Symbolic Computation and Theorem Proving in Program Analysis

Laura Kovács

Chalmers
Outline

Part 1: Weakest Precondition for Program Analysis and Verification

Part 2: Polynomial Invariant Generation (TACAS’08, LPAR’10)

Part 3: Quantified Invariant Generation (FASE’09, MICAI’11)

Part 4: Invariants, Interpolants and Symbol Elimination (CADE’09, POPL’12, APLAS’12)
Part 4: Invariants, Interpolants and Symbol Elimination

Symbol Elimination by First-Order Theorem Proving

Invariants, Interpolants and Symbol Elimination

Interpolants from Proofs

Interpolation in Vampire

Quality of Interpolants

Conclusions
Outline

Invariants, Interpolants and Symbol Elimination

Interpolants from Proofs

Interpolation in Vampire

Quality of Interpolants

Conclusions
Invariants, Symbol Elimination, and Interpolation

\{ c = d = 0 \land N > 0 \land (\forall k)(0 \leq k < N \rightarrow D[k] = 0) \} \quad \text{precondition } A(c, d)

\textbf{while } (c < N) \textbf{ do}

\hspace{1em} C[c] := D[d];

\hspace{1em} c := c + 1;
\hspace{1em} d := d + 1

\textbf{end do}

\{(\forall k)(0 \leq k < N \rightarrow C[k] = 0) \} \quad \text{postcondition } B(c, d)
Reachability of $B$ in ONE iteration: $A(c, d) \land T(c, d, c', d') \rightarrow B(c', d')$

$\{ c = d = 0 \land N > 0 \land (\forall k) (0 \leq k < N \rightarrow D[k] = 0) \}$  
precondition $A(c, d)$

while $(c < N)$ do

$C[c] := D[d]$;  
$c < N \land C[c] = D[d] \land c' = c + 1 \land d' = d + 1 \land c' \geq N$  
$T(c, d, c', d')$

$c := c + 1$;  
$d := d + 1$

end do

$\{(\forall k)(0 \leq k < N \rightarrow C[k] = 0)\}$  
postcondition $B(c', d')$
Reachability of $B$ in ONE iteration: $A(c, d) \land T(c, d, c', d') \rightarrow B(c', d')$

$$\{c = d = 0 \land N > 0 \land (\forall k) (0 \leq k < N \rightarrow D[k] = 0)\} \text{ precondition } A(c, d)$$

while $(c < N)$ do

$$C[c] := D[d];$$
$$c := c + 1;$$
$$d := d + 1$$

end do

$$\{(\forall k)(0 \leq k < N \rightarrow C[k] = 0)\} \text{ postcondition } B(c', d')$$

Refutation: $A(c, d) \land T(c, d, c', d') \land \neg B(c', d')$

- The formula is of 2 states $(c, d, c', d')$.
- Need a state formula $l(c', d')$ such that: (Jhala and McMillan)

$$A(c, d) \land T(c, d, c', d') \rightarrow l(c', d') \quad \text{and} \quad l(c', d') \land \neg B(c', d') \rightarrow \bot$$
Reachability of $B$ in ONE iteration: $A(c, d) \land T(c, d, c', d') \to B(c', d')$

$$\{c = d = 0 \land N > 0 \land (\forall k) (0 \leq k < N \to D[k] = 0)\} \quad \text{precondition } A(c, d)$$

while $(c < N)$ do

$$C[c] := D[d]; \quad c < N \land C[c] = D[d] \land c' = c + 1 \land d' = d + 1 \land c' \geq N$$

$$T(c, d, c', d')$$

$$c := c + 1; \quad d := d + 1$$

end do

$$\{(\forall k)(0 \leq k < N \to C[k] = 0)\} \quad \text{postcondition } B(c', d')$$

Refutation: $A(c, d) \land T(c, d, c', d') \land \neg B(c', d')$

- The formula is of 2 states $(c, d, c', d')$.
- Need a state formula $I(c', d')$ such that: (Jhala and McMillan)
  $$A(c, d) \land T(c, d, c', d') \to I(c', d') \quad \text{and} \quad I(c', d') \land \neg B(c', d') \to \bot$$
Invariants, Symbol Elimination, and Interpolation

Reachability of $B$ in ONE iteration: $A(c, d) \land T(c, d, c', d') \rightarrow B(c', d')$

$$\{ c = d = 0 \land N > 0 \land (\forall k) (0 \leq k < N \rightarrow D[k] = 0) \} \text{ precondition } A(c, d)$$

while $(c < N)$ do

$$c < N \land C[c] = D[d] \land c' = c + 1 \land d' = d + 1 \land c' \geq N$$

$$T(c, d, c', d')$$

$c := c + 1;$

$d := d + 1$

end do

$$\{(\forall k)(0 \leq k < N \rightarrow C[k] = 0)\} \text{ postcondition } B(c', d')$$

Refutation: $A(c, d) \land T(c, d, c', d') \land \neg B(c', d')$

- The formula is of 2 states ($c, d, c', d'$).
- Need a state formula $I(c', d')$ such that: (Jhala and McMillan)

$$A(c, d) \land T(c, d, c', d') \rightarrow I(c', d') \text{ and } I(c', d') \land \neg B(c', d') \rightarrow \bot$$

Taks: Compute interpolant $I(c', d')$ by eliminating symbols $c, d$. 
Reachability of $B$ in ONE iteration: $A(c, d) \land T(c, d, c', d') \rightarrow B(c', d')$

\[
\begin{align*}
\{ c = d = 0 \land N > 0 \land (\forall k) (0 \leq k < N \rightarrow D[k] = 0) \} & \quad \text{precondition } A(c, d) \\
\textbf{while } (c < N) \textbf{ do} & \\
& C[c] := D[d]; & c < N \land C[c] = D[d] \land c' = c + 1 \land d' = d + 1 \land c' \geq N \\
& c := c + 1; & T(c, d, c', d') \\
& d := d + 1 & \\
\textbf{end do} & \\
\{(\forall k)(0 \leq k < N \rightarrow C[k] = 0)\} & \quad \text{postcondition } B(c', d')
\end{align*}
\]

\[
\begin{align*}
l(c', d') & \equiv 0 < c' = 1 \land C[0] = D[0] \\
l(c'', d'') & \equiv 0 < c'' = 2 \land C[0] = D[0] \land C[1] = D[1]
\end{align*}
\]

Taks: Compute interpolant $l(c', d')$ by eliminating symbols $c, d$. 

