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

$$\frac{\texttt{pre} \Rightarrow \phi \qquad \{\ \phi\ \}\ \texttt{Body}\ \{\ \phi\ \} \qquad \phi \Rightarrow \texttt{post}}{\{\ \texttt{pre}\ \}\ \texttt{while (*) Body}\ \{\ \texttt{post}\ \}}$$

How to compute ϕ automatically?

From intermediate assertions to invariants

$\{\texttt{pre}\}\ \texttt{Body; Body}\ \{\texttt{post}\}$?

Bounded model checking problem ✓

Compute intermediate assertion ψ_1

$\{\texttt{pre}\}\ \texttt{Body}\ \{\psi_1\}$ $\{\psi_1\}\ \texttt{Body}\ \{\texttt{post}\}$

[McMillan, 2003]

From intermediate assertions to invariants

$\{\texttt{pre}\}$ Body; Body $\{\texttt{post}\}$?

$\Downarrow$

Bounded model checking problem ✓

$\Downarrow$

Compute intermediate assertion ψ_1

$\Downarrow$

$\{\texttt{pre}\}$ Body $\{\psi_1\}$ $\{\psi_1\}$ Body $\{\texttt{post}\}$

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

pre is invariant ✓

[McMillan, 2003]

From intermediate assertions to invariants

{pre} Body; Body {post} ?

Bounded model checking problem ✓

Compute intermediate assertion ψ_1

{pre} Body $\{\psi_1\}$ | $\{\psi_1\}$ Body {post}

[$\psi_1 \Rightarrow$ pre]

pre is invariant ✓

[otherwise]

[McMillan, 2003]

From intermediate assertions to invariants

$\{\texttt{pre} \lor \psi_1\}$ Body; Body $\{\texttt{post}\}$?

$\downarrow$

Bounded model checking problem ✓

$\downarrow$

Compute intermediate assertion ψ_2

$\downarrow$

$\{\texttt{pre} \lor \psi_1\}$ Body $\{\psi_2\}$ $\quad$ $\{\psi_2\}$ Body $\{\texttt{post}\}$

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

$\texttt{pre}$ is invariant ✓

[otherwise]

[McMillan, 2003]

From intermediate assertions to invariants

$\{\texttt{pre} \vee \psi_1\}$ Body; Body $\{\texttt{post}\}$?

Bounded model checking problem ✓

Compute intermediate assertion ψ_2

$\{\texttt{pre} \vee \psi_1\}$ Body $\{\psi_2\}$

$\{\psi_2\}$ Body $\{\texttt{post}\}$

$[\psi_2 \Rightarrow \texttt{pre} \vee \psi_1]$

$\texttt{pre} \vee \psi_1$ is invariant ✓

[otherwise]

[McMillan, 2003]

From intermediate assertions to invariants

$\{\texttt{pre} \vee \psi_1\}$ Body; Body $\{\texttt{post}\}$?

Bounded model checking problem ✓

Compute intermediate assertion ψ_2

$\{\texttt{pre} \vee \psi_1\}$ Body $\{\psi_2\}$ $\{\psi_2\}$ Body $\{\texttt{post}\}$

$[\psi_2 \Rightarrow \texttt{pre} \vee \psi_1]$

$\texttt{pre} \vee \psi_1$ is invariant ✓

...

[McMillan, 2003]

How to compute intermediate assertions?

VC generation

$$\begin{array}{ll}
\{\ \texttt{pre}\ \} & \qquad\quad \texttt{pre}\ (s_0) \\
\ \texttt{Body;} & \rightarrow \texttt{Body}\,(s_0, s_1) \\
\ \texttt{Body} & \rightarrow \texttt{Body}\,(s_1, s_2) \\
\{\ \texttt{post}\ \} & \rightarrow \texttt{post}\,(s_2)
\end{array}$$

How to compute intermediate assertions?

VC generation

`{ pre }`	`pre` (s_0)
`Body;`	$\rightarrow$ `Body` (s_0, s_1)
`Body`	$\rightarrow$ `Body` (s_1, s_2)
`{ post }`	$\rightarrow$ `post` (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.*

How to compute intermediate assertions?

VC generation

{ `pre` }	`pre` (s_0)	$A(s_0, s_1)$
`Body;`	$\rightarrow$ `Body` (s_0, s_1)	$\downarrow$
		$I(s_1)$
`Body`	$\rightarrow$ `Body` (s_1, s_2)	$\downarrow$
{ `post` }	$\rightarrow$ `post` (s_2)	$C(s_1, s_2)$

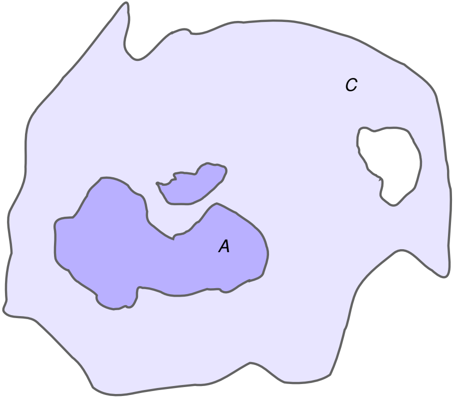
Theorem (Craig, 1957)

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.

