Craig Interpolation in SAT and SMT

Philipp Rümmer
Uppsala University
Philipp.Ruemmer@it.uu.se

July 12, 2014
SAT/SMT Summer School

Outline

- General introduction
- Computation of interpolants
- Generalised forms of interpolation
- Controlling interpolation

Basics, Application

Bit of history

- W. Craig (1957a), Linear reasoning. A new form of the Herbrand-Gentzen theorem
  ________ (1957b), Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory
- P. D. Bacsich. 1975. Amalgamation properties and interpolation theorems for equational theories. Algebra Universalis 5 (1975), 45–55.

Bit of history (2)

- D. Mundici, A lower bound for the complexity of Craig's interpolants in sentential logic. Archiv. Math. Logik 23 (1983) 27–36.
- J. Krajcek. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. The Journal of Symbolic Logic, 62(2):457–486, 1997.
- P. Pudlak. Lower bounds for resolution and cutting plane proofs and monotone a computations. The Journal of Symbolic Logic, 62(3):981–998, 1997.

Bit of history (3)

- Kenneth L. McMillan: Interpolation and SAT-Based Model Checking. CAV 2003: 1-13
- Kenneth L. McMillan: An interpolating theorem prover. Theor. Comput. Sci. 345(1): 101-121 (2005)

Some basic notions

- Signatures Σ
    - Functions, predicates, constants
- First-order formulae; clauses
    - $\top, \bot, \wedge, \vee, \neg, \rightarrow, \forall, \exists, =, p(\bar{t}), f(\bar{t})$
- Model semantics
- Entailment $\models$
- Theories $T = (\Sigma_T, Ax_T)$
    - Interpreted symbols Σ_T, axioms Ax_T

Binary interpolants

Definition
Suppose an implication $A \to C$ in some logic.
A *Craig interpolant* is a formula I such that
- $A \to I$ and $I \to C$ are valid, and
- every non-logical symbol of I occurs in both A and C.

- "Non-logical" symbols: variables, uninterpreted functions, etc.
- Clearly, if I exists, then the implication $A \to C$ is valid
- Converse?

Craig's theorem

Theorem [Craig, 1957]
Suppose $A \to C$ is a valid (closed) implication in first-order logic.
Then there is a Craig interpolant I for $A \to C$.

- Same result for many fragments:
    - Propositional formulae
    - Quantifier-free first-order formulae

Example

$$\underbrace{\big(f(a) = b \wedge p(f(a))\big)}_{A} \rightarrow \underbrace{(b = c \rightarrow p(c))}_{C}$$

Local symbols in A: f, a
Local symbols in C: c
Global/shared symbols: p, b

Interpolant: $p(b)$

Reverse interpolants

Definition
Suppose a conjunction $A \wedge B$ is given.
A *reverse interpolant* is a formula I such that
- $A \to I$ and $B \to \neg I$ are valid, and
- every non-logical symbol of I occurs in both A and B.

- Reverse interpolants for $A \wedge B$
  =
  Ordinary interpolants for $A \to \neg B$
- From now on:
  interpolant := reverse interpolant

Craig's theorem (2)

Existence of an interpolant if conjunction $A \wedge B$ is unsatisfiable.

Proof sketch (constructive):

1. Clausify A, B

2. By Herbrand + compactness, there are finite sets A', B' of ground instances such that $A' \wedge B'$ is unsat.

3. Compute ground interpolant I' for $A' \wedge B'$ (→ later)

4. Abstract by introducing quantifiers → interpolant I

Applications

- *Safety for finite-state systems:*
  Transition system: $I(\bar{s}), T(\bar{s}, \bar{s}')$
  Property: $P(\bar{s})$
- Bounded model checking:

$$I(\bar{s}_0) \wedge \neg P(\bar{s}_0) \quad ?$$

$$I(\bar{s}_0) \wedge T(\bar{s}_0, \bar{s}_1) \wedge \neg P(\bar{s}_1) \quad ?$$

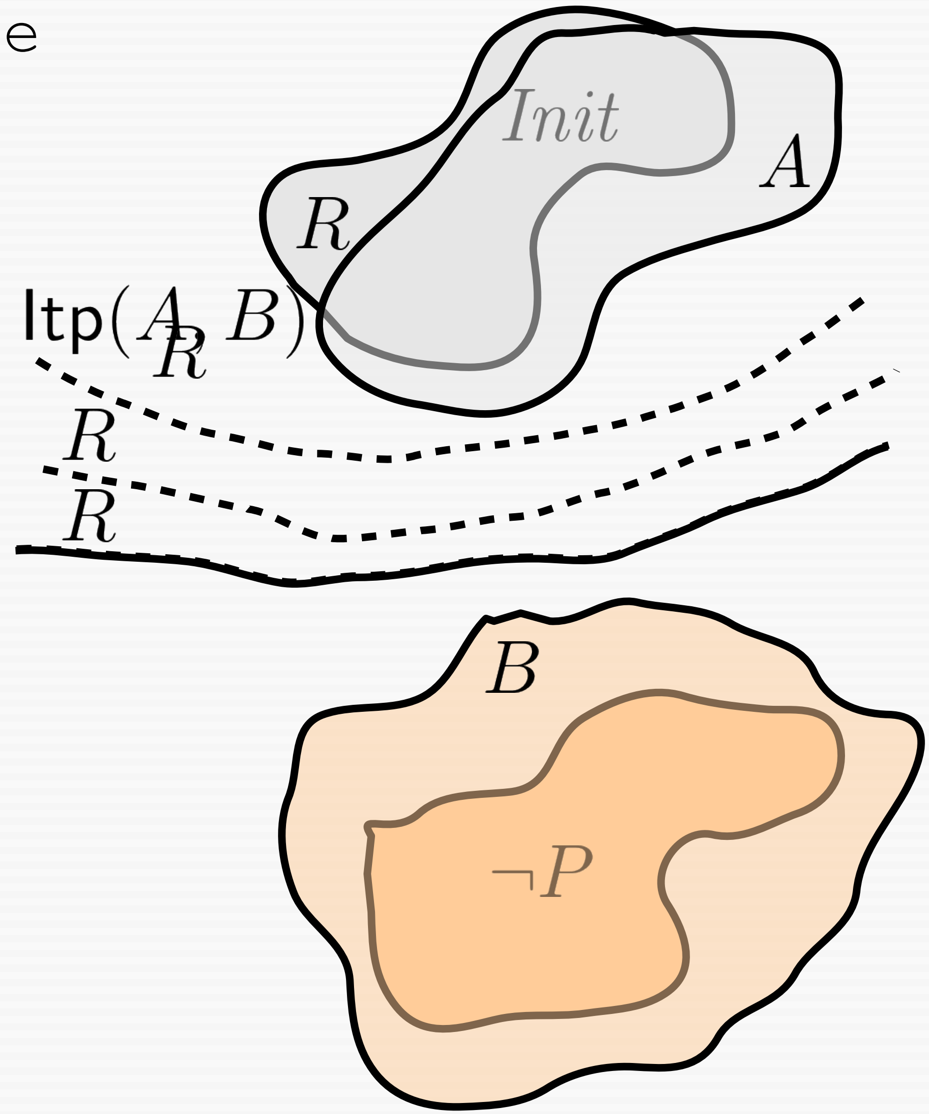
$$I(\bar{s}_0) \wedge T(\bar{s}_0, \bar{s}_1) \wedge T(\bar{s}_1, \bar{s}_2) \wedge \neg P(\bar{s}_2) \quad ?$$

$$\vdots$$

Interpolation-based MC (simp.)

[McMillan, 2003]

If $Init(\bar{s}) \wedge \neg P(\bar{s})$ is satisfiable
    return Unsafe
$R = Init(\bar{s}_{-1})$
while (**true**) {
    $A = R \wedge T(\bar{s}_{-1}, \bar{s})$
    $B = T(\bar{s}, \bar{s}_1) \wedge (\neg P(\bar{s}) \vee \neg P(\bar{s}_1))$
    if $A \wedge B$ is satisfiable {
        return Unknown
    } **else** {
        $R' = R \vee \mathsf{Itp}(A, B)[\bar{s}/\bar{s}_{-1}]$
        if $R == R'$
            return Safe
        else
            $R = R'$
    }
}



Further applications

- Beth's theorem
- Synthesis
  - Database queries
  - Programs

Theories

- Depending on domain, interpolation queries are often formulated over some theories

General theory interpolation

Theorem [Kovacs, Voronkov, 2009]
Suppose $A \wedge B$ is an unsatisfiable conjunction in first-order logic modulo T.
Then there is an interpolant I such that
- $A \models_T I$
- $B \models \neg I$
- every non-logical symbol of I occurs in both A and B.

- Symmetric case possible: $A \models I, \; B \models_T \neg I$
- However: even if A and B are quantifier-free, I might contain quant.

Plain quantifier-free interpolation

Definition [Bruttomesso, Ghilardi, Ranise, 2013]
A theory $T = (\Sigma_T, Ax_T)$ admits *plain quantifier-free* interpolation if for every quantifier-free T-unsat. conjunction $A \wedge B$ over Σ_T (with arbitrary free variables) there is a quantifier-free I such that

- $A \models_T I$
- $B \models_T \neg I$
- I only contains variables common to A and B.

Properties

- Theories that admit **quantifier elimination** also admit plain quantifier-free interpolation. E.g.
    - Presburger arithmetic
    - Real arithmetic
- Strongest interpolant: $\exists \bar{x}.\ A[\bar{x}]$
  Weakest interpolant: $\forall \bar{y}.\ \neg B[\bar{y}]$
- Converse?
    - E.g., EUF

General quantifier-free interpol.

Definition [Bruttomesso, Ghilardi, Ranise, 2013]
A theory $T = (\Sigma_T, Ax_T)$ admits ***general*** *quantifier-free* interpolation if for every **closed q-f** T-unsat. conjunction $A \wedge B$ over $\Sigma_T \cup \Sigma$ there is a quantifier-free I such that

- $A \models_T I$
- $B \models_T \neg I$
- I only contains Σ-symbols common to A and B.

