Craig Interpolation for Integer Arithmetic: Results, Implementation, Experiences

Philipp Rümmer

Uppsala University

IWIL Workshop
March 10th, 2012
Outline

Craig Interpolation for Presburger Arithmetic
- Motivation
- Craig’s theorem
- Results and methods for integers

Implementation, Experiences
- Implementation in the theorem prover PRINCESS
- Experiences with Scala for solvers
- Some experimental data
Motivation: inference of invariants

Generic verification problem ("safety")

{ pre } while (*) Body { post }

Standard approach: loop rule using invariant

\[
\begin{align*}
\text{pre} & \Rightarrow \phi \\
\{ \phi \} & \text{Body} \{ \phi \} \\
\phi & \Rightarrow \text{post}
\end{align*}
\]

How to compute \( \phi \) automatically?
From intermediate assertions to invariants

\[\{\text{pre}\} \text{ Body; Body } \{\text{post}\}\]

Bounded model checking problem

Compute intermediate assertion \(\psi_1\)

\[\{\text{pre}\} \text{ Body } \{\psi_1\}\]
\[\{\psi_1\} \text{ Body } \{\text{post}\}\]

[McMillan, 2003]
From intermediate assertions to invariants

\(\{\text{pre}\} \text{ Body; Body } \{\text{post}\}\)

Bounded model checking problem

Compute intermediate assertion \(\psi_1\)

\(\{\text{pre}\} \text{ Body } \{\psi_1\}\)

\(\{\psi_1\} \text{ Body } \{\text{post}\}\)

\([\psi_1 \Rightarrow \text{pre}]\)

\(\text{pre} \text{ is invariant}\)

[McMillan, 2003]
From intermediate assertions to invariants

{pre} Body; Body {post}

? Bounded model checking problem

Compute intermediate assertion $\psi_1$

{pre} Body {$\psi_1$}  {pre} Body {$\psi_1$}

$[\psi_1 \Rightarrow \text{pre}]$

pre is invariant

{post} Body

[otherwise]

pre is invariant

[McMillan, 2003]
From intermediate assertions to invariants

\{\text{pre } \lor \psi_1\} \text{ Body; Body } \{\text{post}\} \quad ?

Bounded model checking problem \checkmark

Compute intermediate assertion \(\psi_2\)

\{\text{pre } \lor \psi_1\} \text{ Body } \{\psi_2\} \quad \{\psi_2\} \text{ Body } \{\text{post}\}

[\psi_1 \Rightarrow \text{pre}]

pre is invariant \checkmark

[otherwise]

[McMillan, 2003]
From intermediate assertions to invariants

\{ \text{pre} \lor \psi_1 \} \text{ Body; Body } \{ \text{post} \} \quad \text{?}

Bounded model checking problem \checkmark

Compute intermediate assertion \( \psi_2 \)

\{ \text{pre} \lor \psi_1 \} \text{ Body } \{ \psi_2 \}

\{ \psi_2 \} \text{ Body } \{ \text{post} \}

[\psi_2 \Rightarrow \text{pre} \lor \psi_1]

\text{pre} \lor \psi_1 \text{ is invariant} \checkmark

[otherwise]

[McMillan, 2003]
From intermediate assertions to invariants

\[ \{ \text{pre} \lor \psi_1 \} \text{ Body; Body } \{ \text{post} \} \]

Bounded model checking problem

Compute intermediate assertion \( \psi_2 \)

\[ \{ \psi_2 \} \text{ Body } \{ \text{post} \} \]

\[ [\psi_2 \Rightarrow \text{pre} \lor \psi_1] \]

\( \text{pre} \lor \psi_1 \) is invariant

[McMillan, 2003]
How to compute intermediate assertions?

\[ \{ \text{pre} \} \quad \text{pre} \ (s_0) \]

\[ \text{Body;} \quad \rightarrow \quad \text{Body} (s_0, s_1) \]

\[ \text{Body} \quad \rightarrow \quad \text{Body} (s_1, s_2) \]

\[ \{ \text{post} \} \quad \rightarrow \quad \text{post} (s_2) \]
How to compute intermediate assertions?

VC generation

\[
\begin{align*}
\{ \text{pre} \} & \quad \text{pre} \ (s_0) \\
\text{Body;} & \quad \rightarrow \ \text{Body} \ (s_0, s_1) \\
\text{Body} & \quad \rightarrow \ \text{Body} \ (s_1, s_2) \\
\{ \text{post} \} & \quad \rightarrow \ \text{post} \ (s_2)
\end{align*}
\]

Theorem (Craig, 1957)

Suppose \( A \rightarrow C \) is a valid FOL implication. Then there is a formula \( I \) (an interpolant) such that

- \( A \rightarrow I \) and \( I \rightarrow C \) are valid,
- every non-logical symbol of \( I \) occurs in both \( A \) and \( C \).
How to compute intermediate assertions?

\[
\begin{align*}
\{ \text{pre} \} & \quad \text{pre} (s_0) \\
\text{Body; } & \quad \rightarrow \text{Body} (s_0, s_1) \\
\text{Body} & \quad \rightarrow \text{Body} (s_1, s_2) \\
\{ \text{post} \} & \quad \rightarrow \text{post} (s_2)
\end{align*}
\]

\[A(s_0, s_1) \downarrow \]
\[I(s_1) \downarrow \]
\[C(s_1, s_2)\]