Invariants, Symbol Elimination, and Interpolation

Reachability of \( B \) in TWO iterations: \( A(c, d) \land T(c, d, c', d') \land T(c', d', c'', d'') \rightarrow B(c'', d'') \)

\[
\{ c = d = 0 \land N > 0 \land (\forall k) (0 \leq k < N \rightarrow D[k] = 0) \} \quad \text{precondition } A(c, d)
\]

\[\text{while } (c < N) \text{ do}\]
\[
C[c] := D[d];
\]
\[
c := c + 1;
\]
\[
d := d + 1
\]
\[\text{end do}\]

\[
\{(\forall k)(0 \leq k < N \rightarrow C[k] = 0)\} \quad \text{postcondition } B(c', d')
\]

\[
\begin{align*}
I(c', d') & \equiv 0 < c' = 1 \land C[0] = D[0] \\
I(c'', d'') & \equiv 0 < c'' = 2 \land C[0] = D[0] \land C[1] = D[1]
\end{align*}
\]

Taks: Compute interpolant \( I(c'', d'') \) by eliminating symbols \( c, d, c', d' \).
Invariants, Symbol Elimination, and Interpolation

Reachability of $B$ in TWO iterations: $A(c, d) \land T(c, d, c', d') \land T(c', d', c'', d'') \rightarrow B(c'', d'')$

\[
\{ c = d = 0 \land N > 0 \land (\forall k) (0 \leq k < N \rightarrow D[k] = 0) \}\quad \text{precondition } A(c, d)
\]

\[
\textbf{while } (c < N) \textbf{ do}
\]
\[
C[c] := D[d];
\]
\[
c := c + 1;
\]
\[
d := d + 1
\]
\[
\textbf{end do}
\]

\[
\{(\forall k)(0 \leq k < N \rightarrow C[k] = 0)\}\quad \text{postcondition } B(c', d')
\]

\[
I(c', d') \equiv (\forall k)0 \leq k < c' \rightarrow C[k] = D[k]
\]
\[
I(c'', d'') \equiv (\forall k)0 \leq k < c'' \rightarrow C[k] = D[k]
\]

Taks: Compute interpolant $I(c'', d'')$ implying invariant in any state.
Outline

Invariants, Interpolants and Symbol Elimination

Interpolants from Proofs

Interpolation in Vampire

Quality of Interpolants

Conclusions
Symbol Elimination and Interpolation

What is an Interpolant?

Computing Interpolants
- Local Derivations
- Symbol Eliminations
- Building Interpolants from Proof

Summary: Invariants, Symbol Elimination, Interpolants
Notation

- First-order predicate logic with equality.

- $\top$: always true, $\bot$: always false.

- $\forall A$: universal closure of $A$.

- **Symbols:**
  - predicate symbols;
  - function symbols;
  - constants.

Equality is part of the language $\rightarrow$ equality is not a symbol.

- $\mathcal{L}_A$: the language of $A$: the set of all formulas built from the symbols occurring in $A$. 
What is an Interpolant?

Let $A, B$ be closed formulas such that $A \rightarrow B$.

**Theorem (Craig's Interpolation Theorem)**

There exists a closed formula $I \in \mathcal{L}_A \cap \mathcal{L}_B$ such that

$$A \rightarrow I \quad \text{and} \quad I \rightarrow B.$$ 

$I$ is an interpolant of $A$ and $B$.

**Note:** if $A$ and $B$ are ground, they also have a ground interpolant.
What is an Interpolant?

Let $A, B$ be closed formulas such that $A \rightarrow B$.

**Theorem (Craig’s Interpolation Theorem)**

There exists a closed formula $I \in \mathcal{L}_A \cap \mathcal{L}_B$ such that

$$A \rightarrow I \quad \text{and} \quad I \rightarrow B.$$  

$I$ is an interpolant of $A$ and $B$.

**Reverse interpolant of $A$ and $B$:** any formula $I$ such that

$$A \rightarrow I \quad \text{and} \quad I, \neg B \rightarrow \bot.$$
Interpolation with Theories

- **Theory** $T$: any set of closed formulas.
- $C_1, \ldots, C_n \rightarrow_T C$ means that the formula $C_1 \land \ldots \land C_1 \rightarrow C$ holds in all models of $T$.
- **Interpreted symbols**: symbols occurring in $T$.
- **Uninterpreted symbols**: all other symbols.

**Theorem**

Let $A, B$ be formulas and let $A \rightarrow_T B$.

Then there exists a formula $I$ such that

1. $A \rightarrow_T I$ and $I \rightarrow B$;
2. every uninterpreted symbol of $I$ occurs both in $A$ and $B$;
3. every interpreted symbol of $I$ occurs in $B$.

Likewise, there exists a formula $I$ such that

1. $A \rightarrow I$ and $I \rightarrow_T B$;
2. every uninterpreted symbol of $I$ occurs both in $A$ and $B$;
3. every interpreted symbol of $I$ occurs in $A$. 
Interpolation with Theories

- **Theory** $T$: any set of closed formulas.
- $C_1, \ldots, C_n \rightarrow_T C$ means that the formula $C_1 \land \ldots \land C_1 \rightarrow C$ holds in all models of $T$.
- **Interpreted symbols**: symbols occurring in $T$.
- **Uninterpreted symbols**: all other symbols.

**Theorem**

*Let $A, B$ be formulas and let $A \rightarrow_T B$.*

*Then there exists a formula $I$ such that*

1. $A \rightarrow_T I$ and $I \rightarrow B$;
2. every uninterpreted symbol of $I$ occurs both in $A$ and $B$;
3. every interpreted symbol of $I$ occurs in $B$.

*Likewise, there exists a formula $I$ such that*

1. $A \rightarrow I$ and $I \rightarrow_T B$;
2. every uninterpreted symbol of $I$ occurs both in $A$ and $B$;
3. every interpreted symbol of $I$ occurs in $A$. 
Computing Interpolants using Inference Systems

- **Inference Rule:**
  \[
  \frac{A_1 \ldots A_n}{A}
  \]

- **Inference system:** a set of inference rules.
- **Axiom:** an inference rule with 0 premises.
- **Derivation of** $A$: tree with the root $A$ built from inferences.
AB-derivation
Let $\mathcal{L} = \mathcal{L}_A \cap \mathcal{L}_B$.

A derivation $\Pi$ is an AB-derivation if

(AB1) For every leaf $C$ of $\Pi$ one of following conditions holds:
   1. $A \rightarrow_T \forall C$ and $C \in \mathcal{L}_A$ or
   2. $B \rightarrow_T \forall C$ and $C \in \mathcal{L}_B$.

(AB2) For every inference

$$
\begin{array}{c}
C_1 & \ldots & C_n \\
\hline
C
\end{array}
$$

of $\Pi$ we have $\forall C_1, \ldots, \forall C_n \rightarrow_T \forall C$.

We will refer to property (AB2) as soundness.
Interpolants and Local AB-Derivations

\[
\begin{array}{c}
C_1 \ldots C_n \\
\hline
C
\end{array}
\]

This inference is \textit{local} if the following two conditions hold:

(L1) Either \( \{C_1, \ldots, C_n, C\} \subseteq \mathcal{L}_A \) or \( \{C_1, \ldots, C_n, C\} \subseteq \mathcal{L}_B \).

(L2) If all of the formulas \( C_1, \ldots, C_n \) are colorless, then \( C \) is colorless, too.