Plain vs. general

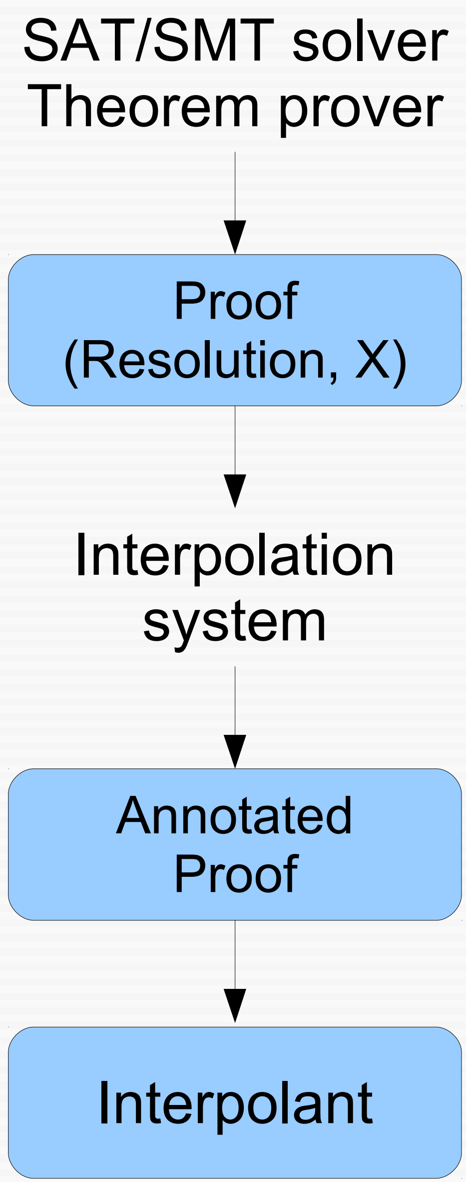
- General q-f interpolation is **strictly stronger** a requirement than plain q-f interpolation
- E.g., Presburger arithmetic
  - Division/modulo operator needed for general q-f interpolation

Computation of Interpolants

Interpolation paradigms

- Extraction from proofs
- Constraint-based
- Beautiful

Interpolants from proofs



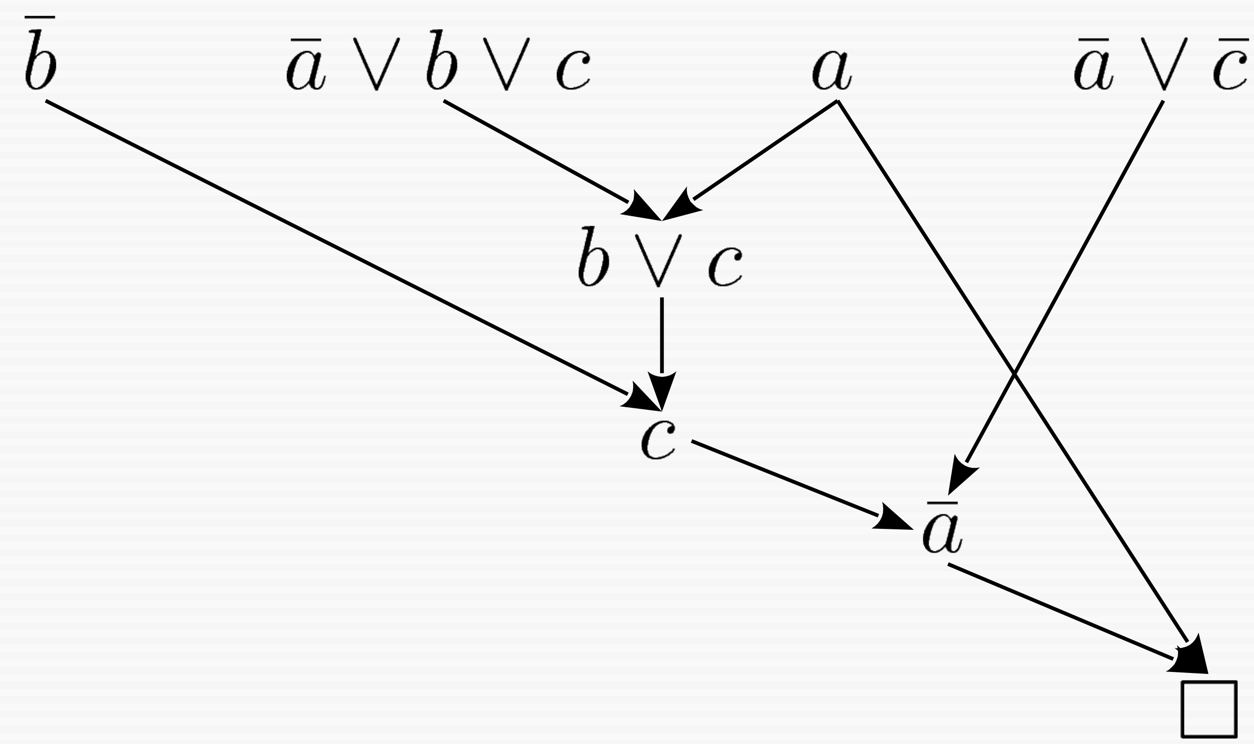
Propositional resolution

Definition
A resolution proof for a set of clauses $\mathcal{C}$ is a directed acyclic graph (V, E), where V is a set of clauses, such that
- the root is the empty clause;
- for every $c \in V$, either
  - $c \in \mathcal{C}$, and c is a leaf; or
  - c is resolvent of exactly two parents c_1 and c_2.

$$\frac{}{c} \; c \in \mathcal{C} \qquad\qquad \frac{v \vee c \qquad \neg v \vee d}{c \vee d}$$

Example

$$\mathcal{C} = \{\bar{b}, \bar{a} \vee b \vee c, a, \bar{a} \vee \bar{c}\}$$



Interpolants from proofs
[McMillan, 2003]

- Suppose sets of clauses A, B, and a resolution proof for $A \cup B$
- Annotate clauses in the proof with *partial interpolants* (Boolean formulae):

$$c \quad [p_c]$$

Augmented proof rules
[McMillan, 2003]

- For a clause c, write $g(c)$ for the sub-clause with only global variables

$$\frac{}{c \quad [g(c)]}\; c \in A \qquad\qquad \frac{}{c \quad [\top]}\; c \in B$$

$$\frac{v \vee c \quad [I_1] \qquad \neg v \vee d \quad [I_2]}{c \vee d \quad [I_1 \vee I_2]}\; v \text{ local to } A$$

$$\frac{v \vee c \quad [I_1] \qquad \neg v \vee d \quad [I_2]}{c \vee d \quad [I_1 \wedge I_2]}\; v \text{ not local to } A$$

Example (2)

$$A = \{\bar{b}, \bar{a} \vee b \vee c, a\} \qquad B = \{\bar{a} \vee \bar{c}\}$$

$$\begin{tikzcd}
\bar{b}\ \ [\bot] \arrow[ddr] & \bar{a} \vee b \vee c\ \ [\bar{a} \vee c] \arrow[dr] & & a\ \ [a] \arrow[dl] \arrow[ddddr] & \bar{a} \vee \bar{c}\ \ [\top] \arrow[dddll] \\
& & b \vee c\ \ [a \wedge (\bar{a} \vee c)] \arrow[dl] & & \\
& c\ \ [\bot \vee (a \wedge c)] \arrow[dr] & & & \\
& & \bar{a}\ \ [(a \wedge c) \wedge \top] \arrow[drr] & & \\
& & & & \square\ \ [a \wedge c]
\end{tikzcd}$$

Correctness

Lemma
In an annotated proof for A, B, for every node $c \quad [p_c]$
it is the case that

$$A \models p_c \vee (c \setminus g(c))$$
$$B, p_c \models g(c)$$
$$p_c \text{ only contains global symbols}$$

- In particular, in root $c = \Box$!

Integration of theories (lazy)
[McMillan, 2005], [Cimatti et al, 2010]

- *Central idea:* add theory-specific partial interpolants for theory lemmas

$$\frac{}{\phi \quad [I_\phi]} \ \phi \text{ is } T\text{-valid}$$

where again

$$\neg(\phi \setminus g(\phi)) \models I_\phi$$
$$\neg g(\phi), I_\phi \models \bot$$
$$I_\phi \text{ only contains global symbols}$$

→ Interpolating theory solvers

- Linear arithmetic
- EUF

Well understood

- Bit-vectors
- Arrays
- Nonlinear arithmetic

Challenging

- *etc.*

Similar interpolation systems

- Sequent proofs for linear rational arithmetic, uninterpreted functions
  [McMillan, 2005]
- First-order resolution
  [McMillan, 2008], [Kovacs, Voronkov, 2009]
- Tableaux, Gentzen-style systems
  [Maehara, 1961], [Fitting, 1996],
  [Brillout et al, 2010]
- *etc.*

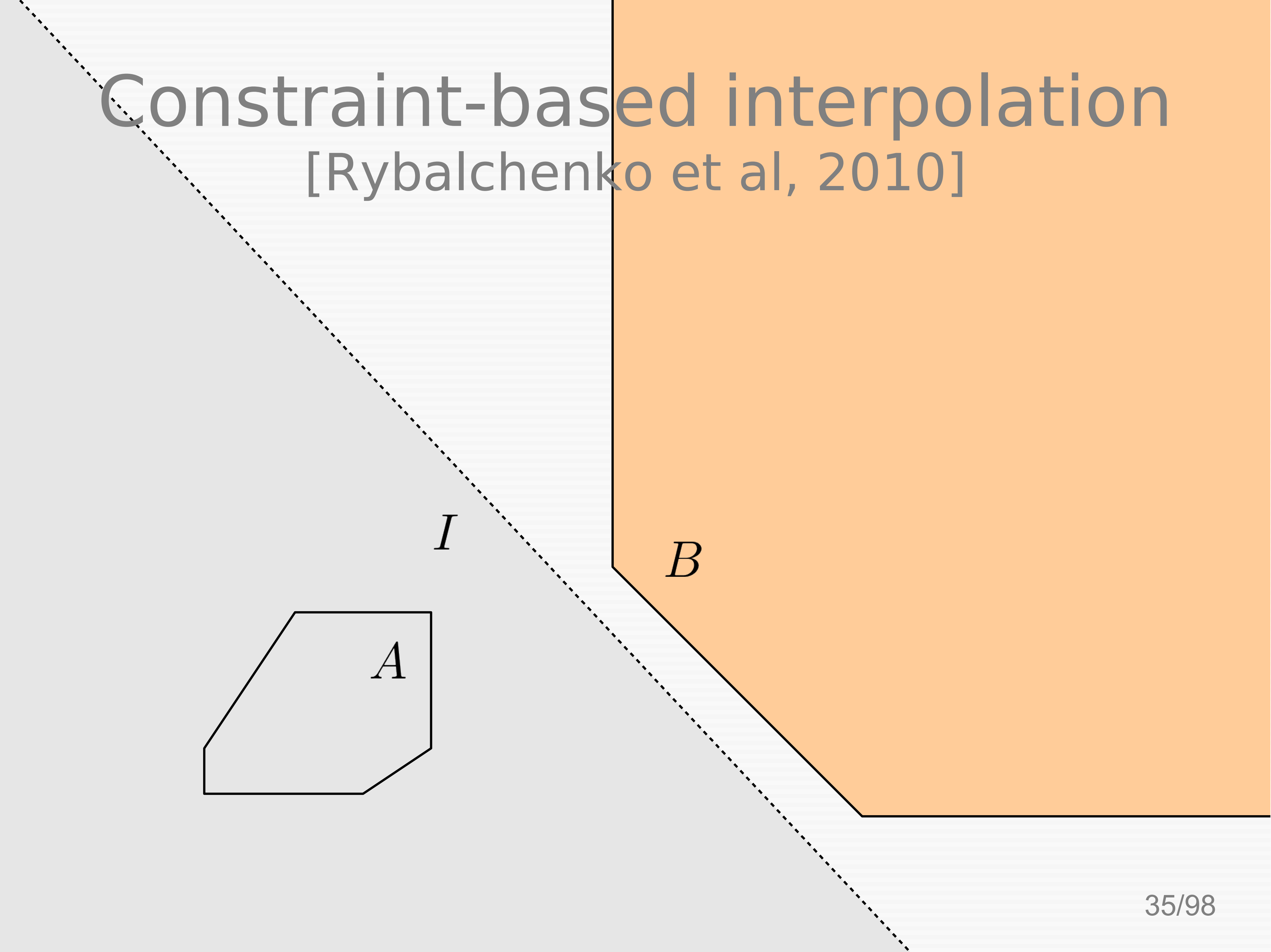
Feasible interpolation

Definition
A proof system has the *feasible interpolation property* if interpolants can be extracted in polynomial time (in the size of a proof).

