# Integration of SMT Solvers with ITPs — There and Back Again

Sascha Böhme and Tjark Weber





ARG Lunch 2 March 2010

Tjark Weber Integration of SMT Solvers with Interactive Theorem Provers

#### 1 Introduction

#### 2 There . . .

- Features: SMT-LIB vs. Yices
- Translation Techniques
- Caveats

#### 3 ... and Back Again

- Z3's Proofs
- LCF-style Theorem Proving
- Reconstruction Techniques
- Performance

#### 4 Conclusions

Motivation System Overview Higher-Order Logic

## Motivation

HOL4 and Isabelle/HOL are popular interactive theorem provers.

Interactive theorem proving benefits from automation.

000	📉 isabelle Proof General: Fib.thy	
Eile Edit View Cads D	ols Options Buffers Proof-Ceneral X-Symbol Isabelle	He1
😟 🛄 🧵 💆 関		
F1b.thy +1sabe11	4	
textErNe disable declare fib.Suc_Sa	Effect fib Suc Suc3 for simplification*3	-
text["then pro- lemma fib_Suc3: "- by (rule fib.Su	re a version that has a more restrictive pattern.*3 "16 (Suc (Suc (Suc n))) = fib (Suc n) + fib (Suc (Suc n))" :_Suc)	
text {* \medskip	Concrete Mathematics, page 280 *3	
lemma fib_add: "f apply (induct n	ib (Suc (n + k)) = fib (Suc k) * fib (Suc n) + fib k * fib rule: fib.induct)	> n"
txt f* simpli-	y the LHS just enough to apply the induction hypotheses *	3
apply (simp a	id: f1b_Suc3) 1 add: f1b.Suc_Suc add_mult_distrib2)	
done done	1 addit 110.366_366 add_861C015C1023	
apply (induct n	0: "fib (Sucin) # 0" rule: fib.induct) 1 add: fib.Suc_Suc)	
lemma_fib_Suc_gr_i by (insert fib_)	k: "0 < fib (Suc n)" suc_meq_0 [of n], simp)	
XEwacs: F1b.		criptin
proof (prove): st	ip 1	-
fixed variables: i	1, K	
cool (leves (Eth.)	udd). 3 subocals):	
1. f1b (Suc (0 +	k)) = fib (Suc k) * fib (Suc 0) + fib k * fib 0	
2. fib (Suc (Suc	0 + k)) = f1b (Suc (Suc 0)) + f1b k * f1b (Suc 0)	
<ol><li>Ax. If the (Suc.</li></ol>	(Suc × + k)) =	
f1b (Suc	k) * fib (Suc (Suc x)) + fib k * fib (Suc x); (x + k)) = fib (Suc k) * fib (Suc x) + fib k * fib x]	
→ fib (S)	ac (Suc (Suc X) + k)) =	
f1b (Si	ic k) * f1b (Suc (Suc (Suc x))) + f1b k * f1b (Suc (Suc x)	0

Motivation System Overview Higher-Order Logic

## Motivation

HOL4 and Isabelle/HOL are popular interactive theorem provers.

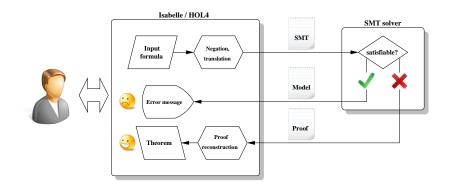
Interactive theorem proving benefits from automation.