A derivation is called \textit{local} if so is every inference of this derivation.
Shape of local derivations for $A \rightarrow B$
Local Derivations: Example $A \rightarrow B$

[Demo]
- $A := \forall x (x = c)$
- $B := a = b$
- Universal interpolant $I: \forall x \forall y (x = y)$

A local refutation of in the superposition calculus:

\[
\begin{align*}
\frac{x = c \quad y = c}{x = y \quad a \neq b} & \quad \frac{a \neq b}{y \neq b \quad \bot}
\end{align*}
\]
Local Derivations: Example $A \rightarrow B$

[demo]
- $A := \forall x(x = c)$
- $B := a = b$
- Universal interpolant $I: \forall x \forall y(x = y)$

A local refutation of in the superposition calculus:

\[
\begin{align*}
\frac{x = c \quad y = c}{x = y} &\quad a \neq b \\
\frac{y \neq b}{\bot}
\end{align*}
\]
Interpolants and Symbol Eliminating Inference

- At least one of the premises colored.
- The conclusion is not colored.

\[
\begin{align*}
x &= c & y &= c \\
x &= y & a &\neq b \\
\end{align*}
\]

Interpolant \(\forall x \forall y (x = y)\): conclusion of a symbol-eliminating inference.
Interpolants and Symbol Eliminating Inference

- At least one of the premises colored.
- The conclusion is not colored.

\[
\frac{x = c \quad y = c}{x = y} \quad a \neq b \\
\frac{y \neq b}{\bot}
\]

Interpolant $\forall x \forall y (x = y)$: conclusion of a symbol-eliminating inference.
Theorem (CADE’09)

Let \( \Pi \) be a closed local \( AB \)-refutation.

Then:

- A reverse interpolant \( I \) of \( A \) and \( B \) can be extracted from \( \Pi \) in linear time.
- \( I \) is ground if all formulas in \( \Pi \) are ground.
- \( I \) is a boolean combination of conclusions of symbol-eliminating inferences of \( \Pi \).

Note:

- No restriction on the calculus (only soundness required) – can be used with theories.
- Can generate interpolants in theories where no good interpolation algorithms exist.
- Shift of interest: what matters are symbol-eliminating inferences.
Theorem (CADE’09)

Let $\Pi$ be a closed local $AB$-refutation.

Then:

- A reverse interpolant $I$ of $A$ and $B$ can be extracted from $\Pi$ in linear time.
- $I$ is ground if all formulas in $\Pi$ are ground.
- $I$ is a boolean combination of conclusions of symbol-eliminating inferences of $\Pi$.

Note:

- No restriction on the calculus (only soundness required) – can be used with theories.
- Can generate interpolants in theories where no good interpolation algorithms exist.
- Shift of interest: what matters are symbol-eliminating inferences.
Extracting Interpolants from Local Proofs

**Theorem (CADE’09)**

*Let* $\Pi$ *be a closed local $AB$-refutation.*

Then:

- A reverse interpolant $I$ of $A$ and $B$ can be extracted from $\Pi$ in linear time.
- $I$ is ground if all formulas in $\Pi$ are ground.
- $I$ is a boolean combination of conclusions of symbol-eliminating inferences of $\Pi$.

**Note:**

- No restriction on the calculus (only soundness required) – can be used with theories.
- Can generate interpolants in theories where no good interpolation algorithms exist.
- Shift of interest: what matters are symbol-eliminating inferences.
Building Interpolants from Proofs

- **Problem:** generation of proofs giving interpolants.
  - Idea 1: look for local refutations only;
  - Idea 2: find calculi that guarantee that local proofs exist.
    - LASCA: Superposition + Linear Arithmetic;
    - Separating orderings (colored symbols are the greatest).

**Theorem (CADE’09)**

*If \(\succ\) is separating, then every AB-derivation in LASCA is local.*

First-order interpolation implemented in Vampire.
Building Interpolants from Proofs

- **Problem**: generation of proofs giving interpolants.
  - Idea 1: look for local refutations only;
  - Idea 2: find calculi that guarantee that local proofs exist.
    - LASCA: Superposition + Linear Arithmetic;
    - Separating orderings (colored symbols are the greatest).

**Theorem (CADE’09)**

*If* $\succ$ *is separating, then every AB-derivation in LASCA is local.*

First-order interpolation implemented in Vampire.
Building Interpolants from Proofs

- **Problem:** generation of proofs giving interpolants.
  - **Idea 1:** look for local refutations only;
  - **Idea 2:** find calculi that guarantee that local proofs exist.
    - LASCA: Superposition + Linear Arithmetic;
    - Separating orderings (colored symbols are the greatest).

**Theorem (CADE’09)**

If $\succ$ is separating, then every AB-derivation in LASCA is local.

First-order interpolation implemented in Vampire.
Building Interpolants from Proofs

Problem: generation of proofs giving interpolants.
  - Idea 1: look for local refutations only;
  - Idea 2: find calculi that guarantee that local proofs exist.
    - LASCA: Superposition + Linear Arithmetic;
    - Separating orderings (colored symbols are the greatest).

Theorem (CADE’09)
If \( \succ \) is separating, then every AB-derivation in LASCA is local.

First-order interpolation implemented in Vampire.
<table>
<thead>
<tr>
<th>Formulas</th>
<th>Coloring</th>
<th>Reverse Interpolant</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L: \ z &lt; 0 \land x \leq z \land y \leq x$</td>
<td>left: $z$</td>
<td>$y \leq x \land x &lt; 0$</td>
</tr>
<tr>
<td>$R: \ y \leq 0 \land x + y \geq 0$</td>
<td>right: $-$</td>
<td></td>
</tr>
<tr>
<td>$L: \ g(a) = c + 5 \land f(g(a)) \geq c + 1$</td>
<td>left: $g, a$</td>
<td>$c + 1 \leq f(c + 5)$</td>
</tr>
<tr>
<td>$R: \ h(b) = d + 4 \land d = c + 1 \land f(h(b)) &lt; c + 1$</td>
<td>right: $h, b$</td>
<td></td>
</tr>
<tr>
<td>$L: \ p \leq c \land c \leq q \land f(c) = 1$</td>
<td>left: $c$</td>
<td>$p \leq q \land (q &gt; p \lor f(p) = 1)$</td>
</tr>
<tr>
<td>$R: \ q \leq d \land d \leq p \land f(d) = 0$</td>
<td>right: $d$</td>
<td></td>
</tr>
<tr>
<td>$L: \ f(x_1) + x_2 = x_3 \land f(y_1) + y_2 = y_3 \land y_1 \leq x_1$</td>
<td>left: $f$</td>
<td>$x_1 &gt; y_1 \lor x_2 \neq y_2 \lor x_3 = y_3$</td>
</tr>
<tr>
<td>$R: \ x_2 = g(b) \land y_2 = g(b) \land x_1 \leq y_1 \land x_3 \leq y_3$</td>
<td>right: $g, b$</td>
<td></td>
</tr>
<tr>
<td>$L: \ c_2 = \text{car}(c_1) \land c_3 = \text{cdr}(c_1) \land \neg(\text{atom}(c_1))$</td>
<td>left: $\text{car}, \text{cons}$</td>
<td>$\neg\text{atom}(c_1) \land c_1 = \text{cons}(c_2, c_3)$</td>
</tr>
<tr>
<td>$R: \ c_1 \neq \text{cons}(c_2, c_3)$</td>
<td>right: $-$</td>
<td></td>
</tr>
<tr>
<td>$L: \ Q(f(a)) \neq Q(f(b))$</td>
<td>left: $Q, a, b$</td>
<td>$\exists x, y : f(x) \neq f(y)$</td>
</tr>
<tr>
<td>$R: \ f(V) = c$</td>
<td>right: $c$</td>
<td></td>
</tr>
<tr>
<td>$L: \ a = c \land f(c) = a$</td>
<td>left: $a$</td>
<td>$c = f(c)$</td>
</tr>
<tr>
<td>$R: \ c = b \land \neg (b = f(c))$</td>
<td>right: $b$</td>
<td></td>
</tr>
<tr>
<td>$L: \ True \land a'[x'] = y \land x' = x \land y' = y + 1 \land z' = x'$</td>
<td>left: $x, y$</td>
<td>$1 + a'[x'] = y' \land x' = z'$</td>
</tr>
<tr>
<td>$R: \ \neg(y' = a'[z'] + 1)$</td>
<td>right: $-$</td>
<td></td>
</tr>
</tbody>
</table>