- Example:
  Cutting plane proofs for integers (Presburger arithmetic):
  - Again, need division/modulo for feasible interpolation property

Constraint-based interpolation

[Rybalchenko et al, 2010]



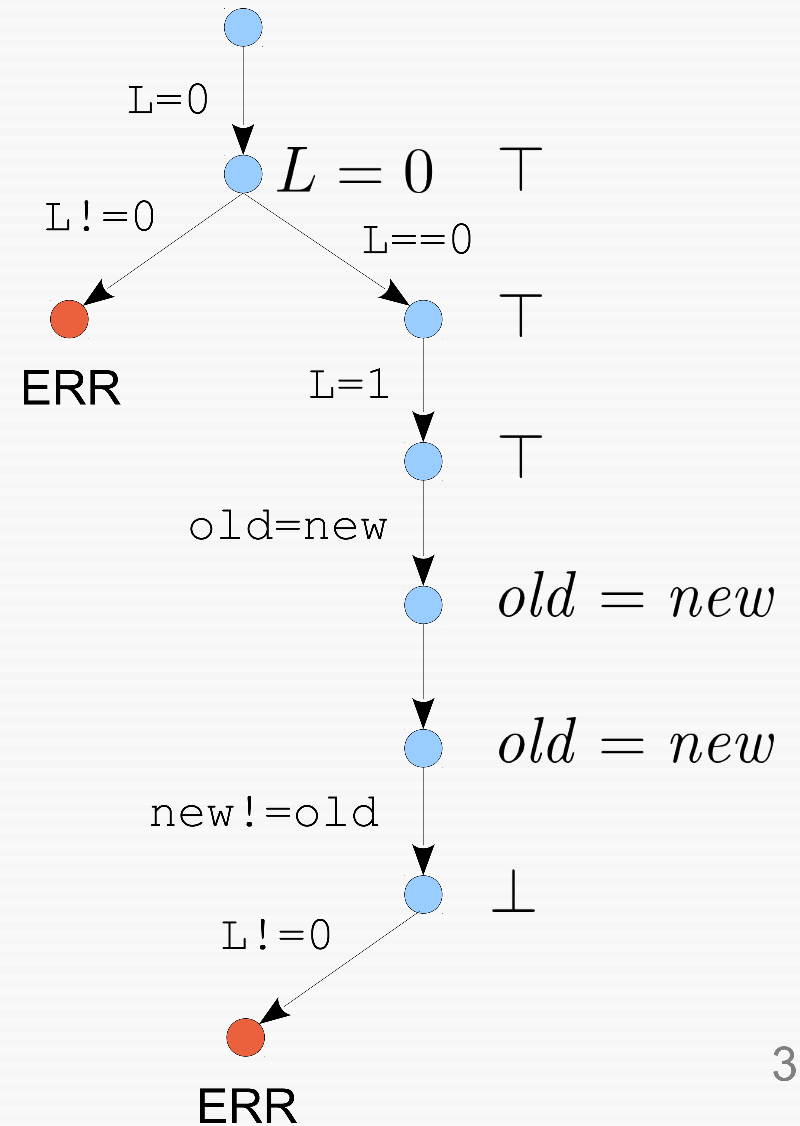
Generalising Interpolants

Motivation

- “Binary” interpolants are often insufficient for applications
- More general forms include
  - Interpolant sequences
  - Tree interpolants
- Computation
  - by repeated binary interpolation; or
  - extraction from a single proof

Software model checking
[McMillan, 2006]

```
L = 0;
do {
   assert(L==0);     } lock()
   L = 1;
   old = new;
   if (*){
      L = 0;         } unlock()
      new++;
   }
} while (new!=old);
```



Interpolant sequence

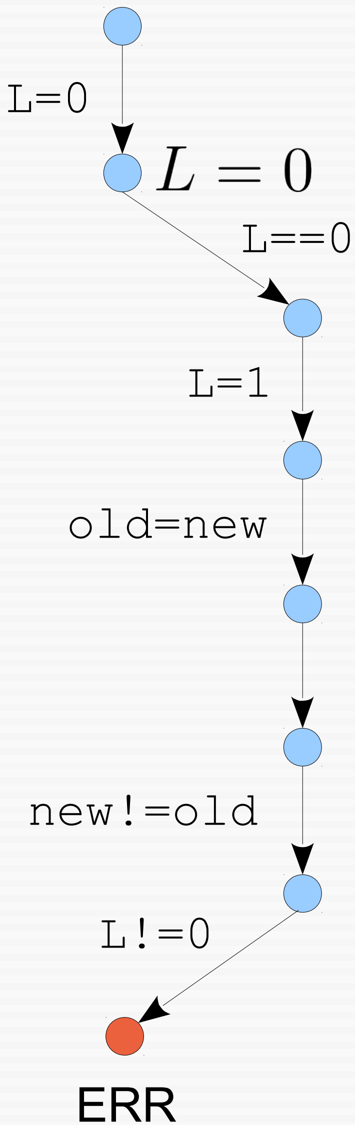
Definition

Suppose a conjunction $T_1 \wedge \cdots \wedge T_n$.

An *interpolant sequence* is a sequence $I_0, \ldots, I_n$ of formulae such that

- $I_0 = \top$
- $I_n = \bot$
- $I_{i-1}, T_i \vdash I_i$ for each $i = 1, \ldots, n$
- for each $i = 1, \ldots, n$, the formula I_i only contains symbols common to $T_1 \wedge \cdots \wedge T_i$ and $T_{i+1} \wedge \cdots \wedge T_n$

In the example



$$T_1 : \quad L_0 = 0$$
$$T_2 : \quad L_0 = 0$$
$$T_3 : \quad L_1 = 1$$
$$T_4 : \quad old_1 = new_0$$
$$T_5 : \quad \top$$
$$T_6 : \quad new_0 \neq old_1$$
$$T_7 : \quad L_1 \neq 0$$

$$I_0 : \quad \top$$
$$I_1 : \quad \top$$
$$I_2 : \quad \top$$
$$I_3 : \quad \top$$
$$I_4 : \quad old_1 = new_0$$
$$I_5 : \quad old_1 = new_0$$
$$I_6 : \quad \bot$$
$$I_7 : \quad \bot$$

Computation of sequence int.

Lemma
If a logic/theory admits binary interpolants, it also admits sequence interpolants.

Proof:
Solve a sequence of binary interpolation problems:

$$
\begin{array}{rcl}
 & & I_0 := \top \\
\big(I_0 \wedge T_1\big) \wedge \big(T_2 \wedge \cdots \wedge T_n\big) & \leadsto & I_1 \\
\big(I_1 \wedge T_2\big) \wedge \big(T_3 \wedge \cdots \wedge T_n\big) & \leadsto & I_2 \\
 & \vdots & \\
\big(I_{i-1} \wedge T_i\big) \wedge \big(T_{i+1} \wedge \cdots \wedge T_n\big) & \leadsto & I_i
\end{array}
$$

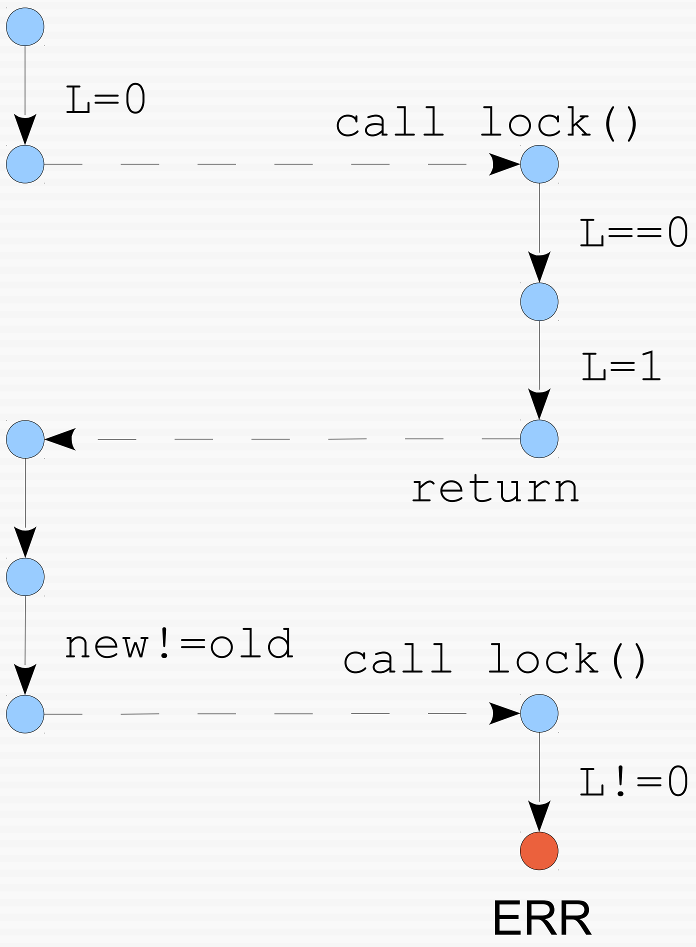
Computation of sequence int. (2)

- In practice:
  - Compute a single SAT/SMT proof
  - Extract a sequence of interpolants directly from this proof
  - Meta-argument: this yields actual interpolant sequence

Procedure calls, recursion
[Heizmann et al, 2010]

```
L = 0;
do {
   lock();
   old = new;
   if (*){
      L = 0;
      new++;
   }
} while (new!=old);


void lock() {
   assert(L==0);
   L = 1;
}
```