	📉 Isabelle Proof General: Fib.thy	
ile Edit View Ced	ds <u>I</u> mols <u>Q</u> ations <u>E</u> uffers <u>Proof-Ceneral X</u> -Symbol <u>I</u> sabelle	He1
2 🖸 🧾 🖉		
F1b.thy *1sabe		
	le @[text fib.Suc_Suc3 for simplification*3] c_Suc [simp del]]	
ext[*then p emma fib_Suc3: by (rule fib.	prove a version that has a more restrictive pattern."] : "fib (Suc (Suc (Suc n))) = fib (Suc n) + fib (Suc (Suc .Suc_Suc))	n>)"
ext {* \medsk:	1p Concrete Mathematics, page 280 *3	
emma fib_add:	"fib (Suc (n + k)) = fib (Suc k) * fib (Suc n) + fib k * t n rule: fib.induct)	fib n"
prefer 3		
txt {* simple apply (simple apply (simple apply ) apply ) apply (simple apply ) apply (simple apply ) apply ) apply (simple apply ) apply ) apply (simple apply ) apply ) apply ) apply (simple apply ) apply ) apply ) apply (simple apply ) apply	plify the LHS just enough to apply the induction hypothes p add: fib_Suc3)	ves *3
apply (simp done	p_all add: fib.Suc_Suc add_mult_distrib2)	
apply (induct	neq_0: "fib (Suc n) # 0" t n rule: fib (induct) p_all add: fib.Suc_Suc)	
essa fib_Suc_c by (insert fi	gr_0: "0 < fib (Suc n)" ib_Suc_neq_0 [of n], simp)	
roof (prove):		Scripti
ixed variables	s:n,k	
oal (lemma (fi 1. fib (Suc (0 2. fib (Suc (5	ib_add), 3 subgoals): 0 + k)) = fib (Suc k) * fib (Suc 0) + fib k * fib 0	
fib (Suc k)	) * fib (Suc (Suc 0)) + fib k * fib (Suc 0)	
fib (s	Suc (Suc x + k)) = Suc (k) * ftb (Suc (Suc x)) + ftb k * ftb (Suc x); Suc (x + k)) = ftb (Suc k) * ftb (Suc x) + ftb k * ftb x]	
	) (Suc (Suc (Suc x) + k)) =	
f1b	(Suc k) * f1b (Suc (Suc (Suc x))) + f1b k * f1b (Suc (Su	ю x))

We want to use SMT solvers to decide SMT formulas.

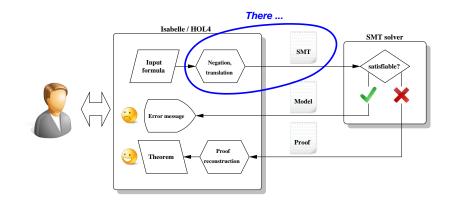
Motivation System Overview Higher-Order Logic Satisfiability Modulo Theories

### System Overview



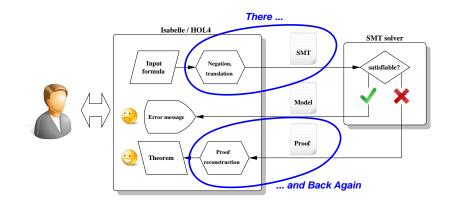
Motivation System Overview Higher-Order Logic Satisfiability Modulo Theories

### System Overview



Motivation System Overview Higher-Order Logic Satisfiability Modulo Theories

### System Overview



Tjark Weber Integration of SMT Solvers with Interactive Theorem Provers

Motivation System Overview **Higher-Order Logic** Satisfiability Modulo Theories

## Higher-Order Logic

Polymorphic  $\lambda$ -calculus, based on Church's simple theory of types:

- $\sigma ::= \alpha \mid (\sigma_1, \ldots, \sigma_n)c$
- $t ::= x_{\sigma} \mid c_{\sigma} \mid (t_{\sigma \to \tau} t_{\sigma})_{\tau} \mid (\lambda x_{\sigma} \cdot t_{\tau})_{\sigma \to \tau}$

Motivation System Overview **Higher-Order Logic** Satisfiability Modulo Theories

## Higher-Order Logic

Polymorphic  $\lambda$ -calculus, based on Church's simple theory of types:

•  $\sigma ::= \alpha \mid (\sigma_1, \dots, \sigma_n)c$ •  $t ::= x_\sigma \mid c_\sigma \mid (t_{\sigma \to \tau} t_\sigma)_\tau \mid (\lambda x_\sigma, t_\tau)_{\sigma \to \tau}$ 

Sufficient for much of mathematics and computer science:

- quantifiers of arbitrary order
- arithmetic (nat, int, real, ...)
- data types (lists, records, bit vectors, ...)