Theorem (Craig, 1957)

Suppose \( A \rightarrow C \) is a valid FOL implication. Then there is a formula \( I \) (an interpolant) such that

- \( A \rightarrow I \) and \( I \rightarrow C \) are valid,
- every non-logical symbol of \( I \) occurs in both \( A \) and \( C \).
Illustration

Interpolation problem: \[ A \rightarrow I \rightarrow C \]
Illustration

Interpolation problem: \( A \rightarrow I \rightarrow C \)
Example

Program with assertion:

```c
if (a == 2*x && a >= 0) {
    b = a / 2;
    c = 3*b + 1;
    assert (c > a);
}
```

As a verification condition:

```
a = 2*x & a >= 0
->
2*b <= a & a <= 2*b + 1
->
c = 3*b + 1
->
c > a
```
Example

Program with assertion:

```java
if (a == 2*x && a >= 0) {
    b = a / 2;
    c = 3*b + 1;
    assert (c > a);
}
```

As a verification condition:

- \( a = 2\times x \&\& a \geq 0 \)  
- \( 2\times b \leq a \& a \leq 2\times b + 1 \)
- \( c = 3\times b + 1 \)

- Interpolant: \( 3\times b \geq a \)
- Interpolant: \( c \geq a + 1 \)
- \( c > a \)
Other applications of interpolation

- Blocking lemmas for test-case generation
- Refinement of abstractions in CEGAR
- Computation of summaries
- Synthesis
Interpolation procedures need to support the program logic:

```c
int a[], i;
max = a[0];
for (i = 1; i < n; ++i)
  if (a[i] > max)
    max = a[i];
assert (max >= a[i/2]);
```

E.g., combined use of linear integer arithmetic and arrays
Relevant questions, given a logic $L$

- Is $L$ closed under interpolation?
- Practical interpolation procedures for $L$

**Definition**

Logic $L$ is **closed under interpolation** if for all $A, B \in F$ such that $A \Rightarrow B$, there is an interpolant expressible in $L$.

- In particular:
  Is quantifier-free fragment of $L$ closed under interpolation?
Interpolation for integers

Presburger Arithmetic (QPA)

\[ t ::= \alpha | c | x | \alpha t + \cdots + \alpha t \]
\[ \phi ::= \phi \land \phi | \phi \lor \phi | \neg \phi | \phi \rightarrow \phi | \forall x. \phi | \exists x. \phi \]
\[ | t \doteq 0 | t \leq 0 | \alpha | t \]

\( t \ldots \) terms
\( \phi \ldots \) formulae
\( x \ldots \) variables
\( c \ldots \) constant symbols
\( \alpha \ldots \) integer literals (\( \mathbb{Z} \))
Interpolation for integers

Presburger Arithmetic (QPA)

\[ t ::= \alpha | c | x | \alpha t + \cdots + \alpha t \]

\[ \phi ::= \phi \land \phi | \phi \lor \phi | \neg \phi | \phi \rightarrow \phi | \forall x. \phi | \exists x. \phi \]

\[ t \div 0 | t \leq 0 | \alpha | t \]

Mainly considered here: the quantifier-free fragment (PA)
Interpolation by quantifier elimination (QE)

**Theorem (QE for Presburger Arithmetic)**

For every formula $\phi$ in full QPA, there is an equivalent quantifier-free formula $\psi$ that can effectively be computed.
Lemma

If \( A \rightarrow C \) is a valid implication, then

- \( \exists_{\text{local-syms}(A)}(A) \) is the strongest interpolant,
- \( \forall_{\text{local-syms}(C)}(C) \) is the weakest interpolant.

local-syms(A): symbols occurring in A, but not in C
local-syms(C): . . .

Corollary

Both PA and QPA are closed under interpolation.
Interpolation vs. QE

However . . .

- QE has high computational complexity
- **strongest** and **weakest** interpolants are often not needed/desirable
  ⇒ Larger interpolants, containing irrelevant information
Proof-based interpolation techniques

Implication $A \rightarrow C$

Theorem prover

Proof of $A \rightarrow C$

Proof lifting

Interpolating proof of $A \rightarrow C$

Craig interpolant $A \rightarrow I \rightarrow C$
Abstraction with interpolants

\{\text{pre}\} \text{ Body; Body } \{\text{post}\} \Rightarrow \text{Interpolant extracted from proof}

\Rightarrow \text{Abstraction from unnecessary details}

\text{Bounded model checking problem} \checkmark

\text{Compute intermediate assertion } \psi_1

...
Abstraction with interpolants

\{\text{pre}\} \text{ Body; Body } \{\text{post}\} \\
\downarrow \\
\text{Bounded model checking problem} \quad \checkmark \\
\downarrow \\
\text{Compute intermediate assertion } \psi_1 \\
\downarrow \\
\ldots

\text{Interpolant extracted from proof} \\
\Rightarrow \\
\text{Abstraction from unnecessary details}
Towards practical integer interpolation procedures

- Difference logic
  [McMillan, 2006]

- Integer equalities + divisibility constraints
  [Jain, Clarke, Grumberg, 2008]

- Unit-two-variable-per-inequality
  [Cimatti, Griggio, Sebastiani, 2009]