Tree interpolants

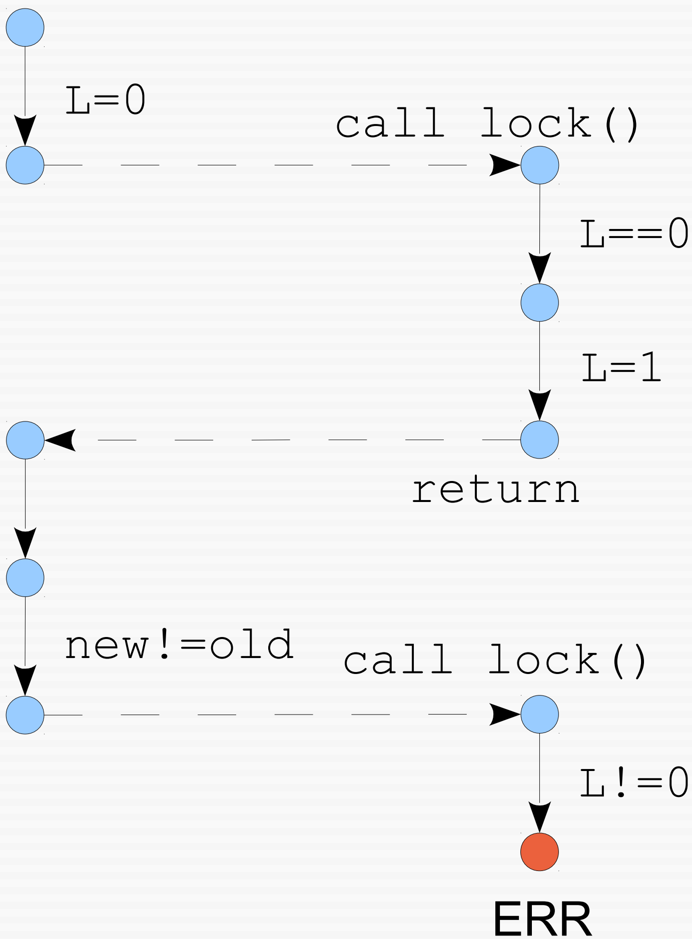
Definition
Suppose (V, E) is a finite directed tree ($E(v, w)$ means that w is a direct child of v). Further, let ϕ be a labelling of nodes $v \in V$ with formulae $\phi(v)$.
A *tree interpolant* I is a mapping from nodes $v \in V$ to formulae such that

- $I(v_0) = \bot$ for the root node $v_0 \in V$;
- for any node $v \in V$, it holds that

$$\phi(v) \wedge \bigwedge_{(v,w)\in E} I(w) \ \vdash \ I(v) \ ;$$

- any non-logical symbol in $I(v)$ occurs both in some $\phi(w)$ with $E^*(v, w)$, and in some $\phi(w')$ with $\neg E^*(v, w')$

In the example



$$
\begin{aligned}
T_1 &: \quad L_0 = 0 \\
T_2 &: \quad L_0 = 0 \\
& \quad\ \top \\
T_3 &: \quad L_1 = 1 \\
T_4 &: \quad old_1 = new_0 \\
T_5 &: \quad \top \\
T_6 &: \quad new_0 \neq old_1 \\
T_7 &: \quad L_1 \neq 0
\end{aligned}
$$

$$
\begin{aligned}
T_2 &: \quad L_0 = 0 \\
T_3 &: \quad L_1 = 1 \\
T_7 &: \quad L_1 \neq 0
\end{aligned}
$$

Computation of tree int.

Lemma
If a logic/theory admits binary interpolants, it also admits tree interpolants.

Proof:
Solve a sequence of binary interpolation problems, starting from the leaves of the tree.

Further schemata

- Symmetric interpolants
- Disjunctive interpolants
- (restricted/unrestricted) DAG interpolants

- *and some more*

Classification using Horn clauses

- Recently proposed as intermediate language for verification tasks
- Recursion-free fragment is useful for characterising Craig interpolation

(Constrained) Horn clauses

Definition
Suppose
- $\mathcal{L}$ is some constraint language (e.g., Presburger a.);
- $\mathcal{R}$ is a set of relation symbols;
- $\mathcal{X}$ is a set of first-order variables.

Then a *Horn clause* is a formula $C \wedge B_1, \ldots, B_n \to H$ where
- C is a constraint in $\mathcal{L}$ (without symbols from $\mathcal{R}$);
- each B_i is a literal of the form $r(t_1, \ldots, t_m)$;
- H is either $\perp$, or of the same form as the B_i .

Solvability

Definition
A set $\mathcal{C}$ of Horn clauses is *syntactically/symbolically solvable* if the $\mathcal{R}$-symbols can be replaced with constraints such that all clauses become valid.

Examples

$$\left.\begin{array}{l} x \geq 3 \rightarrow p(x) \\ p(0) \rightarrow \bot \end{array}\right\} \quad \text{e.g.,} \quad p(x) \equiv x \geq 2$$

$$\left.\begin{array}{l} \top \rightarrow le(x, x+1) \\ le(x,y) \wedge le(y,z) \rightarrow le(x,z) \\ le(x,x) \rightarrow \bot \end{array}\right\} \quad le(x,y) \equiv x < y$$

$$\left.\begin{array}{l} \top \rightarrow le(x, x+1) \\ le(x,y) \wedge le(y,z) \rightarrow le(x,z) \\ le(x,x) \rightarrow \bot \\ p(x,y) \rightarrow le(x,y) \\ \top \rightarrow p(2,0) \end{array}\right\} \quad \text{unsolvable}$$

Recursive Horn clauses

- Dependency graph $(\mathcal{R}, D_{\mathcal{R}})$ on set $\mathcal{R}$ of relation symbols:
  - $(p, q) \in D_{\mathcal{R}}$ if there is a clause with p in the head and q in the body

In the examples

$$\left.\begin{array}{l} x \geq 3 \to p(x) \\ p(0) \to \bot \end{array}\right\} \qquad p$$

$$\left.\begin{array}{l} \top \to le(x, x+1) \\ le(x,y) \wedge le(y,z) \to le(x,z) \\ le(x,x) \to \bot \end{array}\right\} \qquad \begin{tikzcd} le \arrow[loop right] \end{tikzcd}$$

$$\left.\begin{array}{l} \top \to le(x, x+1) \\ le(x,y) \wedge le(y,z) \to le(x,z) \\ le(x,x) \to \bot \\ p(x,y) \to le(x,y) \\ \top \to p(2,0) \end{array}\right\} \qquad \begin{tikzcd} p & le \arrow[l] \arrow[loop right] \end{tikzcd}$$

Recursive Horn clauses (2)

- Cyclic dependencies →
  "Recursive" Horn clauses:
  Resemble a programming language;
  solvability undecidable
- Acyclic graph →
  "Recursion-free" Horn clauses:
  Resemble a program path;
  solvability decidable, **equivalent to Craig interpolation**

Binary interpolation vs. Horn clauses

Definition

Suppose a conjunction $A \wedge B$.

A *reverse interpolant* is a formula I such that

- $A \rightarrow I$ and $B \rightarrow \neg I$ are valid, and
- every non-logical symbol of I occurs in both A and B.

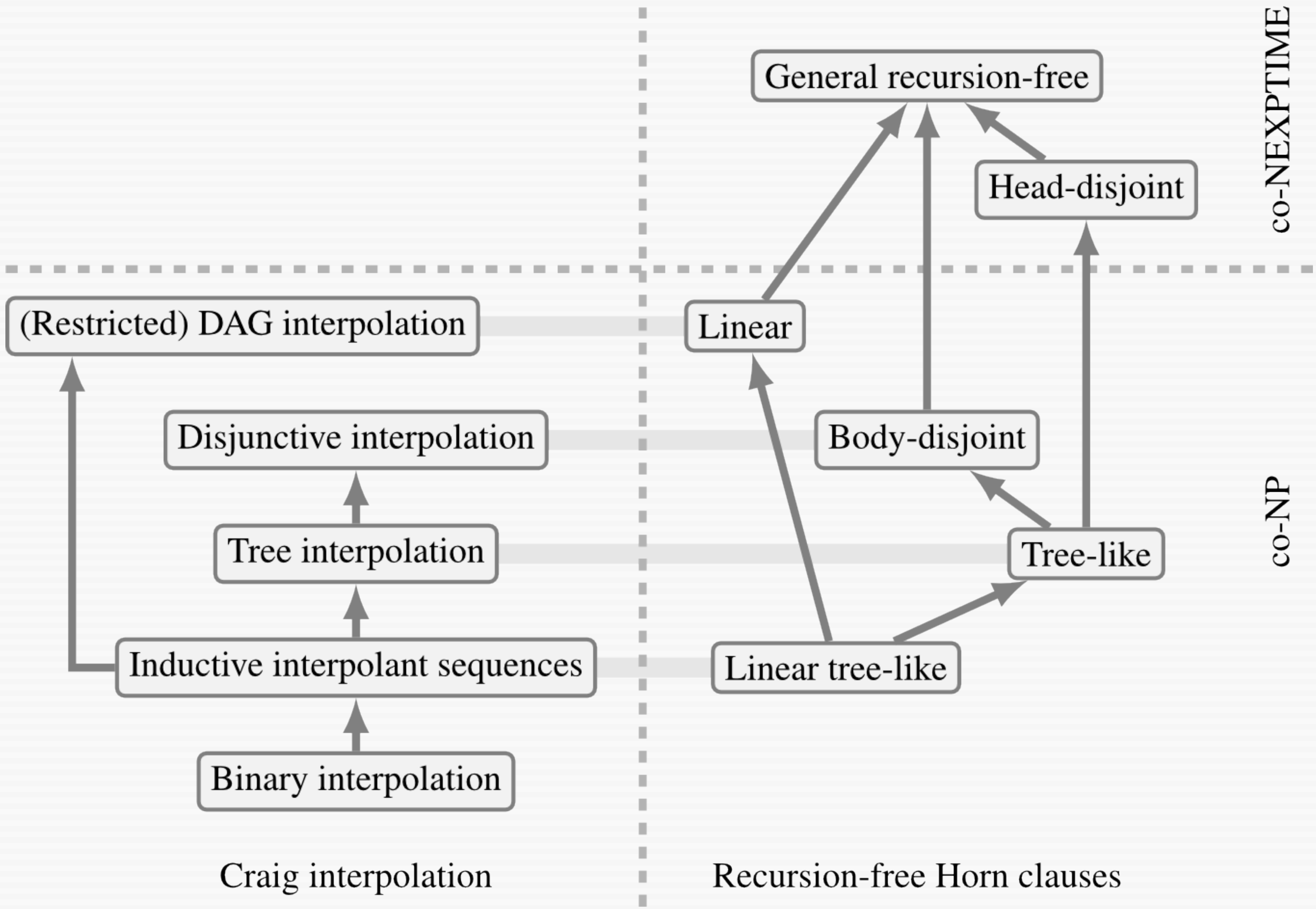
- Equivalent Horn clauses:
  - Let $\bar{x}$ be a vector of common variables in A, B
  - Encode implications as: $A \rightarrow r(\bar{x})$
    $B, r(\bar{x}) \rightarrow \bot$

Interpolant sequence

$$T_1 \wedge \cdots \wedge T_n$$

$$\begin{aligned}
\top &\rightarrow r_0(\bar{x}_0) \\
r_0(\bar{x}_0) \wedge T_1 &\rightarrow r_1(\bar{x}_1) \\
r_1(\bar{x}_1) \wedge T_2 &\rightarrow r_2(\bar{x}_2) \\
&\vdots \\
r_{i-1}(\bar{x}_{i-1}) \wedge T_i &\rightarrow r_i(\bar{x}_i) \\
&\vdots \\
r_n(\bar{x}_n) &\rightarrow \bot
\end{aligned}$$