**Table:** Interpolation with Vampire, within 1 second time limit.
Symbol Elimination and Interpolation

Invariants, Interpolants and Symbol Elimination

Interpolants from Proofs

Interpolation in Vampire

Quality of Interpolants

Conclusions
Interpolation Through Colors in Vampire

- There are three colors: blue, red and green.
Interpolation Through Colors in Vampire

- There are three colors: blue, red and green.
- Each symbol (function or predicate) is colored in exactly one of these colors.

We have two formulas: $A$ and $B$.
- Each symbol in $A$ is either blue or green.
- Each symbol in $B$ is either red or green.

We know that $\rightarrow A \rightarrow B$.

Our goal is to find a green formula $I$ such that
1. $\rightarrow A \rightarrow I$;
2. $\rightarrow I \rightarrow B$. 
There are three colors: blue, red and green.

Each symbol (function or predicate) is colored in exactly one of these colors.

We have two formulas: $A$ and $B$.

Each symbol in $A$ is either blue or green.

Each symbol in $B$ is either red or green.
Interpolation Through Colors in Vampire

- There are three colors: blue, red and green.
- Each symbol (function or predicate) is colored in exactly one of these colors.
- We have two formulas: $A$ and $B$.
- Each symbol in $A$ is either blue or green.
- Each symbol in $B$ is either red or green.
- We know that $\rightarrow A \rightarrow B$.
- Our goal is to find a green formula $I$ such that
  1. $\rightarrow A \rightarrow I$;
  2. $\rightarrow I \rightarrow B$. 
Interpolation Example in Vampire

fof(fA, axiom, \( q(f(a)) \land \sim q(f(b)) \) ).
fof(fB, conjecture, \(?[V]: V \neq c\) ).
% request to generate an interpolant
vampire(option,show_interpolant,on).
% symbol coloring
vampire(symbol,predicate,q,1,left).
vampire(symbol,function,f,1,left).
vampire(symbol,function,a,0,left).
vampire(symbol,function,b,0,left).
vampire(symbol,function,c,0,right).
% formula L
vampire(left_formula).
  fof(fA,axiom, q(f(a)) & \neg q(f(b)) ).
vampire(end_formula).
% formula R
vampire(right_formula).
  fof(fB,conjecture, \exists[V]\: V \neq c).
vampire(end_formula).
Symbol Elimination and Interpolation

Invariants, Interpolants and Symbol Elimination

Interpolants from Proofs

Interpolation in Vampire

Quality of Interpolants

Conclusions
Given: a problem (an interpolation problem)  
Generate: a formula (an interpolant)

\[-1 + a + -a = -1 \land
\forall x((-x \leq 5) \lor -6 + x \leq -1) \land
-(-1 + -1 + a) = -1 \land
\forall x((1 \leq x + --(-1 + a) \lor -(-1 \leq x))) \land
(a \leq 6 \lor 1 \leq a + -1) \land
\forall x(-(-1 \leq x) \lor -(x \leq -2)) \land
\forall x(-1 \leq x + -a \lor -(-1 + a \leq x)) \land
\forall x(-1 + x = 1 + -2 + x) \land
-a + -1 + a = -1 \land
\forall x((---(-1 + a) \leq x) \lor 1 \leq x + -1) \land
\forall x((-(x \leq 4) \lor -5 + x \leq -1)) \land
\forall x(x + -3 \leq -1 \lor -(x \leq 2)) \land
\forall x(-(-x \leq 3) \lor -4 + x \leq -1) \land
\forall x(x + -a \leq -1 \lor -(x \leq -1 + a)) \land
\forall x(-1 + x = -1 + -1 + a + -(-1 + a) + x) \land
6 \leq b\]
**Given:** a problem (*an interpolation problem*)

**Generate:** a formula (*an interpolant*)

\[-1 + a + -a = -1 \land \\
\forall x (\neg (x \leq 5) \lor -6 + x \leq -1) \land \\
-(-1 + -1 + a) = -1 \land \\
\forall x ((1 \leq x + --(-1 + a) \lor \neg(-1 \leq x))) \land \\
(a \leq 6 \lor 1 \leq a + -1) \land \\
\forall x (\neg(-1 \leq x) \lor \neg(x \leq -2)) \land \\
\forall x (-1 \leq x + -a \lor \neg(-1 + a \leq x)) \land \\
\forall x (-1 + x = 1 + -2 + x) \land \\
-a + -1 + a = -1 \land \\
\forall x (\neg(--(-1 + a) \leq x) \lor 1 \leq x + -1) \land \\
\forall x ((\neg(x \leq 4) \lor -5 + x \leq -1)) \land \\
\forall x (x + -3 \leq -1 \lor \neg(x \leq 2)) \land \\
\forall x (\neg(x \leq 3) \lor -4 + x \leq -1) \land \\
\forall x (x + -a \leq -1 \lor \neg(x \leq -1 + a)) \land \\
\forall x (-1 + x = -1 + -1 + a + (-1 + a) + x) \land \\
6 \leq b\]

or

\[\neg(a \leq 6) \land \\
-a \leq -1 \land \\
\neg(-1 \leq -a) \land \\
a = 3 \land \\
1 \leq -1 + a \land \\
\neg(2 + a \leq 6) \land \\
\neg(-1 + a \leq 1) \land \\
(a \neq 6 \lor \neg(b \leq 6))\]
Given: a problem \textit{(an interpolation problem)}
Generate: a formula \textit{(an interpolant)} which is \textbf{small}