- Simplex-based, full PA
  [Lynch, Tang, 2008]

  $\Rightarrow$ Leaves local symbols/quantifiers in interpolants
Towards practical interpolation procedures (2)

Proof-based methods for full PA:

- **Sequent calculus-based**
  [Brillout, Kroening, Rümmer, Wahl, 2010]
- **Simplex-based, special branch-and-cut rule**
  [Kroening, Leroux, Rümmer, 2010]
- **Simplex-based, targeting SMT**
  [Griggio, Le, Sebastiani, 2011]
- **From Z3 proofs**
  [McMillan, 2011]
What makes interpolation over integers difficult?
**Definition**

Suppose $A \land B$ is unsatisfiable. A reverse interpolant is a formula $I$ such that

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

**Lemma**

$I$ is reverse interpolant for $A \land B$ if and only if $I$ is interpolant for $A \rightarrow \neg B$.
What makes interpolation over integers difficult?

Consider rational case:

\[ \bigwedge_{i=1}^{n} t_i \leq 0 \quad \land \quad \bigwedge_{j=1}^{m} s_j \leq 0 \]

\( A \) and \( B \)
What makes interpolation over integers difficult?

Consider [**rational**](https://en.wikipedia.org/wiki/Rational_number) case: 

\[
\bigwedge_{i=1}^{n} t_i \leq 0 \quad \wedge \quad \bigwedge_{j=1}^{m} s_j \leq 0
\]

\[A \wedge B\]

**Lemma** (Witnesses)

*\(A \wedge B\) is unsat over \(\mathbb{Q}\) iff there are non-negative \(\{\alpha_i\}_{i=1}^{n}, \{\beta_j\}_{j=1}^{m}\) such that:

\[
\sum_{i=1}^{n} \alpha_i t_i + \sum_{j=1}^{m} \beta_j s_j \in \mathbb{Q}_{>0}
\]
What makes interpolation over integers difficult?

Consider rational case:

\[
\prod_{i=1}^{n} t_i \leq 0 \quad \land \quad \prod_{j=1}^{m} s_j \leq 0
\]

Lemma ( Witnesses )

\( A \land B \text{ is unsat over } \mathbb{Q} \text{ iff there are non-negative } \{ \alpha_i \}_{i=1}^{n}, \{ \beta_j \}_{j=1}^{m} \text{ such that:} \)

\[
\sum_{i=1}^{n} \alpha_i t_i + \sum_{j=1}^{m} \beta_j s_j \in \mathbb{Q}_{>0}
\]

Then:

\[
\sum_{i=1}^{n} \alpha_i t_i \leq 0 \text{ is a reverse interpolant}
\]
What makes interpolation over integers difficult? (2)

Why does this not work for integers?
What makes interpolation over integers difficult? (2)

Why does this not work for integers?

Over \( \mathbb{Z} \), additional rules are needed, such as:

- Branch-and-bound  
  (unproblematic, but incomplete)
- Cutting planes, Gomory cuts
- Cuts-from-proofs
- Omega rule

\( \Rightarrow \) Interpolation more intricate
What makes interpolation over integers difficult? (3)

Theorem

There is a family \( \{ A_n \land B_n \}_n \) of PA formulae such that

- \( A_n \land B_n \) is unsatisfiable,
- \( A_n \land B_n \) has a cutting plane proof of size independent of \( n \),
- all reverse interpolants have size at least linear in \( n \).

(for the definition of PA shown earlier)
What makes interpolation over integers difficult? (4)

Example:

\[ A_n = -n < y + 2nx \land y + 2nx \leq 0 \]
\[ B_n = 0 < y + 2nz \land y + 2nz \leq n \]

All reverse interpolants for \( A_n \land B_n \) are equivalent to:

\[ I_n = (2n \mid y) \lor (2n \mid y + 1) \lor (2n \mid y + 2) \lor \cdots \lor (2n \mid y + n - 1) \]
What makes interpolation over integers difficult? (4)

Example:

\[ A_n = -n < y + 2nx \land y + 2nx \leq 0 \]
\[ B_n = 0 < y + 2nz \land y + 2nz \leq n \]

All reverse interpolants for \( A_n \land B_n \) are equivalent to:

\[ I_n = (2n \mid y) \lor (2n \mid y + 1) \lor (2n \mid y + 2) \lor \cdots \lor (2n \mid y + n - 1) \]

Problematic: mixed cuts
Three main approaches to handle mixed cuts

- Fully expanded interpolants
- Restricted/taylor-made cut rule
- Extended interpolant language

Next:
Comparison + unifying description
Interpolation outline

Implication \( A \rightarrow C \)

\[ \text{Theorem prover} \]

Proof of \( A \rightarrow C \)

\[ \text{Proof lifting} \]

Interpolating proof of \( A \rightarrow C \)

Craig interpolant \( A \rightarrow I \rightarrow C \)
Interpolation outline

Implication $A \rightarrow C$

Proof of $A \rightarrow C$

Proof lifting

Interpolating proof of $A \rightarrow C$

Craig interpolant $A \rightarrow I \rightarrow C$
Main non-interpolating proof rules