General picture

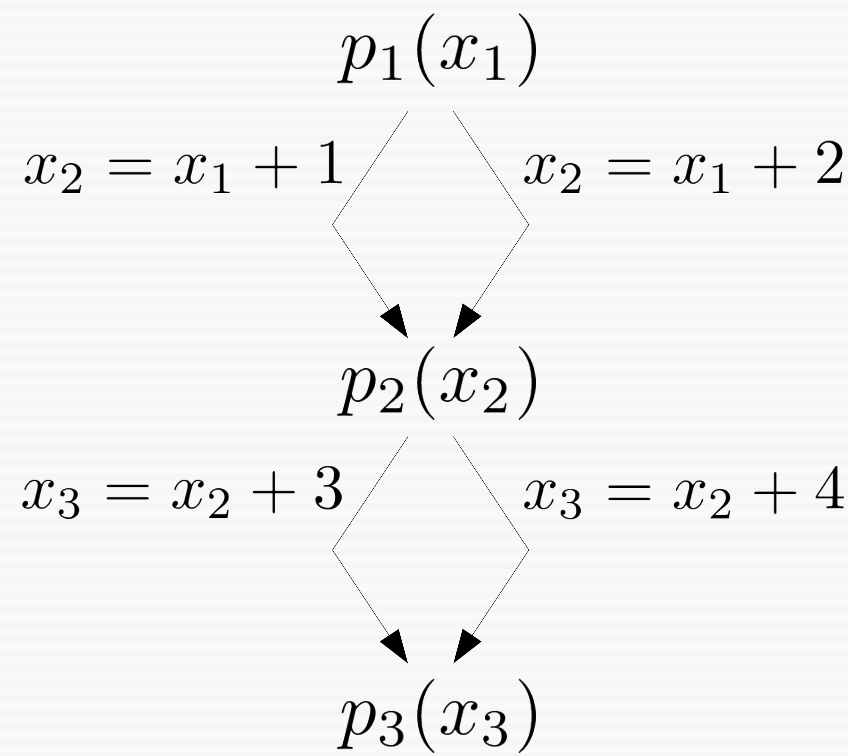


Horn fragments

- **Linear**: at most one literal per body
- **Head-disjoint**: no relation symbol occurs in more than one head
- **Body-disjoint**: no relation symbol occurs more than once in a body
- **Tree-like**: head-disjoint + body-disjoint

Linear

```
if (*) {          // p1
    x = x + 1;
} else {
    x = x + 2;
}
if (*) {          // p2
    x = x + 3;
} else {
    x = x + 4;
}
                  // p3
[...]
```



$$p_1(x) \to p_2(x+1)$$
$$p_1(x) \to p_2(x+2)$$
$$p_2(x) \to p_3(x+3)$$
$$p_2(x) \to p_3(x+4)$$

Tree-like

$T_2: \quad L_0 = 0$

$T_3: \quad L_1 = 1$

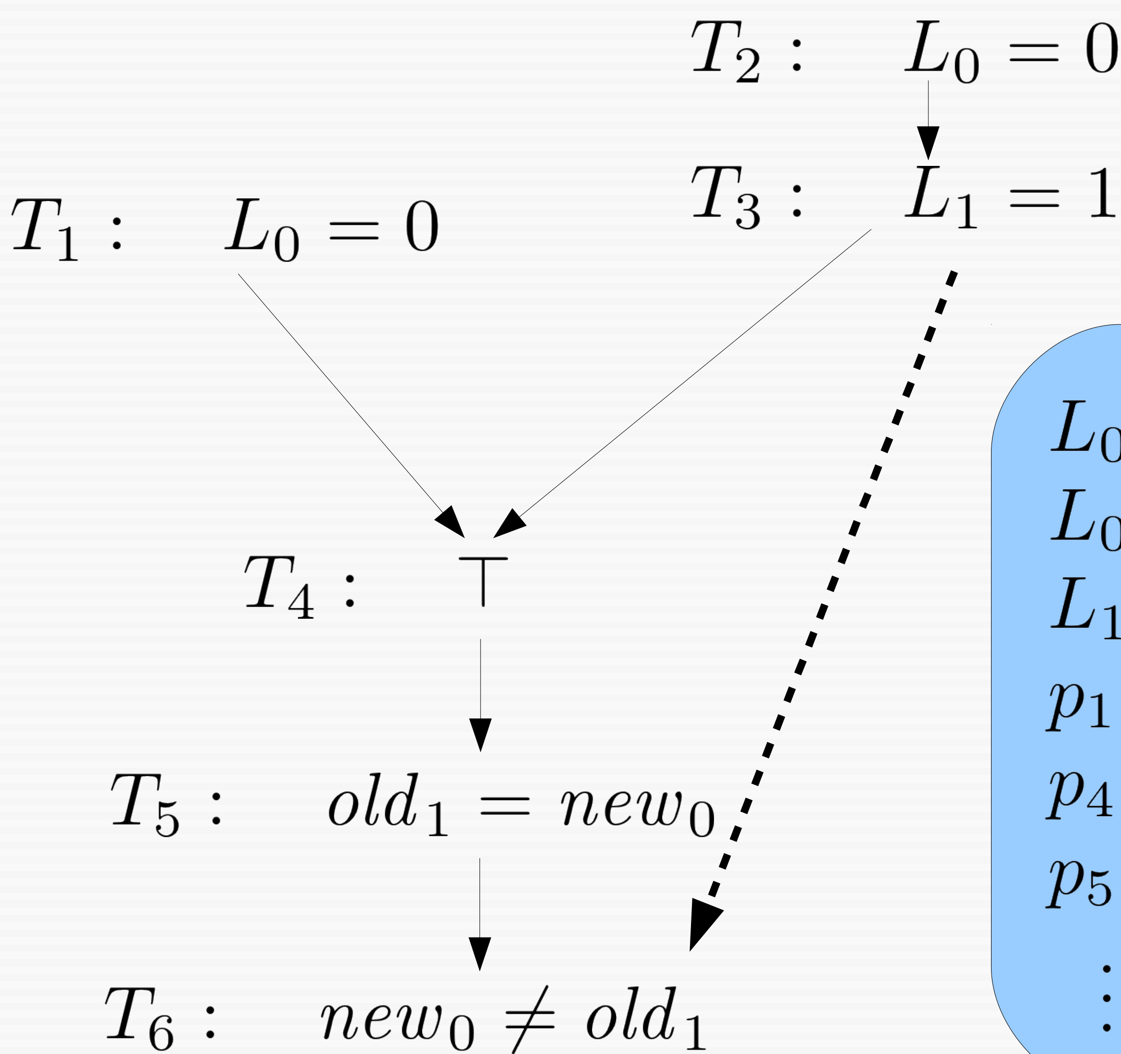
$T_1: \quad L_0 = 0$

$T_4: \quad \top$

$T_5: \quad old_1 = new_0$

$$
\begin{array}{l}
L_0 = 0 \rightarrow p_1(L_0) \\
L_0 = 0 \rightarrow p_2(L_0) \\
L_1 = 1 \rightarrow p_3(L_0, L_1) \\
p_1(L_1) \wedge p_3(L_0, L_1) \rightarrow p_4 \\
\vdots
\end{array}
$$

Head-disjoint



$$
\begin{aligned}
&L_0 = 0 \rightarrow p_1(L_0)\\
&L_0 = 0 \rightarrow p_2(L_0)\\
&L_1 = 1 \rightarrow p_3(L_0, L_1)\\
&p_1(L_1) \wedge p_3(L_0, L_1) \rightarrow p_4\\
&p_4 \wedge \cdots \rightarrow p_5(new_0, old_1)\\
&p_5 \wedge p_3(L_0, L_1) \wedge \cdots \rightarrow p_6\\
&\quad\vdots
\end{aligned}
$$

Interpolant Quality

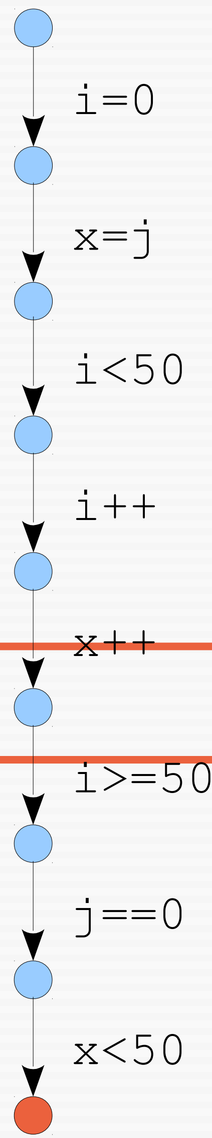
Motivation

- Interpolation queries can have many solutions
- Performance of model checkers highly depends on quality interpolants
  - Small + simple enough
  - General enough

- How can interpolation be controlled?

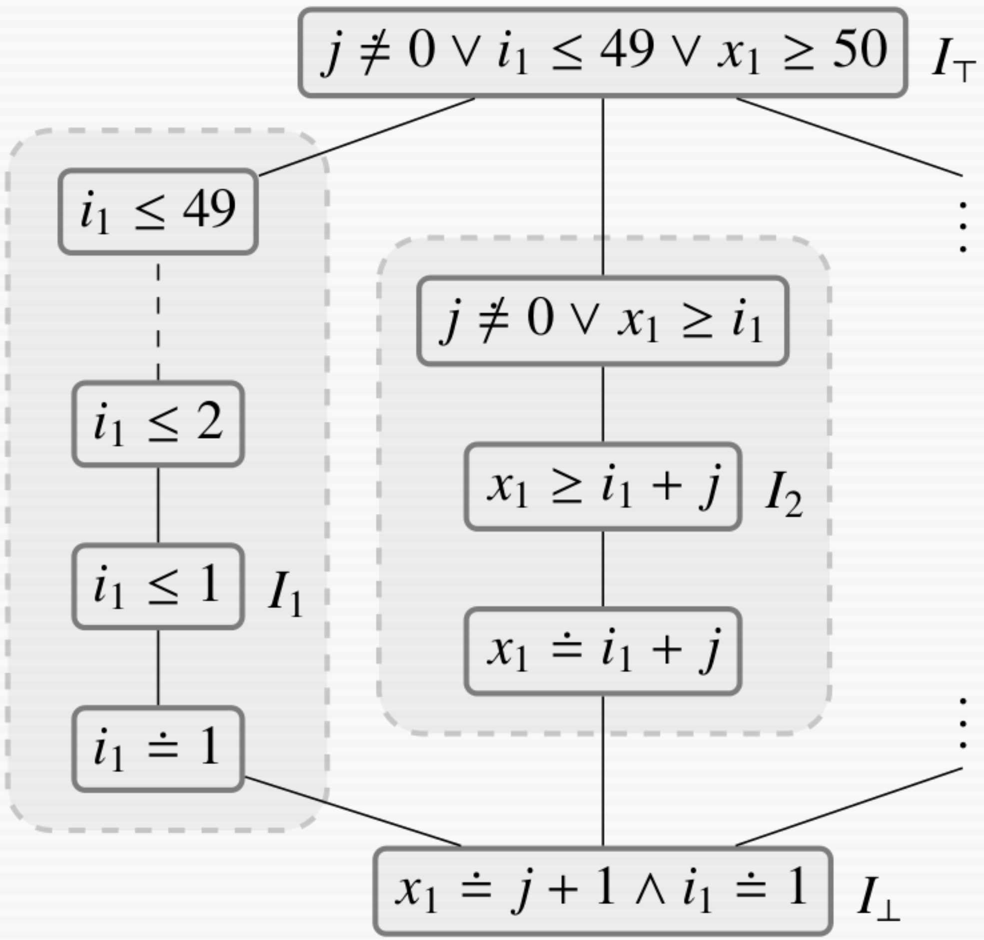
Example

```
i = 0;
x = j;
while (i < 50) {
    i++;
    x++;
}
if (j == 0)
    assert (x >= 50);
```



Bad:	Good:
$\top$	$\top$
$i \leq 0$	$i \leq 0$
$i \leq 0$	$x - i \geq j$
$i \leq 0$	$x - i \geq j$
$i \leq 1$	$x - i + 1 \geq j$
$\boxed{i \leq 1}$	$\boxed{x - i \geq j}$
$\bot$	$x - j \geq 50$
$\bot$	$x \geq 50$
$\bot$	$\bot$

Interpolant lattice



Controlling interpolants ...