\[-1 + a + -a = -1 \land \forall x(\neg(x \leq 5) \lor -6 + x \leq -1) \land
\neg(-1 + -1 + a) = -1 \land
\forall x((1 \leq x + -(-1 + a) \lor \neg(-1 \leq x))) \land
(a \leq 6 \lor 1 \leq a + -1) \land
\forall x(\neg(-1 \leq x) \lor \neg(x \leq -2)) \land
\forall x(-1 \leq x + -a \lor \neg(-1 + a \leq x)) \land
\forall x(-1 + x = 1 + -2 + x) \land
-a + -1 + a = -1 \land
\forall x(\neg(\neg(-1 + a) \leq x) \lor 1 \leq x + -1) \land
\forall x((\neg(x \leq 4) \lor -5 + x \leq -1)) \land
\forall x(x + -3 \leq -1 \lor \neg(x \leq 2)) \land
\forall x(\neg(x \leq 3) \lor -4 + x \leq -1) \land
\forall x(x + -a \leq -1 \lor \neg(x \leq -1 + a)) \land
\forall x(-1 + x = -1 + -1 + a + -(-1 + a) + x) \land
6 \leq b\]

\[-(a \leq 6) \land
-a \leq -1 \land
\neg(-1 \leq -a) \land
a = 3 \land
1 \leq -1 + a \land
\neg(2 + a \leq 6) \land
\neg(-1 + a \leq 1) \land
(a \neq 6 \lor \neg(b \leq 6))\]
Given: a problem (an interpolation problem)
Generate: a formula (an interpolant) which is small

\[-1 + a + -a = -1 \land
\forall x((- x \leq 5) \lor -6 + x \leq -1) \land
-(-1 + -1 + a) = -1 \land
\forall x((1 \leq x + --(-1 + a) \lor -(-1 \leq x))) \land
(a \leq 6 \lor 1 \leq a + -1) \land
\forall x(-(-1 \leq x) \lor -(-1 \leq x)) \land
\forall x(-1 \leq x + -a \lor -(-1 + a \leq x)) \land
\forall x(-1 + x = 1 + -2 + x) \land
-a + -1 + a = -1 \land
\forall x(--(-1 + a) \leq x) \lor 1 \leq x + -1) \land
\forall x((-(x \leq 4) \lor -5 + x \leq -1)) \land
\forall x(x + -3 \leq -1 \lor -(x \leq 2)) \land
\forall x(--(x \leq 3) \lor -4 + x \leq -1) \land
\forall x(x + -a \leq -1 \lor -(x \leq -1 + a)) \land
\forall x(-1 + x = -1 + -1 + a + -(-1 + a) + x) \land
6 \leq b\]

\neg(a \leq 6) \land
-a \leq -1 \land
\neg(-1 \leq -a) \land
a = 3 \land
1 \leq -1 + a \land
\neg(2 + a \leq 6) \land
\neg(-1 + a \leq 1) \land
(a \neq 6 \lor \neg(b \leq 6))

What is a good interpolant?

- logical strength [Jhala07, D'Silva09, McMillan08];
- small size [Kroening10, Brillout11, Griggio11].
How to Make Interpolants Smaller/Nicer?

- in size;
- in weight;
- in the number of quantifiers;
- ...
How to Make Interpolants Smaller/Nicer?

- in size;
- in weight;
- in the number of quantifiers;
- ... 

Revised Interpolation Problem:

Given $\rightarrow R \rightarrow B$, find a green formula $I$:

- $\rightarrow R \rightarrow I$;
- $\rightarrow I \rightarrow B$;
- $I$ is small.
Extracting Interpolants from Local Proofs

Digest Interpolant: boolean combination of 

\{G_1, \ldots, G_4\}

[McMillan05, KV09]
Extracting Interpolants from Local Proofs

\[ \text{Interpolant: boolean combination of } \{G_1, \ldots, G_4\} \]

[McMillan05, KV09]
Extracting Interpolants from Local Proofs

Interpolant: boolean combination of \{G_1, \ldots, G_4\}

\[\text{Digest}\]
Extracting Interpolants from Local Proofs

G is in the digest:
- comes from a red block
- followed by a blue or green block

Digest

Interpolant: boolean combination of \{G_1, \ldots, G_4\}
Extracting Interpolants from Local Proofs

G is in the digest:
- comes from a red block
- followed by a blue or green block
or
- comes from a blue block
- followed by a red

Interpolant: boolean combination of \{G_1, \ldots, G_4\}
How to Make Interpolants Smaller/Nicer?

Task: minimise interpolants = minimise digest
How to Make Interpolants Smaller/Nicer?

Task: minimise interpolants = minimise digest

Idea: Change the green areas of the local proof
How to Make Interpolants Smaller/Nicer?

Task: minimise interpolants = minimise digest

Idea: Change the green areas of the local proof

Slicing off formulas:

\[
\begin{array}{c}
A_1 \cdots A_n \quad A_{n+1} \cdots A_m \\
\hline
A_0
\end{array}
\quad \overset{slicing \ off \ A}{\rightarrow}
\begin{array}{c}
A_1 \cdots A_{n+1} \cdots A_m \\
\hline
A_0
\end{array}
\]
How to Make Interpolants Smaller/Nicer?

Task: minimise interpolants = minimise digest

Idea: Change the green areas of the local proof

Slicing off formulas

\[
\begin{align*}
A_1 & \quad \cdots \quad A_n & \quad A_{n+1} & \quad \cdots & \quad A_m \\
\overbrace{A_1 \cdots A_n A_{n+1} \cdots A_m}^{slicing \ off\ A} \\
A_0 & \quad \rightarrow \quad A
\end{align*}
\]

If \( A \) is green: Green slicing
How to Make Interpolants Smaller/Nicer?

Task: minimise interpolants = minimise digest

Idea: Change the green areas of the local proof

Slicing off formulas

\[
\begin{array}{c}
B_0 \quad \frac{R_0}{G_1} \\
G_0 \\
\end{array}
\quad \text{slicing off } G_1

\begin{array}{c}
B_0 \quad \frac{R_0}{G_0} \\
\end{array}
\]
How to Make Interpolants Smaller/Nicer?

Task: minimise interpolants = minimise digest

Idea: Change the green areas of the local proof, but preserve locality!

Slicing off formulas

\[
\begin{array}{c}
B_0 \\
R_0 \\
G_1
\end{array}
\quad \quad \quad \quad \quad \quad 
\begin{array}{c}
B_0 \\
R_0 \\
G_1
\end{array}
\]

If \( A \) is green: Green slicing
How to Make Interpolants Smaller/Nicer?

\[ \begin{array}{c}
R_1 & G_1 & B_1 & G_2 \\ 
\hline 
& G_3 & & G_4 \\
& & G_5 & \\
& & & G_6 \\
R_3 \\
& & & G_7 \\
& & \bot \\
R_4 \\
& & & \\
\end{array} \]

Digest:
Reverse interpolant:
Note that the interpolant has changed from $G_4 \rightarrow G_7$ to $\neg G_6$.

▶ There is no obvious logical relation between $G_4 \rightarrow G_7$ and $\neg G_6$, for example none of these formulas implies the other one; ▶ These formulas may even have no common atoms or no common symbols.
How to Make Interpolants Smaller/Nicer?