**Closure rule ($\alpha > 0$)**

$$
\Gamma, \alpha \leq 0 \vdash \Delta \quad \text{CLOSE-INEQ}'
$$

**Linear combination of inequalities ($\alpha > 0, \beta > 0$)**

$$
\frac{\Gamma, \ldots, \alpha s + \beta t \leq 0 \vdash \Delta}{\Gamma, s \leq 0, t \leq 0 \vdash \Delta} \quad \text{FM-ELIM}'
$$

**Strengthening inequalities (subsumes rounding + cuts)**

$$
\frac{\Gamma, t \not\vdash 0 \vdash \Delta \quad \Gamma, t + 1 \leq 0 \vdash \Delta}{\Gamma, t \leq 0 \vdash \Delta} \quad \text{STRENGTHEN}'
$$
Example of non-interpolating proof

\[ \begin{align*}
\ast & \quad \text{INEQ-CLOSE'} \\
\ldots, \, & \quad 3 \leq 0 \vdash \\
\ldots, \, & \quad 3x \leq 0, \quad -2x + 1 \leq 0 \vdash \\
\ldots, \, & \quad 3x - 2 \leq 0, \quad -2x + 1 \leq 0 \vdash \\
a + x \leq 0, \quad & \quad -a + 2x - 2 \leq 0, \quad -2x + 1 \leq 0 \vdash \\
\end{align*} \]

\[ \text{FM-ELIM'} \]

\[ \text{STRENGTHEN'} \times 2 \]

\[ \text{FM-ELIM'} \]
Interpolation outline

PA implication $A \rightarrow C$

Theorem prover

Proof of $A \rightarrow C$

Proof lifting

Interpolating proof of $A \rightarrow C$

Craig interpolant $A \rightarrow I \rightarrow C$
Basic idea of proof lifting

Interpolation problem: \( A \rightarrow I \rightarrow C \)

\[
\begin{align*}
\Gamma_3 & \vdash \Delta_3 \\
\Gamma_2 & \vdash \Delta_2 \\
\Gamma_1 & \vdash \Delta_1 \\
\vdots & \\
A & \vdash C
\end{align*}
\]
Basic idea of proof lifting

Interpolation problem: \( A \rightarrow I \rightarrow C \)

Main idea: annotations track inequalities from \( A \)
Basic idea of proof lifting

Interpolation problem: \( A \rightarrow I \rightarrow C \)

Main idea: annotations track inequalities from \( A \)
Basic idea of proof lifting

Interpolation problem: \( A \rightarrow I \rightarrow C \)

Main idea: annotations track inequalities from \( A \)
Basic idea of proof lifting

Interpolation problem: \( A \rightarrow I \rightarrow C \)

... 

\[ \Gamma_3 \vdash \Delta_3 \]
\[ \Gamma_2^* \vdash \Delta_2^* \]
\[ \Gamma_1^* \vdash \Delta_1^* \]

... 

\[ A^* \vdash C^* \]

Main idea: annotations track inequalities from \( A \)
Basic idea of proof lifting

Interpolation problem: \( A \rightarrow I \rightarrow C \)

Main idea: annotations track inequalities from \( A \)
Basic idea of proof lifting

Interpolation problem: \( A \rightarrow I \rightarrow C \)

Main idea: annotations track inequalities from \( A \)
Basic idea of proof lifting

Interpolation problem: \( A \Rightarrow I \Rightarrow C \)

Main idea: annotations track inequalities from \( A \)
Basic idea of proof lifting

Interpolation problem: $A \rightarrow I \rightarrow C$

annotation of formulae with labels

$\Gamma^* \vdash \Delta^*_3 \triangleright l_3$

$\Gamma^*_2 \vdash \Delta^*_2 \triangleright l_2$

$\Gamma^*_1 \vdash \Delta^*_1$

$A^* \vdash C^*$

propagation of interpolants

Main idea: annotations track inequalities from $A$
Basic idea of proof lifting

Interpolation problem: $A \rightarrow I \rightarrow C$

annotation of formulae with labels

$\Gamma_3^* \vdash \Delta_3^* \uparrow l_3$
$\Gamma_2^* \vdash \Delta_2^* \uparrow l_2$
$\Gamma_1^* \vdash \Delta_1^* \uparrow l_1$

$A^* \vdash C^*$

propagation of interpolants

Main idea: annotations track inequalities from $A$
Basic idea of proof lifting

Interpolation problem: \( A \rightarrow I \rightarrow C \)

Main idea: annotations track inequalities from \( A \)
Labelled formulae

Interpolation problem: \( A \rightarrow I \rightarrow C \)

<table>
<thead>
<tr>
<th>Labelled formula</th>
<th>Intuition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \phi[\phi^A] )</td>
<td>“( \phi^A ) is ( A )-contribution to ( \phi )”</td>
</tr>
<tr>
<td></td>
<td>( \phi^A ) is the partial interpolant of ( \phi )</td>
</tr>
</tbody>
</table>
Interpolating rules

Interpolation problem: \( A \rightarrow I \rightarrow C \)

**Initialisation rule: \( t \leq 0 \) comes from \( A \)**

\[
\frac{\Gamma, t \leq 0 \left[ t \leq 0 \right] \vdash \Delta \triangleright I}{\Gamma, t \leq 0 \vdash \Delta \triangleright I} \text{ IPI-LEFT-L}
\]