- Ad-hoc
- Strength
- Syntactically
- Beautifully
- Semantically

Ad-hoc: useful tricks

- First compute unsatisfiable cores of interpolation problem $A \wedge B$
  - One interpolant can be obtained per core → bigger chance to get something useful

- Rewrite equations $s = t$ to inequalities $s \leq t, s \geq t$
  - Tend to be more general

Interpolant strength

[D'Silva et al, 2010]

- Observation: there are many ways to extract interpolants from a proof
- McMillan's interpolation system Itp_M:

$$\frac{}{c \quad [g(c)]}\ c \in A \qquad\qquad \frac{}{c \quad [\top]}\ c \in B$$

$$\frac{v \vee c \quad [I_1] \qquad \neg v \vee d \quad [I_2]}{c \vee d \quad [I_1 \vee I_2]}\ v \text{ local to } A$$

$$\frac{v \vee c \quad [I_1] \qquad \neg v \vee d \quad [I_2]}{c \vee d \quad [I_1 \wedge I_2]}\ v \text{ not local to } A$$

Interpolant strength (2)

[D'Silva et al, 2010]

- Inverse McMillan ltp_M^{-1}:

$$\frac{}{c \quad [\bot]}\ c \in A \qquad\qquad \frac{}{c \quad [\neg g(c)]}\ c \in B$$

$$\frac{v \lor c \quad [I_1] \qquad \neg v \lor d \quad [I_2]}{c \lor d \quad [I_1 \land I_2]}\ v \text{ local to } B$$

$$\frac{v \lor c \quad [I_1] \qquad \neg v \lor d \quad [I_2]}{c \lor d \quad [I_1 \lor I_2]}\ v \text{ not local to } B$$

Interpolant strength (3)
[D'Silva et al, 2010]

- *Observation:*
  ltp_M interpolants are always **at least as strong** as those from ltp_M^{-1} (when working with same proof)
- Further systems:
  - Symmetric interpolation system
  - Labelled interpolation

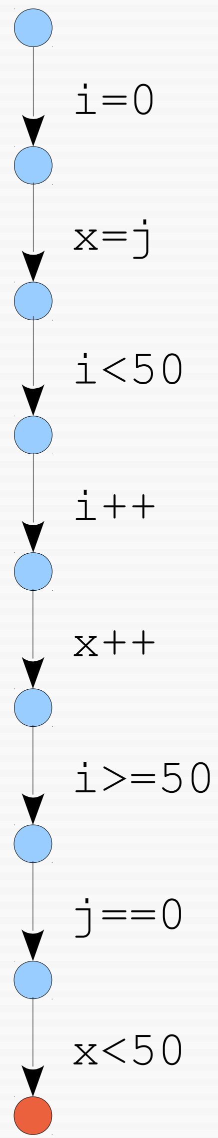
→ Strength can be controlled

Does strength affect MC?

- So far, no conclusive results correlating strength with MC convergence
(for analysis of hardware designs)

- Principal problem:
Interpolants from a *single proof* are not too diverse

In the example



Bad:	Good:
$\top$	$\top$
$i \leq 0$	$i \leq 0$
$i \leq 0$	$x - i \geq j$
$i \leq 0$	$x - i \geq j$
$i \leq 1$	$x - i + 1 \geq j$
$i \leq 1$	$x - i \geq j$
$\bot$	$x - j \geq 50$
$\bot$	$x \geq 50$
$\bot$	$\bot$

Unrelated in strength

Syntactic restrictions
[Jhala, McMillan, 2006]

- *Idea:* restrict interpolant language to enforce generation of right proof

Definition
An L*-restricted interpolant* for a conjunction $T_1 \wedge \cdots \wedge T_n$ is an interpolant sequence $I_0, \ldots, I_n$ such that $I_i \in L$ for every $i = 0, \ldots, n$

Syntactic restrictions (2)

[Jhala, McMillan, 2006]

- Define an ascending chain $L_0 \subseteq L_1 \subseteq \cdots$ of *finite* interpolant languages, such that $\bigcup_i L_i$ includes all formulae
- E.g., L_k only contains formulae with coefficients in $[-k, k]$
- Verification loop:

```
for (k = 0; ; k++) {
    if (verification succeeds using interpolants
          from L_k)
      return Safe!
}
```

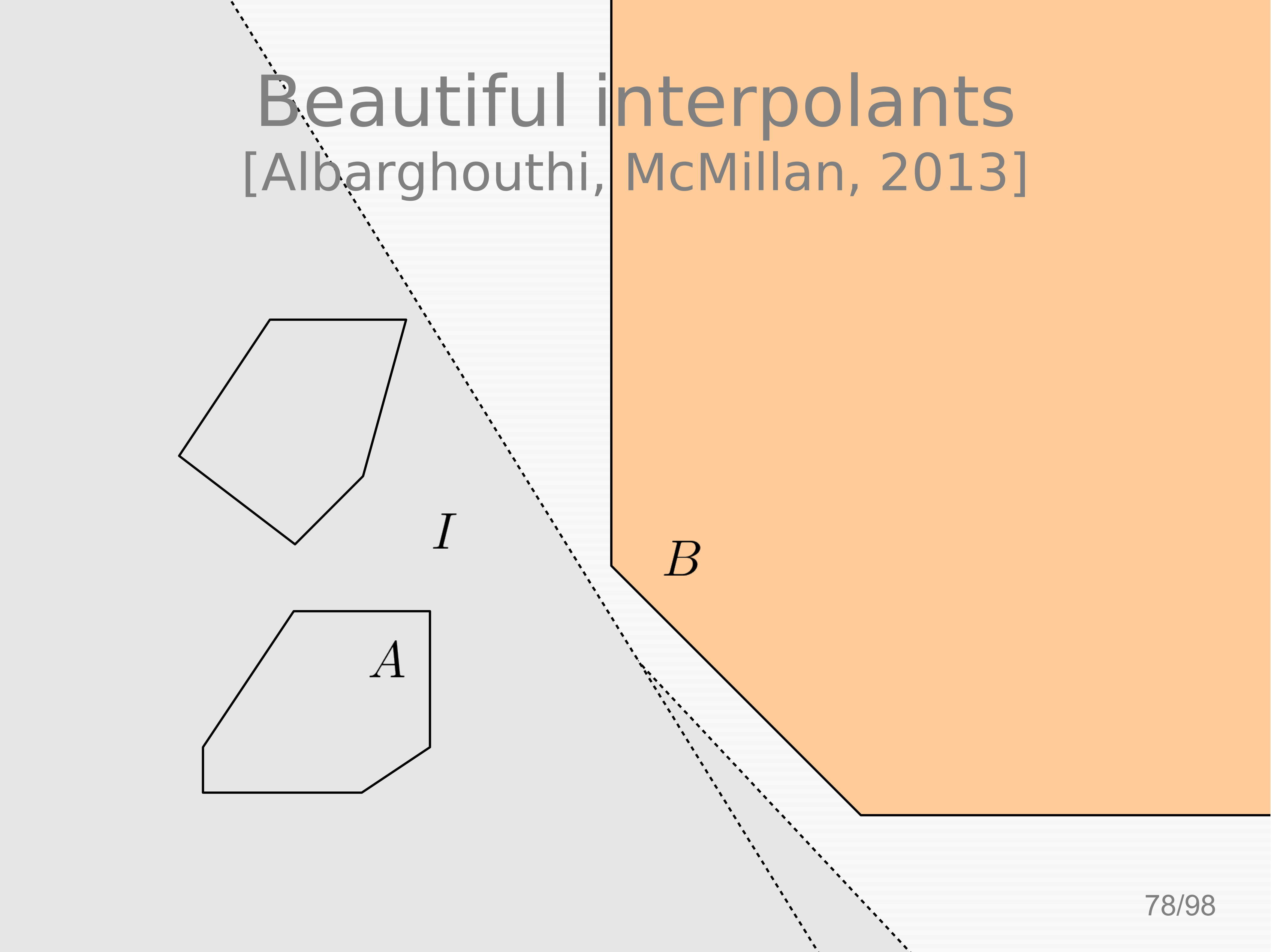
Syntactic restrictions (3)
[Jhala, McMillan, 2006]

Lemma
The verification loop is *complete:* if predicates exist to verify safety of a program, the algorithm will find them eventually.