Extensive libraries with thousands of theorems

Motivation System Overview Higher-Order Logic Satisfiability Modulo Theories

### Satisfiability Modulo Theories

Goal: To decide the satisfiability of (quantifier-free) first-order formulas with respect to combinations of (decidable) background theories.

$$\varphi ::= \mathcal{A} \mid \neg \varphi \mid \varphi \lor \varphi \mid \varphi \land \varphi$$

Motivation System Overview Higher-Order Logic Satisfiability Modulo Theories

### Satisfiability Modulo Theories: Example

#### Theories:

- $\mathcal{I}$ : theory of integers  $\Sigma_{\mathcal{I}} = \{\leq, +, -, 0, 1\}$
- $\mathcal{L}$ : theory of lists  $\Sigma_{\mathcal{L}} = \{=, \text{ hd, tl, nil, cons}\}$
- $\mathcal{E}$ : theory of equality
  - $\Sigma:$  free function and predicate symbols

Problem: Is

 $x \leq y \land y \leq x + hd (cons 0 nil) \land P (f x - f y) \land \neg P 0$ satisfiable in  $\mathcal{I} \cup \mathcal{L} \cup \mathcal{E}$ ?

Features: SMT-LIB vs. Yices Translation Techniques Caveats

#### There . . .

We must translate HOL formulas into the input language of SMT solvers.

- SMT-LIB format
- 2 Yices's native format



Features: SMT-LIB vs. Yices Translation Techniques Caveats

### Features: SMT-LIB vs. Yices

	SMT-LIB	Yices		SMT-LIB	Yices
int, real	<ul> <li>✓</li> </ul>	<ul> <li>✓</li> </ul>	let	( < )	<ul> <li>✓</li> </ul>
nat, bool, $ ightarrow$	×	<ul> <li>✓</li> </ul>	$\lambda$ -terms	×	<ul> <li>✓</li> </ul>
prop. logic	✓	<ul> <li>✓</li> </ul>	tuples	×	✓
equality	✓	<ul> <li>✓</li> </ul>	records	×	✓
FOL	✓	<ul> <li>✓</li> </ul>	data types	×	✓
HOL	×	<ul> <li>✓</li> </ul>	bit vectors	✓	✓
arithmetic	<ul> <li>✓</li> </ul>	<ul> <li>✓</li> </ul>		I	I

Features: SMT-LIB vs. Yices Translation Techniques Caveats

### Recursion & Abstraction

We translate HOL formulas by recursion over their term structure:

$$\llbracket P_{\alpha \to \mathsf{bool}} \ x_{\alpha} \rrbracket = (\llbracket P_{\alpha \to \mathsf{bool}} \rrbracket \ \llbracket x_{\alpha} \rrbracket)$$

Abstraction is used to deal with unsupported terms/types.

Features: SMT-LIB vs. Yices Translation Techniques Caveats

## Recursion & Abstraction

We translate HOL formulas by recursion over their term structure:

$$\llbracket P_{\alpha \to \mathsf{bool}} \ x_{\alpha} \rrbracket = (\llbracket P_{\alpha \to \mathsf{bool}} \rrbracket \ \llbracket x_{\alpha} \rrbracket)$$

Abstraction is used to deal with unsupported terms/types.

SMT-LIB	Yices
:extrasorts (a)	(define-type a)
:extrafuns ((x a))	(define P::(-> a bool))
:extrapreds ((P a))	(define x::a)
:formula (not (P x))	(assert (not (P x)))

Features: SMT-LIB vs. Yices Translation Techniques Caveats

## **Basic Techniques**

A simple dictionary approach is sufficient for many HOL constants (e.g., propositional logic, arithmetic, bit vectors).