**Initialisation rule: \( t \leq 0 \) comes from \( C \)**

\[
\frac{\Gamma, t \leq 0 \left[ 0 \leq 0 \right] \vdash \Delta \triangleright I}{\Gamma, t \leq 0 \vdash \Delta \triangleright I} \text{ IPI-LEFT-R}
\]

- Similarly for equations, etc.
Interpolating rules

**Closure rule \((\alpha > 0)\)**

\[
\frac{}{\Gamma, \alpha \leq 0 [t^A \leq 0] \vdash \Delta \triangleright t^A \leq 0} \quad \text{CLOSE-INEQ}
\]

**Linear combination of inequalities \((\alpha > 0, \beta > 0)\)**

\[
\frac{}{\Gamma, \ldots, \alpha s + \beta t \leq 0 [\alpha s^A + \beta t^A \leq 0] \vdash \Delta \triangleright l} \quad \text{FM-ELIM}
\]

\[
\frac{}{\Gamma, s \leq 0 [s^A \leq 0], t \leq 0 [t^A \leq 0] \vdash \Delta \triangleright l}
\]
How to interpolate \textsc{strengthen'}?

\[
\Gamma, t \div 0 \vdash \Delta \quad \Gamma, t + 1 \leq 0 \vdash \Delta
\]

\[
\Gamma, t \leq 0 \vdash \Delta \quad \text{\textsc{strengthen'}}
\]

Three sound & complete ways . . .
1. Method: only do pure strengthening

### Pure STRENGTHEN

<table>
<thead>
<tr>
<th>Rule</th>
<th>Premise</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma, t \doteq 0 [t \doteq 0] \vdash \Delta \triangleright l )</td>
<td>( \Gamma, t + 1 \leq 0 [t + 1 \leq 0] \vdash \Delta \triangleright J )</td>
<td>STRENGTHEN-L</td>
</tr>
<tr>
<td>( \Gamma, t \leq 0 [t \leq 0] \vdash \Delta \triangleright l \lor J )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \Gamma, t \doteq 0 [0 \doteq 0] \vdash \Delta \triangleright l )</td>
<td>( \Gamma, t + 1 \leq 0 [0 \leq 0] \vdash \Delta \triangleright J )</td>
<td>STRENGTHEN-R</td>
</tr>
<tr>
<td>( \Gamma, t \leq 0 [0 \leq 0] \vdash \Delta \triangleright l \land J )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Resembles Omega test
Can lead to large proofs, but interpolants of linear size
Integration with Simplex in [LPAR, 2010]
1. Method: only do pure strengthening

**Pure STRENGTHEN**

\[
\begin{align*}
\Gamma, t \doteq 0 [ t \doteq 0 ] & \vdash \Delta \triangleright I \\
\Gamma, t + 1 \leq 0 [ t + 1 \leq 0 ] & \vdash \Delta \triangleright J \\
\hline
\Gamma, t \leq 0 [ t \leq 0 ] & \vdash \Delta \triangleright I \lor J & \text{STRENGTHEN-L}
\end{align*}
\]

\[
\begin{align*}
\Gamma, t \doteq 0 [ 0 \doteq 0 ] & \vdash \Delta \triangleright I \\
\Gamma, t + 1 \leq 0 [ 0 \leq 0 ] & \vdash \Delta \triangleright J \\
\hline
\Gamma, t \leq 0 [ 0 \leq 0 ] & \vdash \Delta \triangleright I \land J & \text{STRENGTHEN-R}
\end{align*}
\]

- Resembles Omega test
- Can lead to large proofs, but interpolants of linear size
- Integration with Simplex in [LPAR, 2010]
  ⇒ Special branch-and-cut rule
Interpolating proof for previous example

\[
\begin{align*}
\ast & \quad \ldots, \ 3 \leq 0 [6x \leq 0] \vdash \ x \leq 0 \\
\ast & \quad \ldots, \ 3x \leq 0 [3x \leq 0], \ -2x + 1 \leq 0 [0 \leq 0] \vdash \ x \leq 0 \\
\ast & \quad \ldots, \ 3x - 2 \leq 0 [3x - 2 \leq 0], \ -2x + 1 \leq 0 [0 \leq 0] \vdash \ x \leq 0 \\
& \quad a + x \leq 0 [a + x \leq 0], \\
& \quad -a + 2x - 2 \leq 0 [-a + 2x - 2 \leq 0], \vdash \ x \leq 0 \\
& \quad -2x + 1 \leq 0 [0 \leq 0]
\end{align*}
\]

Original proof

\[
\begin{align*}
\ast & \quad \ldots, \ 3 \leq 0 \vdash \ \text{INEQ-CLOSE'} \\
\ast & \quad \ldots, \ 3x \leq 0, \ -2x + 1 \leq 0 \vdash \ \text{FM-ELIM'} \quad \ldots \\
\ast & \quad \ldots, \ 3x - 2 \leq 0, \ -2x + 1 \leq 0 \vdash \ \text{STRENGTHEN'} \times 2 \\
& \quad a + x \leq 0, \ -a + 2x - 2 \leq 0, \ -2x + 1 \leq 0 \vdash \ \text{FM-ELIM'}
\end{align*}
\]
2. Method: allow **mixed strengthening**