- Problems:
  - Computation of restricted interpolants is difficult
  - Completeness is of limited value in practice

Beautiful interpolants

[Albarghouthi, McMillan, 2013]

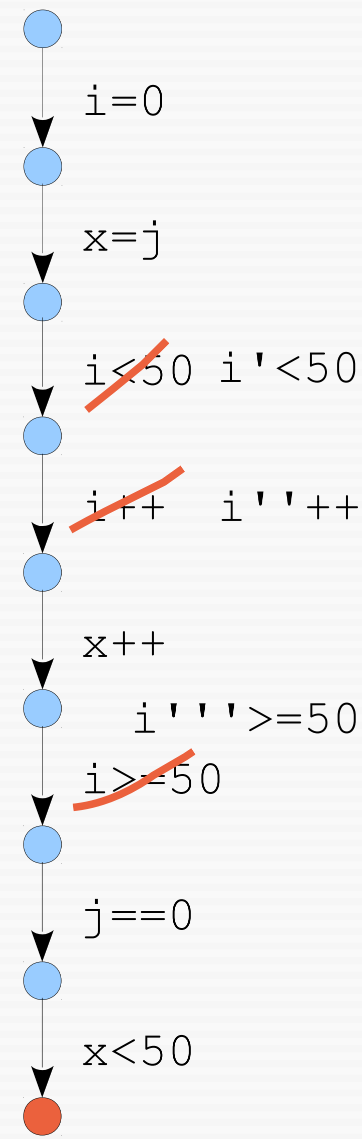


Term abstraction

[Alberti et al, 2012]

- *Idea:* occurrence of symbols in interpolants can be prevented by renaming

```
i = 0;
x = j;
while (i < 50) {
    i++;
    x++;
}
if (j == 0)
    assert (x >= 50);
```



- *Problem:* very coarse-grained

In the example

	Bad:	Good:
	$\top$	$\top$
`i=0`	$i \leq 0$	$i \leq 0$
`x=j`	$i \leq 0$	$x - i \geq j$
`i<50`	$i \leq 0$	$x - i \geq j$
`i++`	$i \leq 1$	$x - i + 1 \geq j$
`x++`	$i \leq 1$	$x - i \geq j$
`i>=50`	$\bot$	$x - j \geq 50$
`j==0`	$\bot$	$x \geq 50$
`x<50`	$\bot$	$\bot$

Interpolation abstractions

[Rümmer et al, 2013]

- *Idea:* prevent occurrences of x, i in interpolant, but still include term $x - i$

Definition

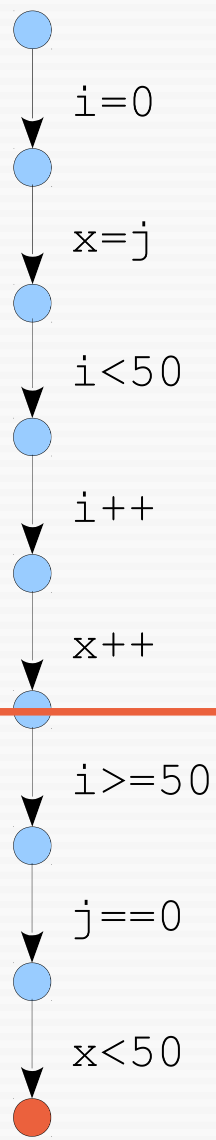
Suppose an interpolation problem $A[\bar{s}_A, \bar{s}] \wedge B[\bar{s}, \bar{s}_B]$

An *interpolation abstraction* is a pair $(R_A[\bar{s}', \bar{s}], R_B[\bar{s}, \bar{s}''])$ of formulae with the property that $R_A[\bar{s}, \bar{s}]$ and $R_B[\bar{s}, \bar{s}]$ are valid.

An *abstract interpolation* problem is an interpolation query

$$\big(A[\bar{s}_A, \bar{s}'] \wedge R_A[\bar{s}', \bar{s}]\big) \wedge \big(R_B[\bar{s}, \bar{s}''] \wedge B[\bar{s}'', \bar{s}_B]\big)$$

In the example



$$\left.\begin{array}{l} i_0 = 0 \wedge \\ x_0 = j' \wedge \\ i_0 < 50 \wedge \\ i'_1 = i_0 + 1 \wedge \\ x'_1 = x_0 + 1 \wedge \end{array}\right\} A[i_0, x_0; i'_1, x'_1, j']$$

$R_A[i'_1, x'_1, j'; i_1, x_1, j]$

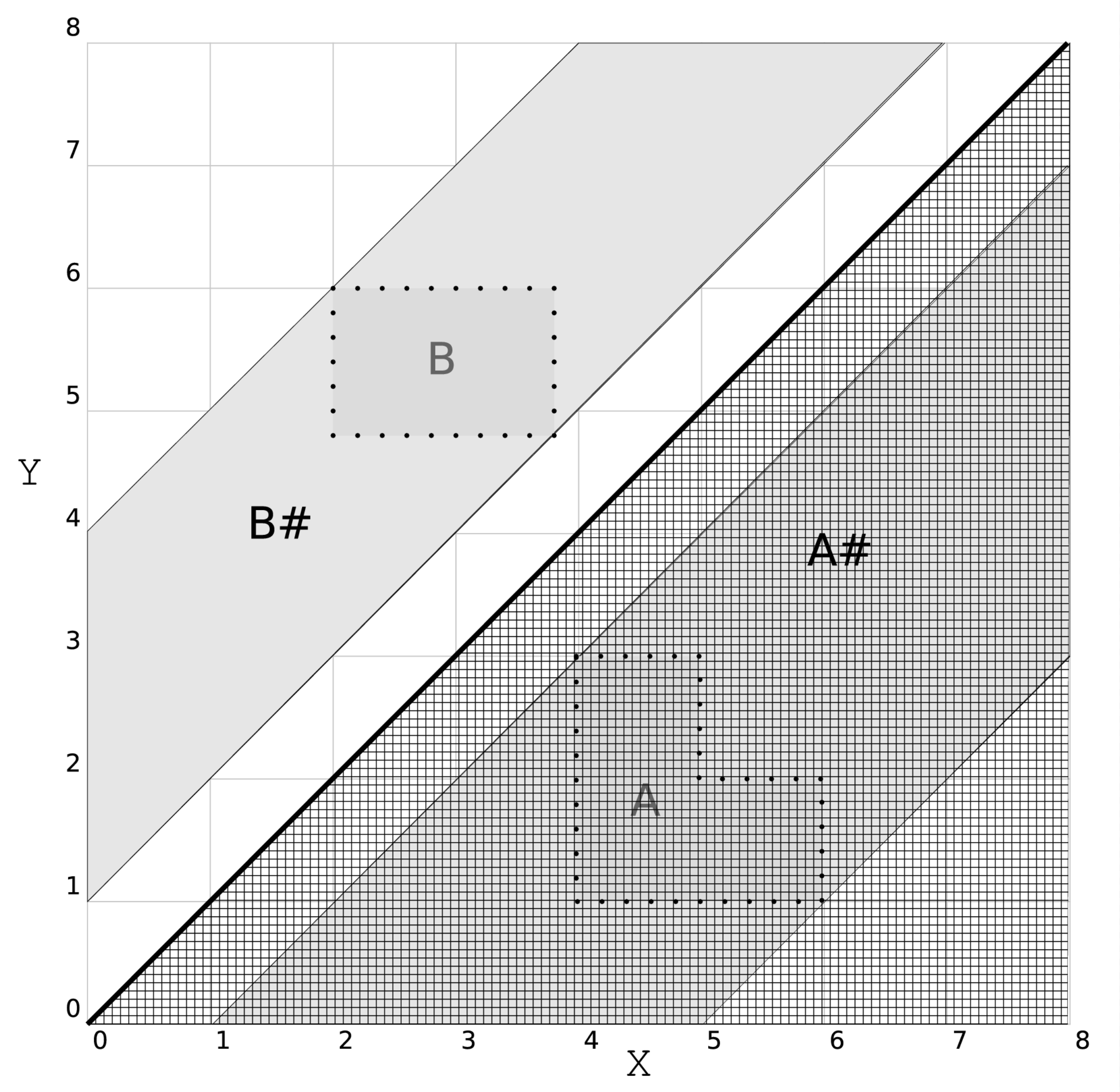
Resulting interpolant:

$$x_1 - i_1 \geq j$$

$R_B[i_1, x_1, j; i''_1, x''_1, j'']$

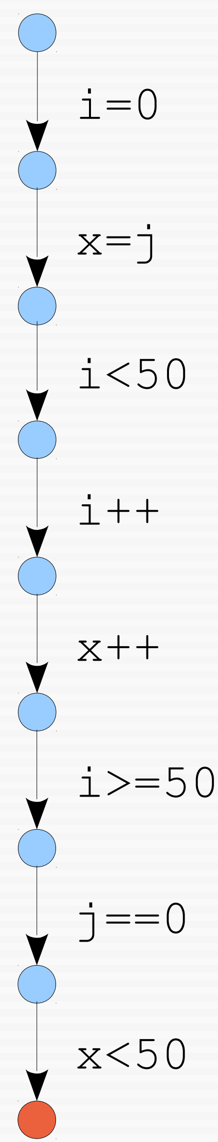
$$\left.\begin{array}{l} i''_1 \geq 50 \wedge \\ j'' = 0 \wedge \\ x''_1 < 50 \end{array}\right\} B[i''_1, x''_1, j'']$$

Semantic view



Syntactic view

Only consider interpolants that can be constructed from *templates* $\{j, x - i\}$

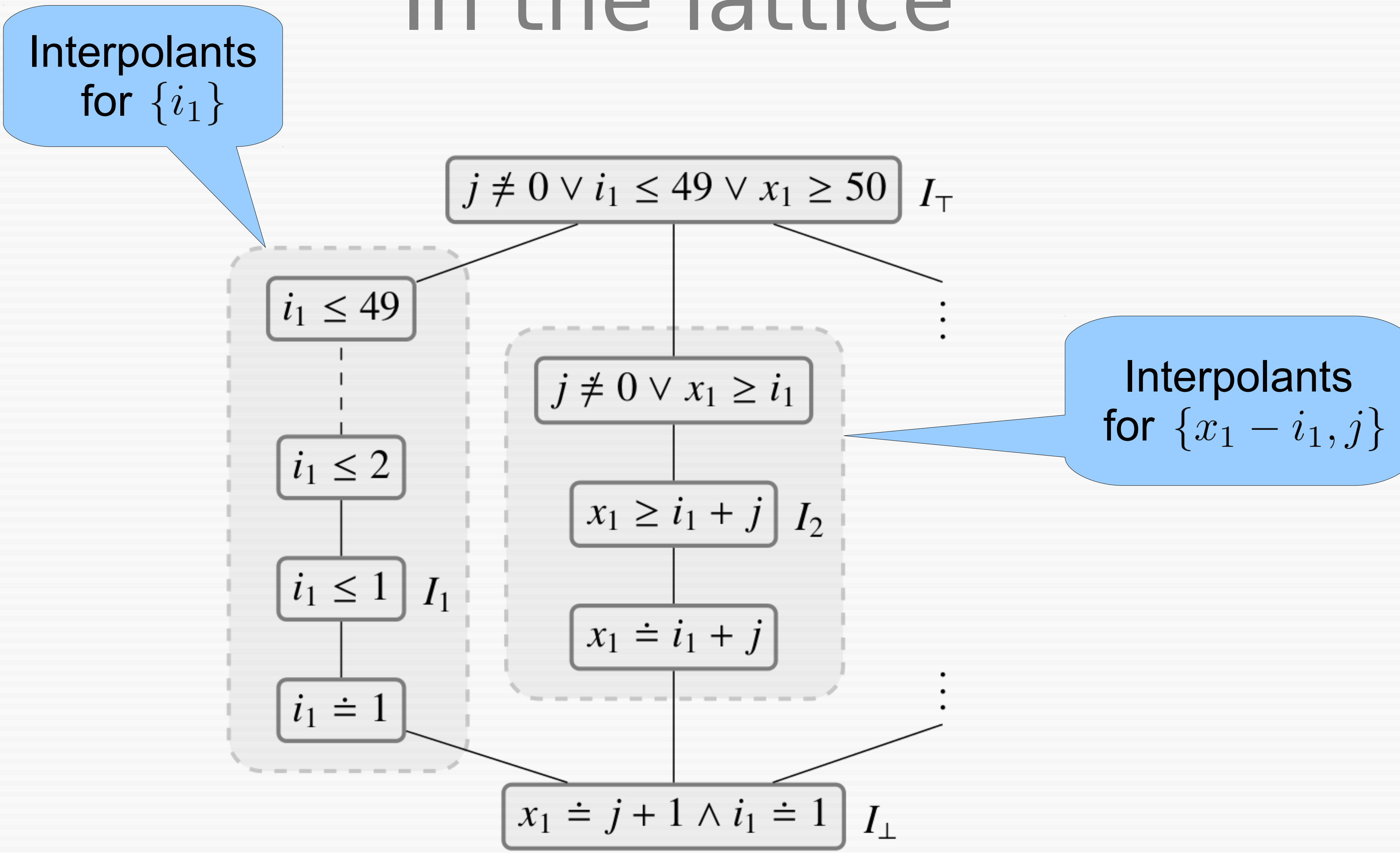


Bad:	Good:
$\top$	$\top$
$i \leq 0$	$i \leq 0$
$i \leq 0$	$x - i \geq j$
$i \leq 0$	$x - i \geq j$
$i \leq 1$	$x - i + 1 \geq j$
$\cancel{i \leq 1}$	$x - i \geq j$
$\bot$	$x - j \geq 50$
$\bot$	$x \geq 50$
$\bot$	$\bot$