Digest: \{G_4, G_7\}

Reverse interpolant: \(G_4 \rightarrow G_7\)
How to Make Interpolants Smaller/Nicer?

\[
\begin{array}{cccc}
R_1 & G_1 & B_1 & G_2 \\
\hline
G_3 & & & G_5 \\
\hline
G_6 & & & \\
R_3 & & & \\
R_4 & & & G_7 \\
\hline
\bot & & & \\
\end{array}
\]

Digest:
Reverse interpolant:
Note that the interpolant has changed from $G_4 \to G_7$ to $\neg G_6$.

▶ There is no obvious logical relation between $G_4 \to G_7$ and $\neg G_6$, for example none of these formulas implies the other one;
▶ These formulas may even have no common atoms or no common symbols.
How to Make Interpolants Smaller/Nicer?

Digest: \{G_5, G_7\}

Reverse interpolant: \(G_5 \rightarrow G_7\)
How to Make Interpolants Smaller/Nicer?

\[ \frac{R_1 \quad G_1 \quad B_1 \quad G_2}{G_3} \]

\[ \frac{R_3 \quad G_6}{G_7} \]

\[ \frac{R_4}{\perp} \]

Digest:

Reverse interpolant:

Note that the interpolant has changed from \( G_4 \rightarrow G_7 \) to \( \neg G_6 \).

▶ There is no obvious logical relation between \( G_4 \rightarrow G_7 \) and \( \neg G_6 \), for example none of these formulas implies the other one; ▶ These formulas may even have no common atoms or no common symbols.
How to Make Interpolants Smaller/Nicer?

\[ \frac{R_1 \ G_1 \quad B_1 \ G_2}{G_3} \]

\[ \frac{R_3 \quad G_6}{G_7} \]

[\begin{align*}
\text{Digest:} & \quad \{ G_6, G_7 \} \\
\text{Reverse interpolant:} & \quad G_6 \rightarrow G_7
\end{align*}]
How to Make Interpolants Smaller/Nicer?

\[
\begin{array}{c}
R_1 & G_1 & B_1 & G_2 \\
\hline
G_3 & \\
R_3 & \overline{G_6} \\
\hline
R_4 & \\
\hline
\bot
\end{array}
\]

Digest:
Reverse interpolant:
Note that the interpolant has changed from \( G_4 \rightarrow G_7 \) to \( \neg G_6 \).

▶ There is no obvious logical relation between \( G_4 \rightarrow G_7 \) and \( \neg G_6 \), for example none of these formulas implies the other one;
▶ These formulas may even have no common atoms or no common symbols.
How to Make Interpolants Smaller/Nicer?

Digest: \{G_6\}

Reverse interpolant: \neg G_6
How to Make Interpolants Smaller/Nicer?

Note that the interpolant has changed from $G_4 \rightarrow G_7$ to $\neg G_6$. 

![Interpolant Diagram]

Reverse interpolant:
How to Make Interpolants Smaller/Nicer?

Note that the interpolant has changed from $G_4 \rightarrow G_7$ to $\neg G_6$.

- There is no obvious logical relation between $G_4 \rightarrow G_7$ and $\neg G_6$, for example none of these formulas implies the other one;
- These formulas may even have no common atoms or no common symbols.
How to Make Interpolants Smaller/Nicer?

If green slicing gives us very different interpolants, we can use it for finding small interpolants.

Problem: if the proof contains \( n \) green formulas, the number of possible different slicing off transformations is \( 2^n \).
How to Make Interpolants Smaller/Nicer?

If green slicing gives us very different interpolants, we can use it for finding small interpolants.

Problem: if the proof contains $n$ green formulas, the number of possible different slicing off transformations is $2^n$. 
How to Make Interpolants Smaller/Nicer?

Solution:

- encode all sequences of transformations as an instance of SAT
- solutions encode all slicing off transformations

\[
\neg \text{sliced}(G_1) \rightarrow \text{Green}(G_1) \]
\[
\text{sliced}(G_1) \rightarrow \text{red}(G_1)
\]
\[
\neg \text{sliced}(G_3) \rightarrow \text{Green}(G_3) \]
\[
\text{sliced}(G_3) \rightarrow (\text{Green}(G_3) \leftrightarrow \text{Green}(G_1) \land \text{Green}(G_2))
\]
\[
\text{digest}(G_1) \rightarrow \neg \text{sliced}(G_1)
\]
\[
\text{digest}(G_3) \leftrightarrow (\text{rc}(G_3) \land \text{rf}(G_3)) \lor (\text{bc}(G_3) \land \text{bf}(G_3))
\]
\[
\text{rc}(G_3) \leftrightarrow (\neg \text{sliced}(G_3) \land \text{red}(G_1) \lor \text{red}(G_2))
\]
\[
\text{···}
\]
How to Make Interpolants Smaller/Nicer?

Solution:

- encode all sequences of transformations as an instance of SAT
- solutions encode all slicing off transformations

\[
\begin{array}{ccc}
R & B \\
G_1 & G_2 & \\
G_3 & \\
\end{array}
\]
How to Make Interpolants Smaller/Nicer?

Solution:

- encode all sequences of transformations as an instance of SAT
- solutions encode all slicing off transformations

\[
\begin{array}{c|c|c}
R & B & \\
\hline
G_1 & G_2 & G_3
\end{array}
\]

\[G_3, \text{ and at most one of } G_1, G_2 \text{ can be sliced off.}\]
How to Make Interpolants Smaller/Nicer?

Solution:
- encode all sequences of transformations as an instance of SAT
- solutions encode all slicing off transformations

\[
\begin{array}{c|c|c}
R & B \\
\hline
G_1 & G_2 \\
\hline
G_3 \\
\end{array}
\]

Some predicates on green formulas:
- \text{sliced}(G): G was sliced off;
- \text{red}(G): the trace of G contains a red formula;
- \text{blue}(G): the trace of G contains a blue formula;
- \text{green}(G): the trace of G contains only green formulas;
- \text{digest}(G): G belongs to the digest.
How to Make Interpolants Smaller/Nicer?

Solution:

- encode all sequences of transformations as an instance of SAT
- solutions encode all slicing off transformations

\[
\begin{array}{c|c|c}
R & B \\
\hline
G_1 & G_2 \\
\hline
G_3 & \\
\end{array}
\]

\[\neg \text{sliced}(G_1) \rightarrow \text{Green}(G_1)\]
\[\text{sliced}(G_1) \rightarrow \text{red}(G_1)\]

Some predicates on green formulas:

- \(\text{sliced}(G)\): \(G\) was sliced off;
- \(\text{red}(G)\): the trace of \(G\) contains a red formula;
- \(\text{blue}(G)\): the trace of \(G\) contains a blue formula;
- \(\text{green}(G)\): the trace of \(G\) contains only green formulas;
- \(\text{digest}(G)\): \(G\) belongs to the digest.
How to Make Interpolants Smaller/Nicer?