General, mixed **STRENGTHEN** ("mixed cuts")

\[
\begin{align*}
\Gamma, t \leq 0 \left[ t^A \leq 0 \right] & \vdash \Delta \triangleright E \\
\Gamma, t + 1 \leq 0 \left[ t^A \leq 0 \right] & \vdash \Delta \triangleright l^0 \\
\Gamma, t + 1 \leq 0 \left[ t^A + 1 \leq 0 \right] & \vdash \Delta \triangleright l^1 \\
\Gamma, t \leq 0 \left[ t^A \leq 0 \right] & \vdash \Delta \triangleright l^1 \vee (E \land l^0) \quad \text{STRENGTHEN}
\end{align*}
\]
2. Method: allow mixed strengthening

General, mixed \texttt{STRENGTHEN} ("mixed cuts")

\[
\begin{align*}
\Gamma, t \leq 0 \left[ t^A \leq 0 \right] & \Rightarrow \Delta \triangleright E \\
\Gamma, t + 1 \leq 0 \left[ t^A \leq 0 \right] & \Rightarrow \Delta \triangleright I^0 \\
\Gamma, t + 1 \leq 0 \left[ t^A + 1 \leq 0 \right] & \Rightarrow \Delta \triangleright I^1 \\
\Gamma, t \leq 0 \left[ t^A \leq 0 \right] & \Rightarrow \Delta \triangleright I^1 \lor (E \land I^0) \quad \text{STRENGTHEN}
\end{align*}
\]

- Covers Omega test, Gomory cuts, etc.
- Interpolants can be exponentially larger than (non-interpolating) proofs
- Sometimes observed in practice:
  Proof can be constructed, but proof lifting times out
3. Method: bounded quantification

**STRENGTHEN** with bounded quantification
3. Method: bounded quantification

\[
\begin{align*}
\Gamma, t \div 0 \left[ t^A \div 0 \right] & \vdash \Delta \triangleright E \\
\Gamma, t + 1 \leq 0 \left[ t^A \leq 0 \right] & \vdash \Delta \triangleright l^0 \\
\Gamma, t + 1 \leq 0 \left[ t^A + 1 \leq 0 \right] & \vdash \Delta \triangleright l^1 \\
\Gamma, t \leq 0 \left[ t^A \leq 0 \right] & \vdash \Delta \triangleright l^1 \lor (E \land l^0)
\end{align*}
\]

STRENGTHEN with bounded quantification
3. Method: bounded quantification

\[ \Gamma, t \leq 0 [t^A \leq 0] \vdash \Delta \triangleright E \]
\[ \Gamma, t + 1 \leq 0 [t^A \leq 0] \vdash \Delta \triangleright I^0 \]
\[ \Gamma, t + 1 \leq 0 [t^A + 1 \leq 0] \vdash \Delta \triangleright I^1 \]
\[ \Gamma, t \leq 0 [t^A \leq 0] \vdash \Delta \triangleright I^1 \lor (E \land I^0) \]

**STRENGTHEN with bounded quantification**
3. Method: bounded quantification

\[ \Gamma, t \leq 0 \left[ t^A \leq 0 \right] \vdash \Delta \triangleright E \]
\[ \Gamma, t + 1 \leq 0 \left[ t^A + p \leq 0 \right] \vdash \Delta \triangleright l(p) \]
\[ \Gamma, t \leq 0 \left[ t^A \leq 0 \right] \vdash \Delta \triangleright l(1) \lor (E \land l(0)) \]

STRENGTHEN with bounded quantification

Proofs + interpolants only grow linearly
Interpolants contain bounded quantifiers
Specialised versions for rounding possible [IJCAR, 2010]
Related observation in [Griggio, Le, Sebastiani, 2011]:
Mixed cuts can be interpolated concisely using integer division
3. Method: bounded quantification

**STRENGTHEN with bounded quantification**

\[
\begin{align*}
\Gamma, t \div 0[t^A \div 0] & \vdash \Delta \triangleright E \\
\Gamma, t + 1 \leq 0[t^A + p \leq 0] & \vdash \Delta \triangleright l(p) \\
\Gamma, t \leq 0[t^A \leq 0] & \vdash \Delta \triangleright \exists 0 \leq p \leq 1.
\quad l(p) \land (p \div 1 \lor E)
\end{align*}
\]
3. Method: bounded quantification

**STRENGTHEN with bounded quantification**

\[
\Gamma, t \doteq 0 [t^A \doteq 0] \vdash \Delta \triangleright E \\
\Gamma, t + 1 \leq 0 [t^A + p \leq 0] \vdash \Delta \triangleright I(p) \\
\Gamma, t \leq 0 [t^A \leq 0] \vdash \Delta \triangleright 0 \leq p \leq 1. \\
I(p) \land (p \doteq 1 \lor E)
\]

- Proofs + interpolants only grow linearly
- Interpolants contain bounded quantifiers
- Specialised versions for rounding possible [IJCAR, 2010]
- Related observation in [Griggio, Le, Sebastiani, 2011]: Mixed cuts can be interpolated concisely using integer division
Combining method 2 + 3

Observation, in practice:
QE methods often eliminate bounded quantifiers \textit{without blowup}
Combining method 2 + 3

Observation, in practice:
QE methods often eliminate bounded quantifiers without blowup