Abstraction search

1. Define a base set of interesting templates; e.g., $\{i_1, j, x_1 - i_1\}$
   → apply domain knowledge
2. Search for *maximum feasible abstractions* $(R_A[\bar{s}', \bar{s}], R_B[\bar{s}, \bar{s}''])$ definable using the templates; e.g., $\{i_1\},\ \{j, x_1 - i_1\}$
3. (Prioritise abstractions using costs)
4. Compute an interpolant for each abstraction

In the lattice



Some results on software programs

Benchmark	Eldarica		Eldarica-ABS		Flata	Z3
	N	sec	N	sec	sec	sec
C programs						
boustrophedon (C)	*	*	10	10.7	*	0.1
boustrophedon_expansed (C)	*	*	11	7.7	*	0.1
halbwachs (C)	*	*	53	2.4	*	0.1
gopan (C)	17	22.2	62	57.0	0.4	349.5
rate_limiter (C)	11	2.7	11	19.1	1.0	0.1
anubhav (C)	1	1.7	1	1.6	0.9	*
cousot (C)	*	*	3	7.7	0.7	*
bubblesort (E)	1	2.8	1	2.3	77.6	0.3
insdel (C)	1	0.9	1	0.9	0.7	0.0
insertsort (E)	1	1.8	1	1.7	1.3	0.1
listcounter (C)	*	*	8	2.0	0.2	*
listcounter (E)	1	0.9	1	0.9	0.2	0.0
listreversal (C)	1	1.9	1	1.9	4.9	*
mergesort (E)	1	2.9	1	2.6	1.1	0.2
selectionsort (E)	1	2.4	1	2.4	1.2	0.2
rotation_vc.1 (C)	7	2.0	7	0.3	1.9	0.2
rotation_vc.2 (C)	8	2.7	8	0.2	2.2	0.3
rotation_vc.3 (C)	0	2.3	0	0.2	2.3	0.0
rotation.1 (E)	3	1.8	3	1.8	0.5	0.1
split_vc.1 (C)	18	3.9	17	3.2	*	1.1
split_vc.2 (C)	*	*	18	1.1	*	0.2
split_vc.3 (C)	0	2.8	0	1.5	*	0.0
Recursive Horn SMT-LIB Benchmarks						
addition (C)	1	0.7	1	0.8	0.4	0.0
bfprt (C)	*	*	5	8.3	-	0.0
binarysearch (C)	1	0.9	1	0.9	-	0.0
buildheap (C)	*	*	*	*	-	*
countZero (C)	2	2.0	2	2.0	-	0.0
disjunctive (C)	10	2.4	5	5.0	0.2	0.3
floodfill (C)	*	*	*	*	41.2	0.1
gcd (C)	4	1.2	4	2.0	-	*
identity (C)	2	1.1	2	2.1	-	0.1
merge-leq (C)	3	1.1	7	7.0	15.7	0.1

Some results on Petri nets

Benchmark		Eldarica		ABS (1)		ABS (2)		ABS (3)		ABS-all		Fast
		N	sec	N	sec	N	sec	N	sec	N	sec	sec
Bounded Petri nets												
Basic ME	U	3	1.3	3	1.55	3	1.3	3	1.3	3	1.7	**<1**
IFIP	U	12	2.3	2	1.7	12	4.3	10	4.6	2	1.8	**<1**
L6000	U	*	*	17	16.5	8	4.7	*	*	3	4.0	**<1**
Long 1	U	*	*	1	1.2	7	7.1	*	*	1	1.2	**<1**
Long 2	U	*	*	1	1.4	10	11.1	13	15.4	1	1.4	**<1**
Long 3	U	*	*	*	*	10	11.5	8	8.2	11	19.2	**<1**
Long 4	U	*	*	1	2.8	9	11.2	103	79.6	1	3.0	**<1**
Manufacturing 3	U	*	*	323	802	441	2635	675	1946	354	1588	**2.4**
Manufacturing 9	R	*	*	232	801	264	632	560	3053	295	1515	**10.8**
Unbounded Petri nets												
Alternating bit prot.	R	64	14.8	16	10.5	44	17.5	35	15.2	16	14.7	**4.5**
FMS	R	25	**20.5**	23	28.4	25	27.3	17	24.7	23	32.4	98.4
”	U	18	9.8	2	7.0	13	17.6	18	10.7	2	**6.7**	37.4
FinkelKM	R	16	5.8	15	8.9	16	11.6	17	11.6	15	22.7	**5.7**
”	U	14	5.7	3	**2.4**	6	6.5	7	3.4	3	2.5	5.7
Finkel Counterex.	R	12	2.3	10	3.5	12	2.3	12	2.6	10	3.6	<1
Kanban	R	28	**33.3**	19	35.8	29	70.0	22	41.5	25	67.3	*
”	U	*	*	1	3.9	*	*	*	*	1	**3.8**	*
Mesh 2x2	R	75	**52.3**	64	82.9	60	56.6	68	102	65	105	97
”	U	186	170	18	**33.7**	*	*	*	*	18	37.8	97
Multipool	U	56	423	1	5.4	*	*	*	*	1	**5.0**	*
Pingpong	U	3	1.4	2	1.5	2	1.4	2	1.3	2	1.5	**<1**
PNCSA Cover	R	32	15.0	17	16.5	32	**14.5**	26	16.4	17	17.7	*
Exponential	U	*	*	8	3.9	8	**3.4**	6	5.1	5	5.2	*
Language inclusion	U	*	*	*	*	5	3.7	2	1.7	6	9.0	**<1**

Summary

- New interpolation procedures for various theories still interesting
  - E.g., arrays, bit-vectors

- Central concern at this point: Control form of interpolants

- For many purposes
  Craig interpolation =
  recursion-free Horn solving

References

References

- E. W. Beth (1953), “On Padoa’s method in the theory of definition”
- William Craig: Linear Reasoning. A New Form of the Herbrand-Gentzen Theorem. J. Symb. Log. 22(3): 250-268 (1957)
- S. Maehara. On the interpolation theorem of Craig (in Japanese). Sugaku, 12:235–237, 1961.
- J. Krajcek. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. J. Symbolic Logic, 62(2):457–486, June 1997.

References

- Kenneth L. McMillan: Interpolation and SAT-Based Model Checking. CAV 2003: 1-13
- Kenneth L. McMillan: An interpolating theorem prover. Theor. Comput. Sci. 345(1): 101-121 (2005)
- Kenneth L. McMillan: Quantified Invariant Generation Using an Interpolating Saturation Prover. TACAS 2008: 413-427
- Vijay D'Silva, Daniel Kroening, Mitra Purandare, Georg Weissenbacher: Interpolant Strength. VMCAI 2010: 129-145

References

- Melvin Fitting: First-Order Logic and Automated Theorem Proving, Second Edition. Graduate Texts in Computer Science, Springer 1996, ISBN 978-1-4612-7515-2, pp. I-XVI, 1-326
- Angelo Brillout, Daniel Kroening, Philipp Rümmer, Thomas Wahl: An Interpolating Sequent Calculus for Quantifier-Free Presburger Arithmetic. IJCAR 2010: 384-399
- Alessandro Cimatti, Alberto Griggio, Roberto Sebastiani: Efficient generation of Craig interpolants in satisfiability modulo theories. ACM Trans. Comput. Log. 12(1): 7 (2010)

References

- Andrey Rybalchenko, Viorica Sofronie-Stokkermans: Constraint solving for interpolation. J. Symb. Comput. 45(11): 1212-1233 (2010)
- Aws Albarghouthi, Kenneth L. McMillan: Beautiful Interpolants. CAV 2013: 313-329

References

- Laura Kovács, Andrei Voronkov: Interpolation and Symbol Elimination. CADE 2009: 199-213
- Roberto Bruttomesso, Silvio Ghilardi, Silvio Ranise: Quantifier-free interpolation in combinations of equality interpolating theories. ACM Trans. Comput. Log. 15(1): 5 (2014)
- Kenneth L. McMillan: Lazy Abstraction with Interpolants. CAV 2006: 123-136
- Matthias Heizmann, Jochen Hoenicke, Andreas Podelski: Nested interpolants. POPL 2010: 471-482

References

- Philipp Rümmer, Hossein Hojjat, Viktor Kuncak: Disjunctive Interpolants for Horn-Clause Verification. CAV 2013: 347-363
- Philipp Rümmer, Hossein Hojjat, Viktor Kuncak: Classifying and Solving Horn Clauses for Verification. VSTTE 2013: 1-21
- Arie Gurfinkel, Simone Fulvio Rollini, Natasha Sharygina: Interpolation Properties and SAT-Based Model Checking. ATVA 2013: 255-271
- Sergey Grebenshchikov, Nuno P. Lopes, Corneliu Popeea, Andrey Rybalchenko: Synthesizing software verifiers from proof rules. PLDI 2012: 405-416

References

- Ranjit Jhala, Kenneth L. McMillan: A Practical and Complete Approach to Predicate Refinement. TACAS 2006: 459-473
- Francesco Alberti, Roberto Bruttomesso, Silvio Ghilardi, Silvio Ranise, Natasha Sharygina: Lazy Abstraction with Interpolants for Arrays. LPAR 2012: 46-61
- Philipp Rümmer, Pavle Subotic: Exploring interpolants. FMCAD 2013: 69-76
- Viktor Kuncak, Régis Blanc: Interpolation for synthesis on unbounded domains. FMCAD 2013: 93-96

References

- David Toman and Grant E. Weddell. Fundamentals of Physical Design and Query Compilation. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011
- Aws Albarghouthi, Arie Gurfinkel, Marsha Chechik: Craig Interpretation. SAS 2012: 300-316