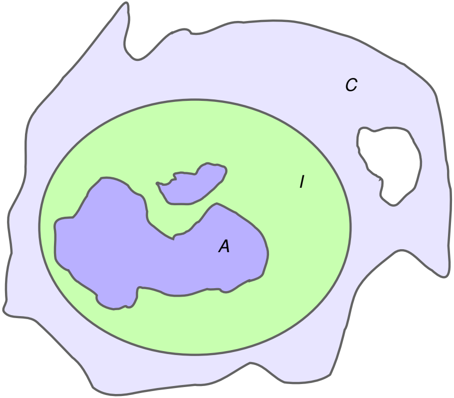
Illustration

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



Illustration

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



Example

Program with assertion:

```
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:

```
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
->                          // Interpolant: 3*b >= a
c = 3*b + 1
->                          // Interpolant: c >= a + 1
c > a
```

Other applications of interpolation

- Blocking lemmas for test-case generation
- Refinement of abstractions in CEGAR
- Computation of summaries
- Synthesis

Interpolation + theories

Interpolation procedures need to support the program logic:

```
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)

$$
\begin{aligned}
t \;&::=\; \alpha \;\big|\; c \;\big|\; x \;\big|\; \alpha t + \cdots + \alpha t \\
\phi \;&::=\; \phi \wedge \phi \;\big|\; \phi \vee \phi \;\big|\; \neg \phi \;\big|\; \phi \to \phi \;\big|\; \forall x.\phi \;\big|\; \exists x.\phi \\
&\;\;\;\;\;\big|\; t \doteq 0 \;\big|\; t \leq 0 \;\big|\; \alpha \mid t
\end{aligned}
$$

t	...	terms
ϕ	...	formulae
x	...	variables
c	...	constant symbols
α	...	integer literals ($\mathbb{Z}$)

Interpolation for integers

Presburger Arithmetic (QPA)

$$
\begin{aligned}
t \;&::=\; \alpha \;\Big|\; c \;\Big|\; x \;\Big|\; \alpha t + \cdots + \alpha t \\
\phi \;&::=\; \phi \wedge \phi \;\Big|\; \phi \vee \phi \;\Big|\; \neg\phi \;\Big|\; \phi \to \phi \;\Big|\; \forall x.\phi \;\Big|\; \exists x.\phi \\
&\quad\;\Big|\; t \doteq 0 \;\Big|\; t \leq 0 \;\Big|\; \alpha \mid t
\end{aligned}
$$

Mainly considered here: the quantifier-free fragment (PA)

Interpolation by quantifier elimination (QE)

Theorem (QE for Presburger Arithmetic)

For every formula ϕ in full QPA, there is an equivalent quantifier-free formula ψ that can effectively be computed.

Interpolation by quantifier elimination (2)

Lemma

If $A \rightarrow C$ is a valid implication, then

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

$\textit{local-syms}(A)$: symbols occurring in A, but not in C
$\textit{local-syms}(C)$: ...

Corollary

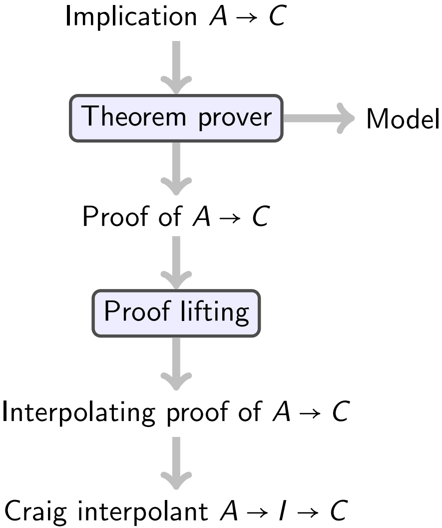
Both PA and QPA are closed under interpolation.

Interpolation vs. QE

However . . .

- QE has high computational complexity
- strongest and weakest interpolants are often not needed/desirable
  $\Rightarrow$ Larger interpolants, containing irrelevant information

Proof-based interpolation techniques



Abstraction with interpolants

{pre} Body; Body {post} ?

↓

Bounded model checking problem ✓

↓

Compute intermediate assertion ψ_1

↓

...

Abstraction with interpolants

{pre} Body; Body {post} ?

↓

Bounded model checking problem ✓

↓

Compute intermediate assertion ψ_1

↓

...

Interpolant extracted from proof
⇒
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?

Reverse interpolants

Definition

Suppose $A \wedge B$ 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 $A \wedge B$

$$\Longleftrightarrow$$

I is interpolant for $A \rightarrow \neg B$

What makes interpolation over integers difficult?

Consider rational case: $$\underbrace{\bigwedge_{i=1}^{n} t_i \leq 0}_{A} \quad \wedge \quad \underbrace{\bigwedge_{j=1}^{m} s_j \leq 0}_{B}$$

What makes interpolation over integers difficult?

Consider rational case: $\underbrace{\bigwedge_{i=1}^{n} t_i \leq 0}_{A} \quad \wedge \quad \underbrace{\bigwedge_{j=1}^{m} s_j \leq 0}_{B}$

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:

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

What makes interpolation over integers difficult?