Implication $A \rightarrow C$

Theorem prover

Proof of $A \rightarrow C$

Proof lifting (method 2)

Quantifier-free interpolant $I$
Combining method 2 + 3

Observation, in practice:
QE methods often eliminate bounded quantifiers without blowup

Implication $A \rightarrow C$

Proof of $A \rightarrow C$

Proof lifting (method 2)

Quantifier-free interpolant $I$

Quantifier elimination

Proof lifting (method 3)

Interpolant $I'$ with bounded quantifiers
Ongoing work

Elimination of bounded quantifiers …

- Often leads to very concise interpolants
- Sometimes causes blowup (e.g., when encoding bitvector problems)

Ongoing: better integration of methods 2 + 3

- Detect in which cases bounded quantifiers are cheap to eliminate

Also ongoing:

- Experimental comparison of methods 1+2+3, in a model checker
Implementations
## Implementations

### Method 1: OpenSMT [LPAR, 2010]
- Simplex-based
- Branch-and-cut rule, avoiding mixed cuts

### Method 2 + 3: Princess [IJCAR, 2010]
- Omega-based
- First interpolants with **bounded quantifiers**, then **quantifier elimination** (Omega)
About PRINCESS

- Started in 2007, slowly moving along
  (name “PRINCESS” → complicated explanation)
- Entirely implemented in Scala

- Original motivation:
  Explore combination of FOL + theory reasoning
- Input logic:
  QPA + uninterpreted predicates/functions
Combination of different prover architectures

Experiment in Princess:

- KE-tableau/DPLL
- Theory procedures
- E-matching
- Free variables + constraints

- Interesting completeness results
- Proof generation (used for interpolation)
- Some features that are rather unique
(In)Completeness of the PRINCESS calculus

Lemma (Completeness)

Complete for fragments:

- $\hat{\text{FOL}}$
- $\hat{\text{PA}}$
- Purely existential formulae
- Purely universal formulae
- Universal formulae with finite parametrisation (same as $\mathcal{ME}(\text{LIA})$)

- Valid formulae in the full logic are not enumerable
  
  [Halpern, 1991]
About Scala

Java + functional features

- Algebraic datatypes
- Pattern matching
- Type inference
- Higher-order functions
- Monads
- Actors, concurrent datatypes

- Developed by Martin Odersky’s group, EPFL
- Compilation to Java bytecode (primarily)
- Full access to Java libraries
# About Scala

<table>
<thead>
<tr>
<th>Java + functional features</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Algebraic datatypes</td>
</tr>
<tr>
<td>• Pattern matching</td>
</tr>
<tr>
<td>• Type inference</td>
</tr>
<tr>
<td>• Higher-order functions</td>
</tr>
<tr>
<td>• Monads</td>
</tr>
<tr>
<td>• Actors, concurrent datatypes</td>
</tr>
</tbody>
</table>

- Developed by Martin Odersky’s group, EPFL
- Compilation to Java bytecode (primarily)
- Full access to Java libraries

- Is Scala a language usable for solver implementations?
Some observations and thoughts on Scala . . .
Elegant APIs possible

val c = new ConstantTerm("c")
val d = new ConstantTerm("d")

val f = new IFunction("f", 1, false, false)

println(isSat(c >= 12 & c*2 < 40 & f(c-d) < 100))
Elegant APIs possible

```scala
val c = new ConstantTerm("c")
val d = new ConstantTerm("d")

val f = new IFunction("f", 1, false, false)

println(isSat(c >= 12 & c*2 < 40 & f(c-d) < 100))
```

Maybe more relevant for solver users than developers
Deployment

- Bytecode is very convenient

- However: Scala tends to generate many many classes
  E.g. in PRINCESS: before compilation \( \approx 350 \)
    after compilation \( \approx 3000 \)

- ProGuard (compression tool) is useful
  \( \Rightarrow \) Generate one jar file, including all Scala libraries
**Princess** was not developed in a very performance-oriented way:

- Mostly functional (immutable) datastructures
- No native datastructures (JNI)
- Generally:
  - Correctness considered more important than efficiency
## Compared to other languages (Compiler shootout)

| compare 2          | |---| |---| | 25% | median | 75% | ---| | ---| | ---| |
|-------------------|---|---|---|---|---|---|---|---|---|
| **Fortran Intel** | 1.00 | 1.00 | 1.00 | **1.00** | 1.49 | 2.24 | 5.15 |
| **C++ GNU g++**   | 1.00 | 1.00 | 1.03 | **1.27** | 1.68 | 2.65 | 4.06 |
| **C GNU gcc**     | 1.00 | 1.00 | 1.00 | **1.30** | 1.47 | 2.17 | 3.24 |
| **ATS**           | 1.01 | 1.01 | 1.17 | **1.37** | 1.60 | 2.24 | 7.95 |
| **Ada 2005 GNAT** | 1.01 | 1.01 | 1.35 | **1.62** | 2.44 | 4.08 | 7.55 |
| **Java 7 -server**| 1.11 | 1.11 | 1.58 | **1.76** | 2.02 | 2.68 | 5.62 |
| **Scala**         | 1.24 | 1.24 | 1.81 | **2.18** | 3.22 | 5.33 | 9.79 |
| **Pascal Free Pascal** | 1.38 | 1.38 | 2.01 | **2.39** | 2.84 | 4.10 | 5.36 |
| **Haskell GHC**   | 1.10 | 1.10 | 1.64 | **2.64** | 4.34 | 8.38 | 8.73 |
| **C# Mono**       | 1.44 | 1.44 | 2.26 | **2.72** | 5.46 | 10.26 | 20.52 |
| **Clean**         | 1.76 | 1.76 | 2.11 | **3.01** | 4.15 | 7.21 | 11.17 |
| **OCaml**         | 1.55 | 1.55 | 2.02 | **3.40** | 4.90 | 6.26 | 6.26 |
| **Lisp SBCL**     | 1.02 | 1.02 | 1.87 | **3.81** | 4.99 | 9.67 | 10.87 |
| **F# Mono**       | 1.43 | 1.43 | 2.53 | **3.97** | 5.62 | 10.24 | 18.28 |
| **Racket**        | 1.17 | 2.16 | 4.19 | **4.79** | 5.55 | 7.58 | 18.58 |
JVM warm-up
JVM warm-up (2)