Solution:

- encode all sequences of transformations as an instance of SAT
- solutions encode all slicing off transformations

\[
\begin{array}{c|c|c}
R & B \\
\hline
G_1 & G_2 \\
G_3 & \\
\end{array}
\]

Some predicates on green formulas:

- \text{sliced}(G): G was sliced off;
- \text{red}(G): the trace of G contains a red formula;
- \text{blue}(G): the trace of G contains a blue formula;
- \text{green}(G): the trace of G contains only green formulas;
- \text{digest}(G): G belongs to the digest.

\[\neg \text{sliced}(G_3) \rightarrow \text{Green}(G_3)\]
\[\text{sliced}(G_3) \rightarrow (\text{Green}(G_3) \leftrightarrow \text{Green}(G_1) \land \text{Green}(G_2))\]
\[\text{sliced}(G_3) \rightarrow (\text{red}(G_3) \leftrightarrow \text{red}(G_1) \lor \text{red}(G_2))\]
\[\text{sliced}(G_3) \rightarrow (\text{blue}(G_3) \leftrightarrow \text{blue}(G_1) \lor \text{blue}(G_2))\]
How to Make Interpolants Smaller/Nicer?

Solution:

- encode all sequences of transformations as an instance of SAT
- solutions encode all slicing off transformations

\[
\begin{array}{c|c|c}
R & B \\
\hline
G_1 & G_2 \\
\hline
G_3 &
\end{array}
\]

Some predicates on green formulas:

- \text{sliced}(G): G was sliced off;
- \text{red}(G): the trace of G contains a red formula;
- \text{blue}(G): the trace of G contains a blue formula;
- \text{green}(G): the trace of G contains only green formulas;
- \text{digest}(G): G belongs to the digest.

\[
digest(G_1) \rightarrow \neg \text{sliced}(G_1)
\]
How to Make Interpolants Smaller/Nicer?

Solution:
- encode all sequences of transformations as an instance of SAT
- solutions encode all slicing off transformations

\[
\begin{array}{c|c|c}
R & B \\
\hline
G_1 & G_2 \\
\hline
G_3 & \\
\end{array}
\]

Some predicates on green formulas:
- \text{sliced}(G): G was sliced off;
- \text{red}(G): the trace of G contains a red formula;
- \text{blue}(G): the trace of G contains a blue formula;
- \text{green}(G): the trace of G contains only green formulas;
- \text{digest}(G): G belongs to the digest.

\[\neg \text{sliced}(G_1) \rightarrow \text{Green}(G_1)\]
\[\text{sliced}(G_1) \rightarrow \text{red}(G_1)\]
\[\neg \text{sliced}(G_3) \rightarrow \text{Green}(G_3)\]
\[\text{sliced}(G_3) \rightarrow (\text{Green}(G_3) \leftrightarrow \text{Green}(G_1) \land \text{Green}(G_2))\]
\[\text{sliced}(G_3) \rightarrow (\text{red}(G_3) \leftrightarrow \text{red}(G_1) \lor \text{red}(G_2))\]
\[\text{sliced}(G_3) \rightarrow (\text{blue}(G_3) \leftrightarrow \text{blue}(G_1) \lor \text{blue}(G_2))\]
\[\text{digest}(G_1) \rightarrow \neg \text{sliced}(G_1)\]
\[\cdots\]
How to Make Interpolants Smaller/Nicer?

**Solution:**

- encode all sequences of transformations as an instance of SAT
- solutions encode all slicing off transformations

\[
\begin{array}{c|c|c}
R & B \\
\hline
G_1 & G_2 \\
\hline
G_3
\end{array}
\]

Express \( \text{digest}(G) \)
How to Make Interpolants Smaller/Nicer?

Solution:

- encode all sequences of transformations as an instance of SAT
- solutions encode all slicing off transformations

\[
\begin{array}{c|c}
R & B \\
\hline
G_1 & G_2 \\
\hline
G_3 & \\
\end{array}
\]

Express \text{digest}(G)

by considering the possibilities:

- \( G \) comes from a
  red/ blue/green formula

- \( G \) is followed by a
  red/ blue/green formula
How to Make Interpolants Smaller/Nicer?

Solution:

- encode all sequences of transformations as an instance of SAT
- solutions encode all slicing off transformations

\[
\begin{array}{c|c|c}
R & B \\
 \hline
G_1 & G_2 \\
 \hline
G_3 & \\
\end{array}
\]

Express \( \text{digest}(G) \)

by considering the possibilities:

- \( G \) comes from a
  - red/ blue/green formula

  \( \text{rc}(G)/\text{bc}(G) \)

- \( G \) is followed by a
  - red/ blue/green formula

  \( \text{bf}(G)/\text{rf}(G) \)
How to Make Interpolants Smaller/Nicer?

Solution:

▶ encode all sequences of transformations as an instance of SAT
▶ solutions encode all slicing off transformations

<table>
<thead>
<tr>
<th>R</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>( G_1 )</td>
<td>( G_2 )</td>
</tr>
</tbody>
</table>

\( G_3 \)

Express \( \text{digest}(G) \)

by considering the possibilities:

▶ \( G \) comes from a red/ blue/green formula

\( \text{rc}(G) / \text{bc}(G) \)

▶ \( G \) is followed by a red/ blue/green formula

\( \text{bf}(G) / \text{rf}(G) \)

\[
\begin{align*}
\text{digest}(G_3) \iff & (\text{rc}(G_3) \land \text{rf}(G_3)) \lor (\text{bc}(G_3) \land \text{bf}(G_3)) \\
\text{rc}(G_3) \iff & (\neg \text{sliced}(G_3) \land (\text{red}(G_1) \lor \text{red}(G_2)))
\end{align*}
\]
How to Make Interpolants Smaller/Nicer?

Solution:

- encode all sequences of transformations as an instance of SAT
- solutions encode all slicing off transformations

```
\begin{array}{c|c|c}
R & B \\
\hline
G_1 & G_2 \\
\hline
G_3 \\
\end{array}
```

Express digest\((G)\) by considering the possibilities:

- \(G\) comes from a red/blue/green formula
  - \(rc(G)/bc(G)\)

- \(G\) is followed by a red/blue/green formula
  - \(bf(G)/rf(G)\)

\(\neg\)sliced\((G_1)\) → Green\((G_1)\)

sliced\((G_1)\) → red\((G_1)\)

\(\neg\)sliced\((G_3)\) → Green\((G_3)\)

sliced\((G_3)\) → (Green\((G_3)\) ↔ Green\((G_1)\) ∧ Green\((G_2)\))

sliced\((G_3)\) → (red\((G_3)\) ↔ red\((G_1)\) ∨ red\((G_2)\))

sliced\((G_3)\) → (blue\((G_3)\) ↔ blue\((G_1)\) ∨ blue\((G_2)\))

digest\((G_1)\) → \(\neg\)sliced\((G_1)\)

digest\((G_3)\) ↔ (rc\((G_3)\) ∧ rf\((G_3)\)) ∨ (bc\((G_3)\) ∧ bf\((G_3)\))

rc\((G_3)\) ↔ (\(\neg\)sliced\((G_3)\) ∧ (red\((G_1)\) ∨ red\((G_2)\))

\ldots
How to Make Interpolants Smaller/Nicer?