Consider rational case: $$\underbrace{\bigwedge_{i=1}^{n} t_i \le 0}_{A} \quad \wedge \quad \underbrace{\bigwedge_{j=1}^{m} s_j \le 0}_{B}$$

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:

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

Then:

$$\sum_{i=1}^{n} \alpha_i t_i \le 0 \quad \text{is a reverse interpolant}$$

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 $\{A_n \wedge B_n\}_n$ of PA formulae such that

- *$A_n \wedge B_n$ is unsatisfiable,*
- *$A_n \wedge B_n$ 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:

$$A_n \;=\; -n < y + 2nx \;\wedge\; y + 2nx \leq 0$$
$$B_n \;=\; 0 < y + 2nz \;\wedge\; y + 2nz \leq n$$

All reverse interpolants for $A_n \wedge B_n$ are equivalent to:

$$I_n \;=\; (2n \mid y) \vee (2n \mid y + 1) \vee (2n \mid y + 2) \vee \cdots \vee (2n \mid y + n - 1)$$

What makes interpolation over integers difficult? (4)

Example:

$$
\begin{aligned}
A_n &= -n < y + 2nx \ \wedge \ y + 2nx \leq 0 \\
B_n &= 0 < y + 2nz \ \wedge \ y + 2nz \leq n
\end{aligned}
$$

All reverse interpolants for $A_n \wedge B_n$ are equivalent to:

$$
I_n = \big(2n \mid y\big) \vee \big(2n \mid y+1\big) \vee \big(2n \mid y+2\big) \vee \cdots \vee \big(2n \mid y+n-1\big)
$$

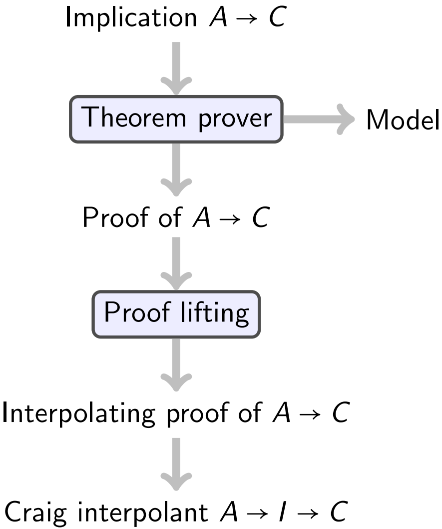
Problematic: mixed cuts

Three main approaches to handle mixed cuts

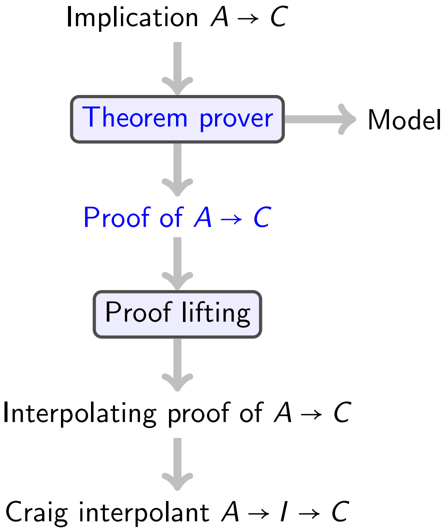
- Fully expanded interpolants
- Restricted/taylor-made cut rule
- Extended interpolant language

Next:
Comparison + unifying description

Interpolation outline



Interpolation outline



Main non-interpolating proof rules

Closure rule ($\alpha > 0$)

$$\frac{*}{\Gamma, \alpha \leq 0 \ \vdash \ \Delta} \ \textsc{close-ineq}'$$

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

$$\frac{\Gamma, \ \ldots, \alpha s + \beta t \leq 0 \ \vdash \ \Delta}{\Gamma, \ s \leq 0, \ t \leq 0 \ \vdash \ \Delta} \ \textsc{fm-elim}'$$

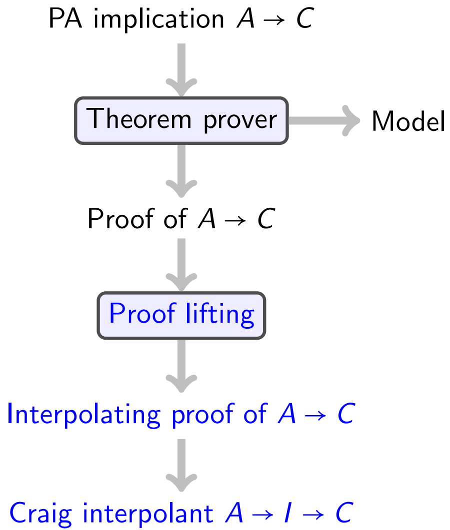
Strengthening inequalities (subsumes rounding + cuts)

$$\frac{\Gamma, t \doteq 0 \ \vdash \ \Delta \qquad \Gamma, t + 1 \leq 0 \ \vdash \ \Delta}{\Gamma, t \leq 0 \ \vdash \ \Delta} \ \textsc{strengthen}'$$