Caused by:
- Dynamic class loading
- Just-in-time compilation + optimisation

This means:
- Restarting solver between queries has to be avoided
- Load solver as a library (jar-file), or
- Run as a daemon
### Evaluation on AUFLIA benchmarks

<table>
<thead>
<tr>
<th></th>
<th>AUFLIA+p (193)</th>
<th>AUFLIA-p (193)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z3</td>
<td>191</td>
<td>191</td>
</tr>
<tr>
<td>Princess</td>
<td>145</td>
<td>137</td>
</tr>
<tr>
<td>CVC3</td>
<td>132</td>
<td>128</td>
</tr>
</tbody>
</table>

- Unsatisfiable AUFLIA benchmarks from SMT-comp 2011
- Intel Core i5 2-core, 3.2GHz, timeout 1200s, 4Gb
Typical PA SAT queries in a model checker (Eldarica)
Profiling Scala applications

Does not work
Synthetic interpolation benchmarks (beginning 2011)

- Evaluation on SMT-LIB QF_LIA benchmarks
- Partitionings:
  First $\frac{k}{10} \cdot n$ benchmark conjuncts as $A$, rest as $B$
  (where $n$ is total number of conjuncts, $k \in \{1, \ldots, 9\}$)

- Intel Xeon X5667 4-core, 3.07GHz, 12GB heap-space, Linux, timeout 900s.

http://www.philipp.ruemmer.org/princess.shtml
Compared tools

- **Princess, OpenSMT**
- **SMTInterpol**: interpolating SMT solver from Uni Freiburg
- **CSIsat**: constraint-based interpolation for linear rational arithmetic + unint. functions
- **Omega**: quantifier elimination procedure (strongest interpolants can be computed using QE)
## Experimental results

<table>
<thead>
<tr>
<th></th>
<th>Multiplier</th>
<th>Bitadder</th>
<th>Mathsat</th>
<th>Rings</th>
<th>Convert</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16 unsat</td>
<td>17 unsat</td>
<td>100 unsat</td>
<td>294 unsat</td>
<td>38 unsat</td>
</tr>
<tr>
<td></td>
<td>1 sat</td>
<td></td>
<td></td>
<td></td>
<td>109 sat</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>172 unkn.</td>
</tr>
<tr>
<td>Princess</td>
<td>8/1/41</td>
<td>7/0/63</td>
<td>44/13/396</td>
<td>130/0/209</td>
<td>38/82/334</td>
</tr>
<tr>
<td></td>
<td>136/1623</td>
<td>298/76953</td>
<td>106/7007</td>
<td>233/5146</td>
<td>88.0/1</td>
</tr>
<tr>
<td>OpenSMT</td>
<td>5/1/45</td>
<td>7/0/63</td>
<td>74/15/666</td>
<td>9/0/81</td>
<td>37/0/333</td>
</tr>
<tr>
<td></td>
<td>48.9/2357</td>
<td>103/23362</td>
<td>53.0/2020</td>
<td>59.9/4611</td>
<td>0.08/1</td>
</tr>
<tr>
<td>SMTInterpol</td>
<td>5/1/45</td>
<td>5/0/45</td>
<td>65/13/585</td>
<td>0/0/-</td>
<td>37/0/333</td>
</tr>
<tr>
<td></td>
<td>24.4/48827</td>
<td>8.58/41077</td>
<td>45.7/126705</td>
<td>-/-</td>
<td>13.6/2</td>
</tr>
</tbody>
</table>
| CSIsat   | 4/1/36     | 1/0/9    | 25/12/225 | -/- | -/-
|          | 106/2640   | 0.56/188 | 70.8/12683 | -/- | -/-
| Omega QE | -/-/125    | -/-/129  | -/-/612 | -/-/1474 | -/-/296 |
|          | 109/15392  | 97.8/93181 | 169/101088 | 227/55307 | 15.4/2668 |

| #unsat / #sat / #interpolants / average time (s) / average int. size |
Experimental results: interpolant sizes
Experimental results: interpolant sizes (2)
Conclusions

Is Scala a language usable for solver implementations?

**Pros**
- Deployment
- Very elegant APIs possible
- Convenient

**Cons**
- Warm-up time of JVM
- Performance penalty still significant
Thanks for your attention!