Features: SMT-LIB vs. Yices Translation Techniques Caveats

## **Basic Techniques**

A simple dictionary approach is sufficient for many HOL constants (e.g., propositional logic, arithmetic, bit vectors).

We try to replace HOL constants without SMT counterparts: terms are  $\beta$ -normalized, some constants (e.g.,  $\in$ ) are unfolded.





Features: SMT-LIB vs. Yices Translation Techniques Caveats

# **Basic Techniques**

A simple dictionary approach is sufficient for many HOL constants (e.g., propositional logic, arithmetic, bit vectors).

We try to replace HOL constants without SMT counterparts: terms are  $\beta$ -normalized, some constants (e.g.,  $\in$ ) are unfolded.

We add (universally quantified) definitions for certain other HOL constants (e.g., min, max).





Features: SMT-LIB vs. Yices Translation Techniques Caveats

# **Basic Techniques**

A simple dictionary approach is sufficient for many HOL constants (e.g., propositional logic, arithmetic, bit vectors).

We try to replace HOL constants without SMT counterparts: terms are  $\beta$ -normalized, some constants (e.g.,  $\in$ ) are unfolded.

We add (universally quantified) definitions for certain other HOL constants (e.g., min, max).

Some terms require special code (e.g., numerals, quantifiers).







\_

Features: SMT-LIB vs. Yices Translation Techniques Caveats

### Monomorphisation

In HOL, types can depend on type parameters. Since Yices only supports monomorphic types, we may need to create multiple copies of a polymorphic data type.



Features: SMT-LIB vs. Yices Translation Techniques Caveats

### Monomorphisation

In HOL, types can depend on type parameters. Since Yices only supports monomorphic types, we may need to create multiple copies of a polymorphic data type.

Example: datatype  $\alpha$  list = NIL | CONS  $\alpha \alpha$  list

```
(define-type a)
(define-type a-list (datatype
  a-NIL (a-CONS a-hd::a a-tl::a-list)))
```

```
(define-type b)
(define-type b-list (datatype
  b-NIL (b-CONS b-hd::b b-tl::b-list)))
```

2	
	2

Features: SMT-LIB vs. Yices Translation Techniques Caveats

### Caveats

Uniformly generating fresh identifiers is easier than re-using HOL identifiers.



Features: SMT-LIB vs. Yices Translation Techniques Caveats

## Caveats

Uniformly generating fresh identifiers is easier than re-using HOL identifiers.



**[**•]

There are subtle semantic differences between certain HOL and (allegedly corresponding) SMT-LIB/Yices functions.

Tjark Weber Integration of SMT Solvers with Interactive Theorem Provers

Features: SMT-LIB vs. Yices Translation Techniques Caveats

## Caveats

Uniformly generating fresh identifiers is easier than re-using HOL identifiers.

There are subtle semantic differences between certain HOL and (allegedly corresponding) SMT-LIB/Yices functions.

Yices "does no checking and can behave unpredictably if given bad input." The burden to produce correct input for the SMT solver is on our translation.



•



Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques Performance

### ... and Back Again

#### What if there is a bug in the translation ... or in the SMT solver?

Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques Performance

## ... and Back Again

What if there is a bug in the translation ... or in the SMT solver?

We require the SMT solver to produce a proof of unsatisfiability.

The proof is then checked (automatically) in the interactive prover.



Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques Performance

## Z3's Proofs

Z3 is a leading SMT solver. It generates natural deduction proofs.

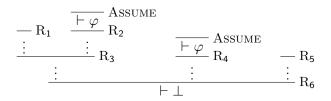
Z3's proof calculus consists of 34 axiom schemata and inference rules—some simple, some very powerful.

Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques Performance

## Z3's Proofs

Z3 is a leading SMT solver. It generates natural deduction proofs.

Z3's proof calculus consists of 34 axiom schemata and inference rules—some simple, some very powerful.