Example of non-interpolating proof

$$
\dfrac{\dfrac{\dfrac{\dfrac{*}{\ldots,\ 3 \le 0\ \vdash}\ \textsc{ineq-close}'}{\ldots,\ 3x \le 0,\ -2x + 1 \le 0\ \vdash}\ \textsc{fm-elim}' \qquad \cdots}{\ldots,\ 3x - 2 \le 0,\ -2x + 1 \le 0\ \vdash}\ \textsc{strengthen}' \times 2}{a + x \le 0,\ -a + 2x - 2 \le 0,\ -2x + 1 \le 0\ \vdash}\ \textsc{fm-elim}'
$$

Interpolation outline



Basic idea of proof lifting

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

$$
\begin{array}{c}
*\\
\vdots\\
\dfrac{\dfrac{\Gamma_3 \ \vdash\ \Delta_3}{\Gamma_2 \ \vdash\ \Delta_2}}{\Gamma_1 \ \vdash\ \Delta_1}\\
\vdots\\
A \ \vdash\ C
\end{array}
$$

Basic idea of proof lifting

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

annotation of
formulae with labels $\uparrow$

$$\dfrac{\dfrac{\begin{array}{c} * \\ \vdots \\ \Gamma_3 \vdash \Delta_3 \end{array}}{\Gamma_2 \vdash \Delta_2}}{\begin{array}{c}\Gamma_1 \vdash \Delta_1 \\ \vdots \\ A \vdash C\end{array}}$$

Main idea: annotations track inequalities from A

Basic idea of proof lifting

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

annotation of formulae with labels $\uparrow$

$$\dfrac{\dfrac{\begin{array}{c} * \\ \vdots \\ \Gamma_3 \vdash \Delta_3 \end{array}}{\Gamma_2 \vdash \Delta_2}}{\begin{array}{c} \Gamma_1 \vdash \Delta_1 \\ \vdots \\ A^* \vdash C^* \end{array}}$$

Main idea: annotations track inequalities from A

Basic idea of proof lifting

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

annotation of formulae with labels $\uparrow$

$$
\dfrac{\dfrac{\begin{array}{c} * \\ \vdots \\ \Gamma_3 \ \vdash \ \Delta_3 \end{array}}{\Gamma_2 \ \vdash \ \Delta_2}}{\begin{array}{c} \Gamma_1^* \ \vdash \ \Delta_1^* \\ \vdots \\ A^* \ \vdash \ C^* \end{array}}
$$

Main idea: annotations track inequalities from A

Basic idea of proof lifting

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

annotation of formulae with labels $\uparrow$

$$
\cfrac{\cfrac{\begin{array}{c} * \\ \vdots \\ \Gamma_3 \vdash \Delta_3 \end{array}}{\Gamma_2^* \vdash \Delta_2^*}}{\begin{array}{c} \Gamma_1^* \vdash \Delta_1^* \\ \vdots \\ A^* \vdash C^* \end{array}}
$$

Main idea: annotations track inequalities from A

Basic idea of proof lifting

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

annotation of formulae with labels $\uparrow$

$$
\dfrac{\dfrac{\dfrac{\vdots^{*}}{\Gamma_3^* \vdash \Delta_3^*}}{\Gamma_2^* \vdash \Delta_2^*}}{\dfrac{\Gamma_1^* \vdash \Delta_1^*}{\vdots}}
$$

$$A^* \vdash C^*$$

Main idea: annotations track inequalities from A

Basic idea of proof lifting

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

$$
\begin{array}{c}
* \\
\vdots \\
\dfrac{\dfrac{\Gamma_3^* \vdash \Delta_3^*}{\Gamma_2^* \vdash \Delta_2^*}}{\Gamma_1^* \vdash \Delta_1^*} \\
\vdots \\
A^* \vdash C^*
\end{array}
$$

annotation of formulae with labels $\uparrow$

$\downarrow$ propagation of interpolants

Main idea: annotations track inequalities from A

Basic idea of proof lifting

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

annotation of formulae with labels $\uparrow$

$$\dfrac{\dfrac{\begin{array}{c} * \\ \vdots \\ \Gamma_3^* \ \vdash \ \Delta_3^* \ \blacktriangleright I_3 \end{array}}{\Gamma_2^* \ \vdash \ \Delta_2^*}}{\Gamma_1^* \ \vdash \ \Delta_1^*}$$
$$\vdots$$
$$A^* \ \vdash \ C^*$$

$\downarrow$ propagation of interpolants

Main idea: annotations track inequalities from A

Basic idea of proof lifting

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

$$\begin{array}{c} * \\ \vdots \\ \Gamma_3^* \vdash \Delta_3^* \ \blacktriangleright I_3 \\ \hline \Gamma_2^* \vdash \Delta_2^* \ \blacktriangleright I_2 \\ \hline \Gamma_1^* \vdash \Delta_1^* \\ \vdots \\ A^* \vdash C^* \end{array}$$

annotation of formulae with labels $\uparrow$

$\downarrow$ propagation of interpolants

Main idea: annotations track inequalities from A

Basic idea of proof lifting

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

annotation of formulae with labels $\uparrow$

$$
\begin{array}{c}
*\\
\vdots\\
\Gamma_3^* \vdash \Delta_3^* \blacktriangleright I_3\\
\hline
\Gamma_2^* \vdash \Delta_2^* \blacktriangleright I_2\\
\hline
\Gamma_1^* \vdash \Delta_1^* \blacktriangleright I_1\\
\vdots\\
A^* \vdash C^*
\end{array}
$$