Solution:

- encode all sequences of transformations as an instance of SAT
- solutions encode all slicing off transformations

<table>
<thead>
<tr>
<th>R</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_1$</td>
<td>$G_2$</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>$G_3$</td>
<td></td>
</tr>
</tbody>
</table>

Express digest$(G)$ by considering the possibilities:

- $G$ comes from a red/ blue/green formula
- $G$ is followed by a red/ blue/green formula

\[
\begin{align*}
\neg \text{sliced}(G_1) & \rightarrow \text{Green}(G_1) \\
\text{sliced}(G_1) & \rightarrow \text{red}(G_1) \\
\neg \text{sliced}(G_3) & \rightarrow \text{Green}(G_3) \\
\text{sliced}(G_3) & \rightarrow (\text{Green}(G_3) \iff \text{Green}(G_1) \land \text{Green}(G_2)) \\
\text{sliced}(G_3) & \rightarrow (\text{red}(G_3) \iff \text{red}(G_1) \lor \text{red}(G_2)) \\
\text{sliced}(G_3) & \rightarrow (\text{blue}(G_3) \iff \text{blue}(G_1) \lor \text{blue}(G_2)) \\
\text{digest}(G_1) & \rightarrow \neg \text{sliced}(G_1) \\
\text{digest}(G_3) & \iff (\text{rc}(G_3) \land \text{bf}(G_3)) \lor (\text{bc}(G_3) \land \text{bf}(G_3)) \\
\text{rc}(G_3) & \iff (\neg \text{sliced}(G_3) \land \text{red}(G_1) \lor \text{red}(G_2))
\end{align*}
\]
How to Make Interpolants Smaller/Nicer?

Solution:

- encode all sequences of transformations as an instance of SAT;
- solutions encode all slicing off transformations;
- compute small interpolants: smallest digest of green formulas;
- use a pseudo-boolean optimisation tool or an SMT solver to minimise interpolants;
- minimising interpolants is an NP-complete problem.
How to Make Interpolants Smaller/Nicer?

**Solution:**

- encode all sequences of transformations as an instance of SAT;
- solutions encode all slicing off transformations;
- compute small interpolants: smallest digest of green formulas;

\[
\min \{G_{i_1}, \ldots, G_{i_n}\} \left( \sum_{G_i} \text{digest}(G_i) \right)
\]

- use a pseudo-boolean optimisation tool or an SMT solver to minimise interpolants;
- minimising interpolants is an NP-complete problem.
How to Make Interpolants Smaller/Nicer?

**Solution:**

- encode all sequences of transformations as an instance of SAT;
- solutions encode all slicing off transformations;
- compute small interpolants: smallest digest of green formulas;

\[
\min\{G_{i_1},\ldots,G_{i_n}\} \left( \sum_{G_i} \text{digest}(G_i) \right)
\]

\[
\min\{G_{i_1},\ldots,G_{i_n}\} \left( \sum_{G_i} \text{quantifier\_number}(G_i) \text{ digest}(G_i) \right)
\]

- use a pseudo-boolean optimisation tool or an SMT solver to minimise interpolants;
- minimising interpolants is an NP-complete problem.
How to Make Interpolants Smaller/Nicer?

Solution:

- encode all sequences of transformations as an instance of SAT;
- solutions encode all slicing off transformations;
- compute small interpolants: smallest digest of green formulas;

$$\min \{G_{i_1}, \ldots, G_{i_n}\} \left( \sum_{G_i} \text{digest}(G_i) \right)$$

$$\min \{G_{i_1}, \ldots, G_{i_n}\} \left( \sum_{G_i} \text{quantifier\_number}(G_i) \text{ digest}(G_i) \right)$$

- use a pseudo-boolean optimisation tool or an SMT solver to minimise interpolants;
- minimising interpolants is an NP-complete problem.
How to Make Interpolants Smaller/Nicer?

Solution:

- encode all sequences of transformations as an instance of SAT;
- solutions encode all slicing off transformations;
- compute small interpolants: smallest digest of green formulas;

\[
\min\{G_{i_1},...,G_{i_n}\} \left( \sum_{G_i} \text{digest}(G_i) \right)
\]

\[
\min\{G_{i_1},...,G_{i_n}\} \left( \sum_{G_i} \text{quantifier\_number}(G_i) \ \text{digest}(G_i) \right)
\]

- use a pseudo-boolean optimisation tool or an SMT solver to minimise interpolants;
- minimising interpolants is an NP-complete problem.
Experiments with Minimising Interpolants

- Experimental results:
  - 9632 first-order examples from the TPTP library:
    for example, for 2000 problems the size of the interpolants became 20-49 times smaller;
  - 4347 SMT examples:
    - we used Z3 for proving SMT examples;
    - Z3 proofs were localised in Vampire;
    - minimal interpolants were generated for 2123 SMT examples.
Experiments with Minimising Interpolants

- Experimental results:
  - 9632 first-order examples from the TPTP library:
    for example, for 2000 problems the size of the interpolants became 20-49 times smaller;
  - 4347 SMT examples:
    - we used Z3 for proving SMT examples;
    - Z3 proofs were localised in Vampire;
    - minimal interpolants were generated for 2123 SMT examples.
Experiments with Minimising Interpolants

- More realistic benchmarks:
  - 4048 problems coming from CPAchecker;
  - we used Vampire to generate local proofs;
  - minimal interpolants were generated for 1903 CPAchecker examples:
    - for 296 examples the size of the interpolant has decreased by a factor of 5;
    - for 6 examples the size of the interpolant has decreased by a factor of 500.
Symbol Elimination and Interpolation

Invariants, Interpolants and Symbol Elimination

Interpolants from Proofs

Interpolation in Vampire

Quality of Interpolants

Conclusions
**Summary:** Invariant Generation, Interpolation, Symbol Elimination

Given the proof obligation $A \rightarrow B$:

1. Run a theorem prover and **eliminate extra symbols**;

2. Generate a (reverse) interpolant from a refutation;

3. Interpolant is a boolean combination of consequences of **symbol-eliminating inferences**.

Given a loop:

1. Express loop properties in a language containing **extra symbols**;

2. Every **logical consequence** of these properties is a valid loop property, but not an invariant;

3. Run a theorem prover for **eliminating extra symbols**;

4. Every derived formula in the language of the loop is a loop invariant;

5. Invariants are consequences of **symbol-eliminating inferences**.
**Summary: Invariant Generation, Interpolation, Symbol Elimination**

Given the proof obligation $A \rightarrow B$:

1. Run a theorem prover and eliminate extra symbols;
2. Generate a (reverse) interpolant from a refutation;
3. Interpolant is a boolean combination of consequences of symbol-eliminating inferences.

Given a loop:

1. Express loop properties in a language containing extra symbols;
2. Every logical consequence of these properties is a valid loop property, but not an invariant;
3. Run a theorem prover for eliminating extra symbols;
4. Every derived formula in the language of the loop is a loop invariant;
5. Invariants are consequences of symbol-eliminating inferences.
End of Session 4

Slides for session 4 ended here . . .