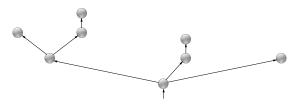
Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques Performance

## Z3's Proofs

Z3 is a leading SMT solver. It generates natural deduction proofs.

Z3's proof calculus consists of 34 axiom schemata and inference rules—some simple, some very powerful.

Proofs are directed acyclic graphs. Nodes are inference steps.



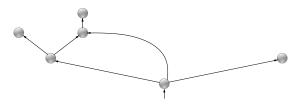
Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques Performance

## Z3's Proofs

Z3 is a leading SMT solver. It generates natural deduction proofs.

Z3's proof calculus consists of 34 axiom schemata and inference rules—some simple, some very powerful.

Proofs are directed acyclic graphs. Nodes are inference steps.



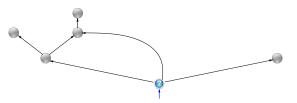
Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques Performance

## Z3's Proofs

Z3 is a leading SMT solver. It generates natural deduction proofs.

Z3's proof calculus consists of 34 axiom schemata and inference rules—some simple, some very powerful.

Proofs are directed acyclic graphs. Nodes are inference steps.



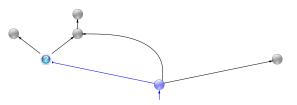
Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques Performance

## Z3's Proofs

Z3 is a leading SMT solver. It generates natural deduction proofs.

Z3's proof calculus consists of 34 axiom schemata and inference rules—some simple, some very powerful.

Proofs are directed acyclic graphs. Nodes are inference steps.



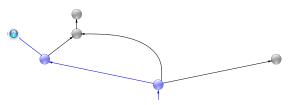
Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques Performance

## Z3's Proofs

Z3 is a leading SMT solver. It generates natural deduction proofs.

Z3's proof calculus consists of 34 axiom schemata and inference rules—some simple, some very powerful.

Proofs are directed acyclic graphs. Nodes are inference steps.



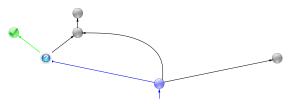
Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques Performance

## Z3's Proofs

Z3 is a leading SMT solver. It generates natural deduction proofs.

Z3's proof calculus consists of 34 axiom schemata and inference rules—some simple, some very powerful.

Proofs are directed acyclic graphs. Nodes are inference steps.



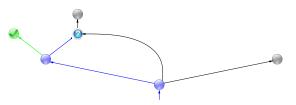
Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques Performance

## Z3's Proofs

Z3 is a leading SMT solver. It generates natural deduction proofs.

Z3's proof calculus consists of 34 axiom schemata and inference rules—some simple, some very powerful.

Proofs are directed acyclic graphs. Nodes are inference steps.



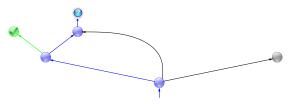
Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques Performance

## Z3's Proofs

Z3 is a leading SMT solver. It generates natural deduction proofs.

Z3's proof calculus consists of 34 axiom schemata and inference rules—some simple, some very powerful.

Proofs are directed acyclic graphs. Nodes are inference steps.



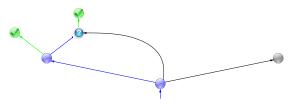
Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques Performance

# Z3's Proofs

Z3 is a leading SMT solver. It generates natural deduction proofs.

Z3's proof calculus consists of 34 axiom schemata and inference rules—some simple, some very powerful.

Proofs are directed acyclic graphs. Nodes are inference steps.



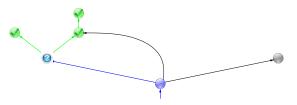
Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques Performance

# Z3's Proofs

Z3 is a leading SMT solver. It generates natural deduction proofs.

Z3's proof calculus consists of 34 axiom schemata and inference rules—some simple, some very powerful.

Proofs are directed acyclic graphs. Nodes are inference steps.