$\downarrow$ propagation of interpolants

Main idea: annotations track inequalities from A

Basic idea of proof lifting

Interpolation problem: $A \rightarrow {\color{blue} I} \rightarrow C$

$$
\begin{array}{c}
*\\
\vdots\\
\Gamma_3^* \;\vdash\; \Delta_3^* \;\blacktriangleright I_3\\
\hline
\Gamma_2^* \;\vdash\; \Delta_2^* \;\blacktriangleright I_2\\
\hline
\Gamma_1^* \;\vdash\; \Delta_1^* \;\blacktriangleright I_1\\
\vdots\\
A^* \;\vdash\; C^* \;\blacktriangleright {\color{blue} I}
\end{array}
$$

annotation of formulae with labels $\uparrow$

$\downarrow$ propagation of interpolants

Main idea: annotations track inequalities from A

Labelled formulae

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

Labelled formula	**Intuition**
$\phi\,[\phi^A]$	"ϕ^A is A-contribution to ϕ" ϕ^A is the *partial interpolant* of ϕ

Interpolating rules

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

Initialisation rule: $t \leq 0$ comes from A

$$\frac{\Gamma, t \leq 0\,[t \leq 0] \;\vdash\; \Delta \;\blacktriangleright I}{\Gamma, t \leq 0 \;\vdash\; \Delta \;\blacktriangleright I}\ \text{IPI-LEFT-L}$$

Initialisation rule: $t \leq 0$ comes from C

$$\frac{\Gamma, t \leq 0\,[0 \leq 0] \;\vdash\; \Delta \;\blacktriangleright I}{\Gamma, t \leq 0 \;\vdash\; \Delta \;\blacktriangleright I}\ \text{IPI-LEFT-R}$$

- Similarly for equations, etc.

Interpolating rules

Closure rule ($\alpha > 0$)

$$\frac{*}{\Gamma, \alpha \leq 0 \, [t^A \leq 0] \vdash \Delta \blacktriangleright t^A \leq 0} \; \text{CLOSE-INEQ}$$

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

$$\frac{\Gamma, \ldots, \alpha s + \beta t \leq 0 \, [\alpha s^A + \beta t^A \leq 0] \vdash \Delta \blacktriangleright I}{\Gamma, s \leq 0 \, [s^A \leq 0], t \leq 0 \, [t^A \leq 0] \vdash \Delta \blacktriangleright I} \; \text{FM-ELIM}$$

How to interpolate STRENGTHEN′?

$$\frac{\Gamma, t \doteq 0 \;\vdash\; \Delta \qquad \Gamma, t + 1 \leq 0 \;\vdash\; \Delta}{\Gamma, t \leq 0 \;\vdash\; \Delta}\ \textsc{strengthen}'$$

Three sound & complete ways ...

1. Method: only do pure strengthening

Pure STRENGTHEN

$$\frac{\Gamma, t \doteq 0\,[t \doteq 0] \;\vdash\; \Delta \;\blacktriangleright I \qquad \Gamma, t+1 \leq 0\,[t+1 \leq 0] \;\vdash\; \Delta \;\blacktriangleright J}{\Gamma, t \leq 0\,[t \leq 0] \;\vdash\; \Delta \;\blacktriangleright I \vee J}\;\text{STRENGTHEN-L}$$

$$\frac{\Gamma, t \doteq 0\,[0 \doteq 0] \;\vdash\; \Delta \;\blacktriangleright I \qquad \Gamma, t+1 \leq 0\,[0 \leq 0] \;\vdash\; \Delta \;\blacktriangleright J}{\Gamma, t \leq 0\,[0 \leq 0] \;\vdash\; \Delta \;\blacktriangleright I \wedge J}\;\text{STRENGTHEN-R}$$

1. Method: only do pure strengthening

Pure STRENGTHEN

$$\frac{\begin{array}{c}\Gamma, t \doteq 0\,[t \doteq 0] \;\vdash\; \Delta \;\blacktriangleright I \\ \Gamma, t+1 \leq 0\,[t+1 \leq 0] \;\vdash\; \Delta \;\blacktriangleright J\end{array}}{\Gamma, t \leq 0\,[t \leq 0] \;\vdash\; \Delta \;\blacktriangleright I \vee J}\ \text{STRENGTHEN-L}$$

$$\frac{\begin{array}{c}\Gamma, t \doteq 0\,[0 \doteq 0] \;\vdash\; \Delta \;\blacktriangleright I \\ \Gamma, t+1 \leq 0\,[0 \leq 0] \;\vdash\; \Delta \;\blacktriangleright J\end{array}}{\Gamma, t \leq 0\,[0 \leq 0] \;\vdash\; \Delta \;\blacktriangleright I \wedge J}\ \text{STRENGTHEN-R}$$

- Resembles Omega test
- Can lead to large proofs, but interpolants of linear size
- Integration with Simplex in [LPAR, 2010]
  $\Rightarrow$ Special branch-and-cut rule

