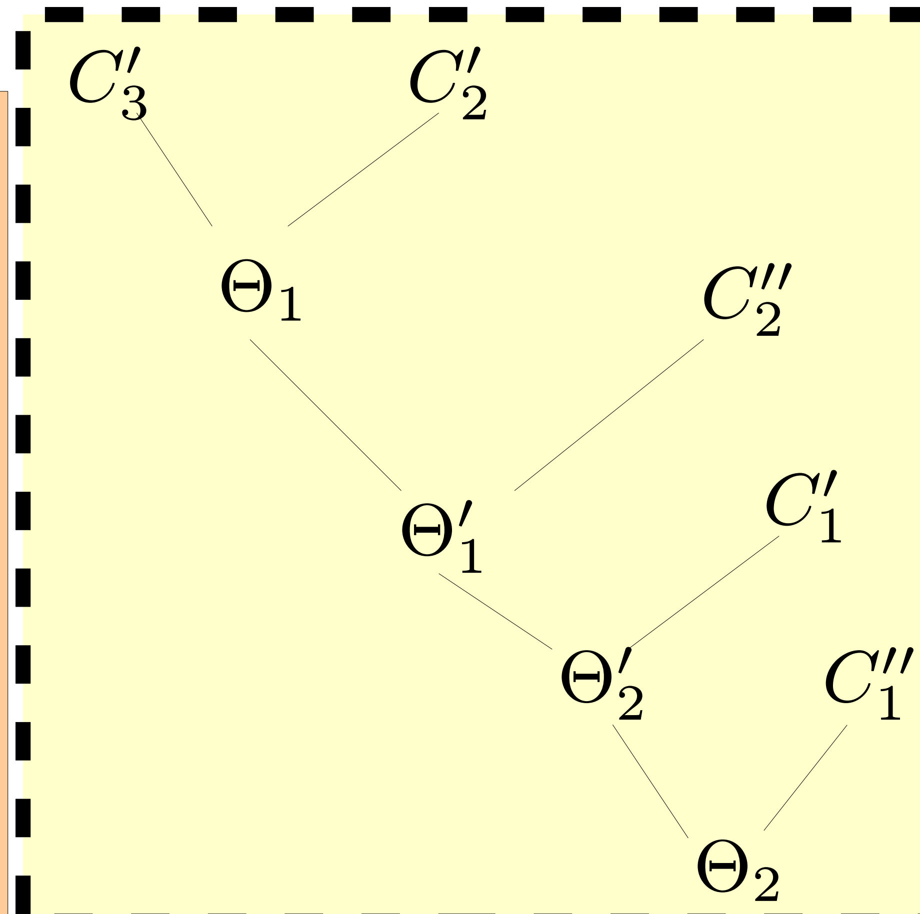
$$B := (a_2 = f(x_2)) \wedge (z - x_2 = 1) \wedge (a_2 + z = 1)$$

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 $(s \leq t)$

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

- 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 $I_k := \text{Interpolant}(\bigwedge_{i=1}^{k} F_i, \bigwedge_{j=k+1}^{n} F_j)$ 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:

$$I' \wedge F_2 \models I'' \vee \bigvee \{l \in C \mid \text{var}(l) \notin F_3\}$$

  - 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*

- Quantifier Elimination in SMT

  - Monniaux. **A Quantifier Elimination Algorithm for Linear Real Arithmetic**. LPAR 2008
  - Bjørner. **Linear Quantifier Elimination as an Abstract Decision Procedure**. IJCAR 2010
  - Komuravelli, Gurfinkel, Chaki. **SMT-based model checking for recursive programs**. FMSD 48(3) 2016

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

Selected bibliography

- 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

Thank You