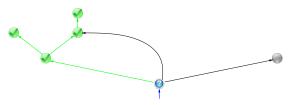
Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques Performance

# Z3's Proofs

Z3 is a leading SMT solver. It generates natural deduction proofs.

Z3's proof calculus consists of 34 axiom schemata and inference rules—some simple, some very powerful.

Proofs are directed acyclic graphs. Nodes are inference steps.



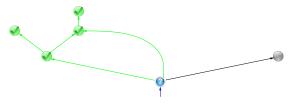
Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques Performance

# Z3's Proofs

Z3 is a leading SMT solver. It generates natural deduction proofs.

Z3's proof calculus consists of 34 axiom schemata and inference rules—some simple, some very powerful.

Proofs are directed acyclic graphs. Nodes are inference steps.



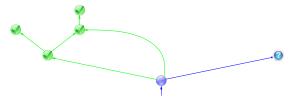
Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques Performance

# Z3's Proofs

Z3 is a leading SMT solver. It generates natural deduction proofs.

Z3's proof calculus consists of 34 axiom schemata and inference rules—some simple, some very powerful.

Proofs are directed acyclic graphs. Nodes are inference steps.



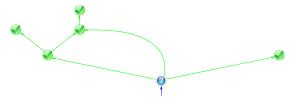
Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques Performance

# Z3's Proofs

Z3 is a leading SMT solver. It generates natural deduction proofs.

Z3's proof calculus consists of 34 axiom schemata and inference rules—some simple, some very powerful.

Proofs are directed acyclic graphs. Nodes are inference steps.



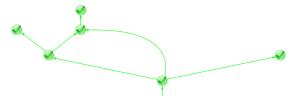
Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques Performance

# Z3's Proofs

Z3 is a leading SMT solver. It generates natural deduction proofs.

Z3's proof calculus consists of 34 axiom schemata and inference rules—some simple, some very powerful.

Proofs are directed acyclic graphs. Nodes are inference steps.



Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques Performance

# LCF-style Theorem Proving

Theorems are implemented as an abstract data type.

There is a fixed number of constructor functions—one for each axiom schema/inference rule of HOL.

More complicated proof procedures must be implemented by composing these functions.



Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques Performance

# LCF-style Theorem Proving

Theorems are implemented as an abstract data type.

There is a fixed number of constructor functions—one for each axiom schema/inference rule of HOL.

More complicated proof procedures must be implemented by composing these functions.



The trusted code base consists only of the theorem ADT.

Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques Performance

# LCF-style Theorem Proving — Disadvantages

• Proof procedures are more difficult to implement.



• Proof procedures are less efficient.



Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques Performance

## **Reconstruction** Techniques

- A single primitive inference rule or theorem instantiation
- ② Combinations of primitive inferences/instantiations
- Output Automated proof procedures
- Ombinations of the above

Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques Performance

### **Reconstruction Techniques**

- A single primitive inference rule or theorem instantiation
- ② Combinations of primitive inferences/instantiations
- Output Automated proof procedures
- Ombinations of the above

Implementation of Z3's inference rules:



Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques **Performance** 

# Performance Optimizations

#### Profiling is essential!

- Avoiding automated proof procedures
- Schematic theorems
- Theorem memoization
- Generalization



#### Speed-ups of up to 3 orders of magnitude

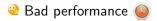
Introduction Z3's Pro There ... LCF-sty .. and Back Again Conclusions Perform

Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques **Performance** 

# Avoiding Automated Proof Procedures

About two thirds of Z3's proof rules perform propositional or simple first-order reasoning. They *could be* implemented by a single call to an automated proof procedure.

🥹 Rapid prototyping 💣

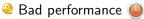


Performance

# Avoiding Automated Proof Procedures

About two thirds of Z3's proof rules perform propositional or simple first-order reasoning. They could be implemented by a single call to an automated proof procedure.

😌 Rapid prototyping 💣





Instead, we use derived rules: combinations of primitive inferences of manageable size that perform specific reasoning tasks.