Interpolating proof for previous example

$$\cfrac{\cfrac{\cfrac{\cfrac{*}{\ldots,\ 3 \le 0\,[6x \le 0] \ \vdash \quad \blacktriangleright\, x \le 0}}{\ldots,\ 3x \le 0\,[3x \le 0],\ -2x+1 \le 0\,[0 \le 0] \ \vdash \quad \blacktriangleright\, x \le 0}}{\ldots,\ 3x-2 \le 0\,[3x-2 \le 0],\ -2x+1 \le 0\,[0 \le 0] \ \vdash \quad \blacktriangleright\, x \le 0}}{\begin{array}{c} a+x \le 0\,[a+x \le 0], \\ -a+2x-2 \le 0\,[-a+2x-2 \le 0], \\ -2x+1 \le 0\,[0 \le 0] \end{array} \ \vdash \quad \blacktriangleright\, x \le 0}$$

Original proof

$$\cfrac{\cfrac{\cfrac{\cfrac{*}{\ldots,\ 3 \le 0 \ \vdash} \ \textsc{ineq-close}'}{\ldots,\ 3x \le 0,\ -2x+1 \le 0 \ \vdash} \ \textsc{fm-elim}' \qquad \ldots}{\ldots,\ 3x-2 \le 0,\ -2x+1 \le 0 \ \vdash} \ \textsc{strengthen}' \times 2}{a+x \le 0,\ -a+2x-2 \le 0,\ -2x+1 \le 0 \ \vdash} \ \textsc{fm-elim}'$$

2. Method: allow mixed strengthening

General, mixed STRENGTHEN ("mixed cuts")

$$\frac{\begin{array}{c}\Gamma, t \doteq 0\,[t^A \doteq 0] \vdash \Delta \blacktriangleright E \\ \Gamma, t+1 \leq 0\,[t^A \leq 0] \vdash \Delta \blacktriangleright I^0 \\ \Gamma, t+1 \leq 0\,[t^A+1 \leq 0] \vdash \Delta \blacktriangleright I^1\end{array}}{\Gamma, t \leq 0\,[t^A \leq 0] \vdash \Delta \blacktriangleright I^1 \vee (E \wedge I^0)}\ \text{STRENGTHEN}$$

2. Method: allow mixed strengthening

General, mixed STRENGTHEN ("mixed cuts")

$$\frac{\begin{array}{c}\Gamma, t \doteq 0\,[t^A \doteq 0] \;\vdash\; \Delta \;\blacktriangleright E \\ \Gamma, t + 1 \leq 0\,[t^A \leq 0] \;\vdash\; \Delta \;\blacktriangleright I^0 \\ \Gamma, t + 1 \leq 0\,[t^A + 1 \leq 0] \;\vdash\; \Delta \;\blacktriangleright I^1\end{array}}{\Gamma, t \leq 0\,[t^A \leq 0] \;\vdash\; \Delta \;\blacktriangleright I^1 \vee (E \wedge I^0)}\ \text{STRENGTHEN}$$

- 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

STRENGTHEN with bounded quantification

$$\frac{\begin{array}{c}\Gamma, t \doteq 0\,[t^A \doteq 0] \;\vdash\; \Delta \;\blacktriangleright E \\ \Gamma, t+1 \leq 0\,[t^A \leq 0] \;\vdash\; \Delta \;\blacktriangleright I^0 \\ \Gamma, t+1 \leq 0\,[t^A + 1 \leq 0] \;\vdash\; \Delta \;\blacktriangleright I^1\end{array}}{\Gamma, t \leq 0\,[t^A \leq 0] \;\vdash\; \Delta \;\blacktriangleright I^1 \vee (E \wedge I^0)}\ \textsc{strengthen}$$

3. Method: bounded quantification

STRENGTHEN with bounded quantification

$$\frac{\begin{array}{c}\Gamma, t \doteq 0\,[t^A \doteq 0] \;\vdash\; \Delta \;\blacktriangleright E \\ \Gamma, t+1 \le 0\,[t^A \le 0] \;\vdash\; \Delta \;\blacktriangleright I^0 \\ \Gamma, t+1 \le 0\,[t^A + 1 \le 0] \;\vdash\; \Delta \;\blacktriangleright I^1\end{array}}{\Gamma, t \le 0\,[t^A \le 0] \;\vdash\; \Delta \;\blacktriangleright I^1 \vee (E \wedge I^0)}\ \textsc{Strengthen}$$

3. Method: bounded quantification

STRENGTHEN with bounded quantification

$$\frac{\Gamma, t \doteq 0\,[t^A \doteq 0] \vdash \Delta \blacktriangleright E \qquad \Gamma, t+1 \leq 0\,[t^A + p \leq 0] \vdash \Delta \blacktriangleright I(p)}{\Gamma, t \leq 0\,[t^A \leq 0] \vdash \Delta \blacktriangleright I(1) \vee (E \wedge I(0))}\ \textsc{strengthen}$$