Example:

$$\frac{\vdash \bigwedge_{i=1}^{n} \varphi_i}{\vdash \bigwedge_{i=1}^{n} \varphi_{\pi(i)}} \text{Rewrite}$$

Introduction Z3's Proofs There . . . . and Back Again Conclusions Performance

# Schematic Theorems

Instantiating a generic theorem is typically much faster than proving the specific instance using primitive inferences alone.

Examples:

• 
$$\vdash (p \implies q) \iff (\neg p \lor q)$$

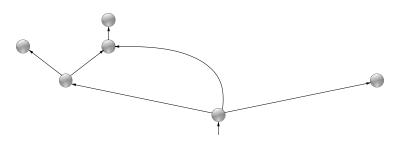
$$\bullet \vdash (x = y) \iff (y = x)$$

$$\bullet \vdash x + 0 = x$$

Over 230 theorems allow about 76% of all REWRITE goals to be proved by instantiation.

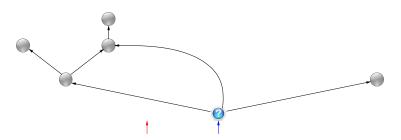
Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques **Performance** 

### **Theorem Memoization**



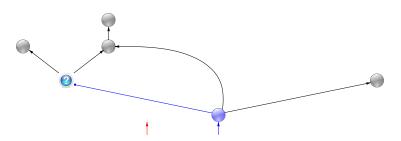
Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques **Performance** 

### **Theorem Memoization**



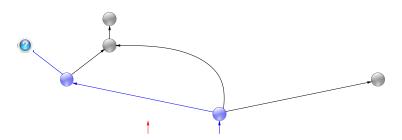
Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques **Performance** 

### **Theorem Memoization**



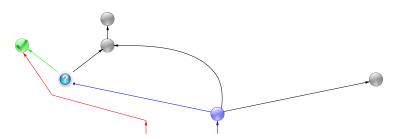
Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques **Performance** 

### **Theorem Memoization**



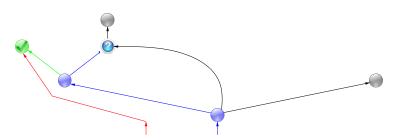
Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques **Performance** 

### Theorem Memoization



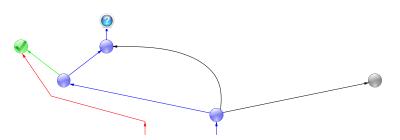
Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques **Performance** 

### **Theorem Memoization**



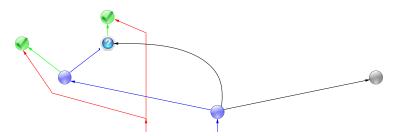
Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques **Performance** 

### Theorem Memoization



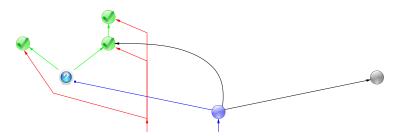
Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques **Performance** 

### **Theorem Memoization**



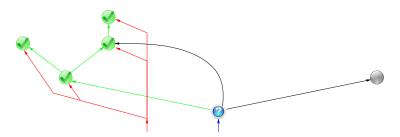
Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques **Performance** 

### **Theorem Memoization**



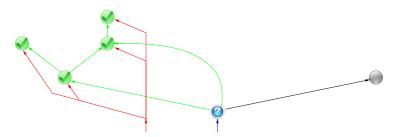
Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques **Performance** 

### **Theorem Memoization**



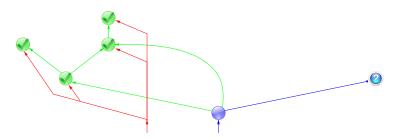
Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques **Performance** 

### Theorem Memoization



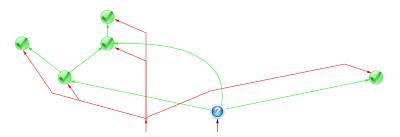
Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques **Performance** 

### Theorem Memoization



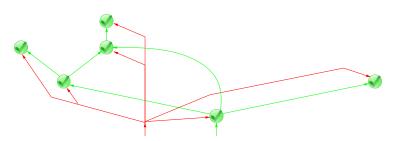
Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques **Performance** 

### **Theorem Memoization**



Z3's Proofs LCF-style Theorem Proving Reconstruction Techniques **Performance** 

### **Theorem Memoization**



Introduction Z3's Proofs There ... LCF-style Theorem Pro and Back Again Conclusions Performance

# Generalization

Goals proved by  $\rm Th{-}LEMMA$  are generalized before being passed to a theory-specific decision procedure.

#### Example:

 $\vdash$  some lengthy expression < some lengthy expression + 1 is a theorem of linear arithmetic—instead we prove  $\vdash x < x + 1$ .

🥹 Avoids expensive preprocessing in the decision procedure

More potential for theorem re-use

Introduction Z3's Proofs There ... LCF-style Theorem Pr and Back Again Reconstruction Techni Conclusions Performance

# Evaluation

Logic	Solved (Z3)			Reconstructed		Ratios		
	#	Time	Size	#	Time	Success	Timeout	Time
AUFLIA+p	187	0.095 s	64 KB	187	0.413 s	100%	0%	4.34
AUFLIA-p	192	0.117 s	81 K B	190	1.962 s	98%	0%	16.72
AUFLIRA	189	0.292 s	366 KB	144	0.794 s	76%	0%	2.72
QF_AUFLIA	92	0.158 s	694 KB	49	136.498 s	53%	42%	863.85
QF_IDL	40	2.322 s	12 MB	19	173.875 s	47%	52%	74.89
QF_LIA	100	17.154 s	77 MB	26	208.713 s	26%	65%	12.17
QF_LRA	88	4.849 s	10 MB	55	142.351 s	62%	36%	29.36
QF_RDL	52	9.773 s	16 MB	26	173.953 s	50%	50%	17.80
QF_UF	87	16.131 s	62 MB	73	73.242 s	83%	16%	4.54
QF_UFIDL	55	4.511 s	12 MB	8	260.351 s	14%	85%	57.72
QF_UFLIA	91	1.543 s	4 MB	85	29.086 s	93%	6%	18.85
QF_UFLRA	100	0.086 s	914 KB	100	3.916 s	100%	0%	45.68
Total	1273	3.656 s	13 MB	962	67.785 s	75%	19%	18.54

Introduction Z3's Proofs There ... LCF-style Theorem Proving ... and Back Again Reconstruction Techniques Conclusions Performance

# **Evaluation**

Logic		Solved (Z	3)	Reco	nstructed	Ratios		
	#	Time	Size	#	Time	Success	Timeout	Time
Total	1273	3.656 s	13 MB	962	67.785 s	75%	19%	18.54

We can check sizeable proofs with millions of inferences.

- Proof search in Z3 is almost 20 times faster (on average) than LCF-style proof reconstruction.
  - Not enough proof information for theory-specific reasoning.

Conclusions Future Work

# Conclusions

Integration of SMT solvers with HOL4 and Isabelle/HOL

- SMT-LIB is restrictive—custom translations seem more worthwhile than sophisticated SMT-LIB encodings.
- Z3's proofs could be easier to check.
- UCF-style proof checking for SMT is feasible.
- Isabelle: Isabelle.in.tum.de/ HOL4: HOL

Related papers at Shttp://www.cl.cam.ac.uk/~tw333/

Conclusions Future Work

# Future Work

- A more expressive SMT-LIB format (Version 2.0?!)
- A better SMT proof format (a standard?!)
- Proof reconstruction for bit vectors
- Case studies, applications



Conclusions Future Work

# Future Work

- A more expressive SMT-LIB format (Version 2.0?!)
- A better SMT proof format (a standard?!)
- Proof reconstruction for bit vectors
- Case studies, applications