3. Method: bounded quantification

STRENGTHEN with bounded quantification

$$\frac{\begin{array}{c}\Gamma, t \doteq 0\,[t^A \doteq 0] \;\vdash\; \Delta \;\blacktriangleright E \\ \Gamma, t+1 \leq 0\,[t^A{+}p \leq 0] \;\vdash\; \Delta \;\blacktriangleright I(p)\end{array}}{\Gamma, t \leq 0\,[t^A \leq 0] \;\vdash\; \Delta \;\blacktriangleright \begin{array}{l}\exists\, 0 \leq p \leq 1. \\ \quad I(p) \land (p \doteq 1 \lor E)\end{array}}\ \text{STRENGTHEN-BQ}$$

3. Method: bounded quantification

STRENGTHEN with bounded quantification

$$\frac{\begin{array}{c}\Gamma, t \doteq 0\,[t^A \doteq 0] \;\vdash\; \Delta \;\blacktriangleright E \\ \Gamma, t+1 \leq 0\,[t^A + p \leq 0] \;\vdash\; \Delta \;\blacktriangleright I(p)\end{array}}{\Gamma, t \leq 0\,[t^A \leq 0] \;\vdash\; \Delta \;\blacktriangleright \begin{array}{l}\exists\, 0 \leq p \leq 1. \\ \quad I(p) \wedge (p \doteq 1 \vee E)\end{array}}\ \textsc{strengthen-bq}$$

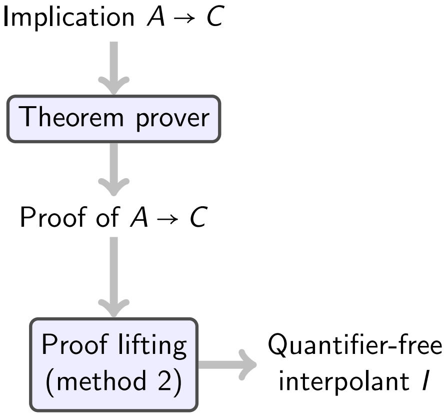
- Proofs + interpolants only grow linearly
- Interpolants contain bounded quantifiers
- Specialised versions for rounding possible [IJCAR, 2010]
- Related observation in [Griggio, Le, Sebastiani, 2011]: Mixed cuts can be interpolated concisely using integer division

Combining method 2 + 3

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

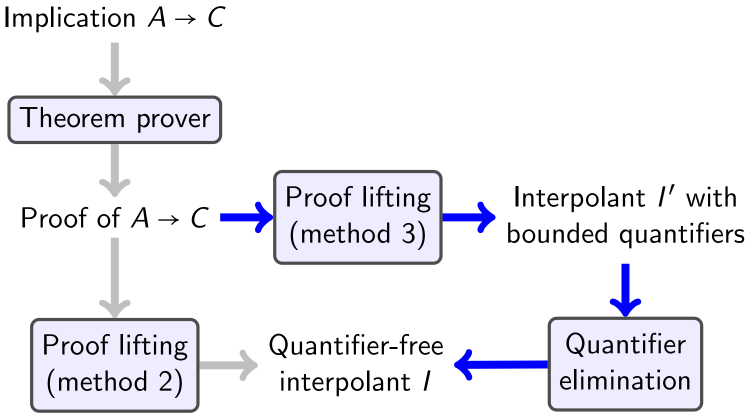
Combining method 2 + 3

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



Combining method 2 + 3

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



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)
- `http://www.philipp.ruemmer.org/princess.shtml`

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**	FOL
• **Theory procedures**	Arithmetic
• **E-matching**	Axiomatisation of theories
• **Free variables + constraints**	Quantifiers

- 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:

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

- 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

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

- 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

```
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 ≈ 350
  after compilation ≈ 3000
- ProGuard (compression tool) is useful
  $\Rightarrow$ Generate one jar file, including all Scala libraries

Performance? (disclaimer)

PRINCESS was not developed in a very performance-oriented way:

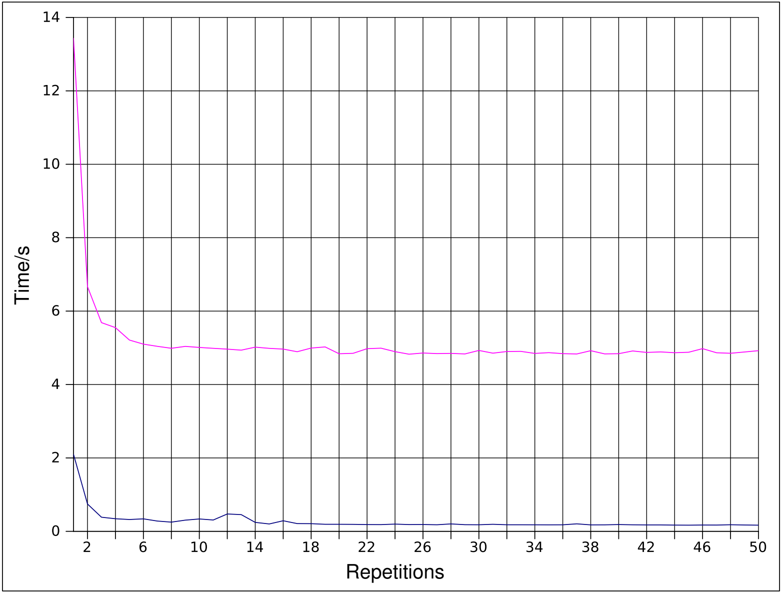
- Mostly functional (immutable) datastructures
- No native datastructures (JNI)
- Generally:
  Correctness considered more important than efficiency

Compared to other languages (Compiler shootout)

chart

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



Time/s
Repetitions
0
2
4
6
8
10
12
14
2
6
10
14
18
22
26
30
34
38
42
46
50

JVM warm-up (2)

Caused by:

- Dynamic class loading
- Just-in-time compilation + optimisation

This means:

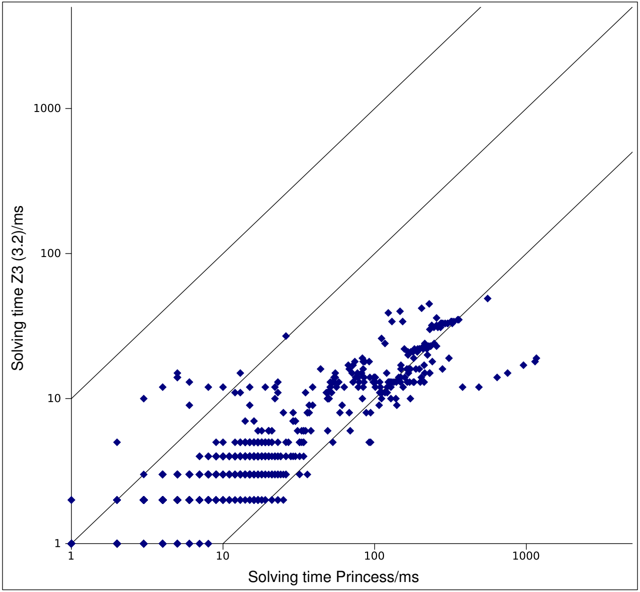
- Restarting solver between queries has to be avoided
- Load solver as a library (jar-file), or
- Run as a daemon

Evaluation on AUFLIA benchmarks

	AUFLIA+p (193)	**AUFLIA-p** (193)
Z3	191	191
Princess	**145**	**137**
CVC3	132	128

- Unsatisfiable AUFLIA benchmarks from SMT-comp 2011
- Intel Core i5 2-core, 3.2GHz, timeout 1200s, 4Gb
- `http://www.philipp.ruemmer.org/princess.shtml`

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`

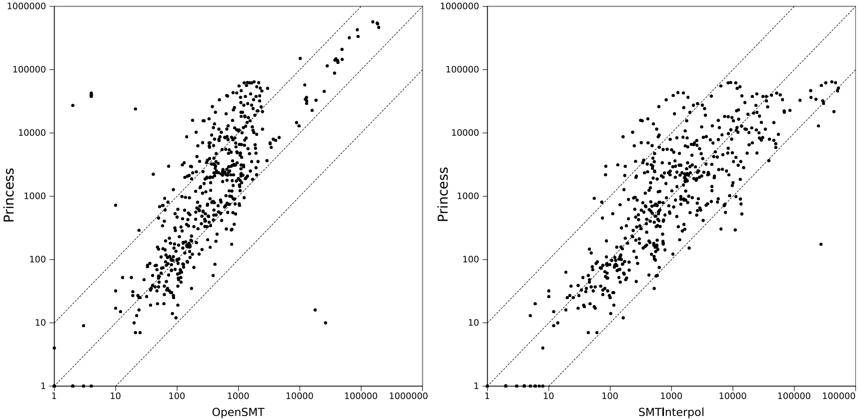
Compared tools

- 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)

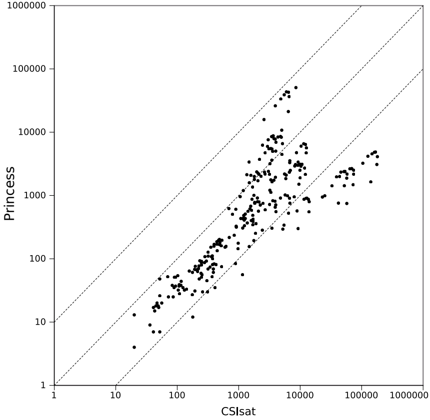
Experimental results

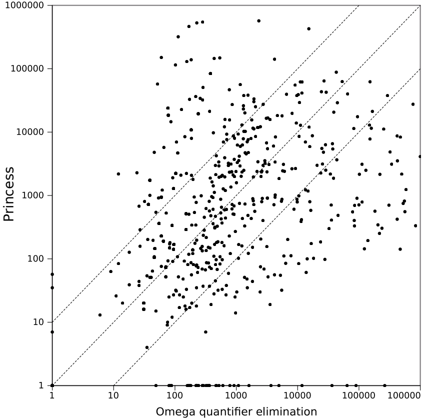
	Multiplier 16 unsat 1 sat	Bitadder 17 unsat	Mathsat 100 unsat	Rings 294 unsat	Convert 38 unsat 109 sat 172 unkn.
Princess	**8**/1/41 136/**1623**	**7**/0/**63** 298/76953	44/13/396 106/7007	**130**/0/**209** 233/5146	**38**/**82**/**334** 88.0/**1**
OpenSMT	5/1/**45** 48.9/2357	**7**/0/**63** 103/**23362**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