Integrating SAT and SMT Solvers with Interactive Theorem Provers

Tjark Weber

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

TypiCal Seminar
Laboratoire d'Informatique
École Polytechnique

10 September 2009

Motivation

1. Interactive theorem proving needs automation.
    - Use SAT solvers to decide formulas of propositional logic.
    - Use SMT solvers to decide SMT formulas.
    - Can we do this without increasing the trusted code base?

Motivation

1. Interactive theorem proving needs automation.
    - Use SAT solvers to decide formulas of propositional logic.
    - Use SMT solvers to decide SMT formulas.
    - Can we do this without increasing the trusted code base?
2. SAT and SMT solvers frequently contain bugs. How can we verify their results?
    - Proofs (of unsatisfiability) can be checked independently.
    - Can we keep the proof checker small?

LCF-Style Proof Checking

Theorems are implemented as an abstract data type. They can be constructed only through a fixed set of functions provided by this data type.

Each constructor function implements an axiom or inference rule of the logic.

Advanced proof procedures must (ultimately) employ combinations of primitive inferences.

An LCF-Style Integration of Proof-Producing SAT Solvers

Propositional Logic

Propositional logic:

- Boolean variables: p, q, ...
- Formulas: $\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi$

Propositional Logic

Propositional logic:

- Boolean variables: p, q, ...
- Formulas: $\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi$

Conjunctive normal form (CNF): a conjunction of clauses, where each clause is a disjunction of literals (i.e., possibly negated variables)

Propositional Logic

Propositional logic:

- Boolean variables: p, q, ...
- Formulas: $\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi$

Conjunctive normal form (CNF): a conjunction of clauses, where each clause is a disjunction of literals (i.e., possibly negated variables)

Abstraction from higher-order to propositional logic: replace subterms by Boolean variables, e.g.,

$$(\forall x.\, P\, x) \vee \neg(\forall x.\, P\, x) \quad \mapsto \quad p \vee \neg p$$

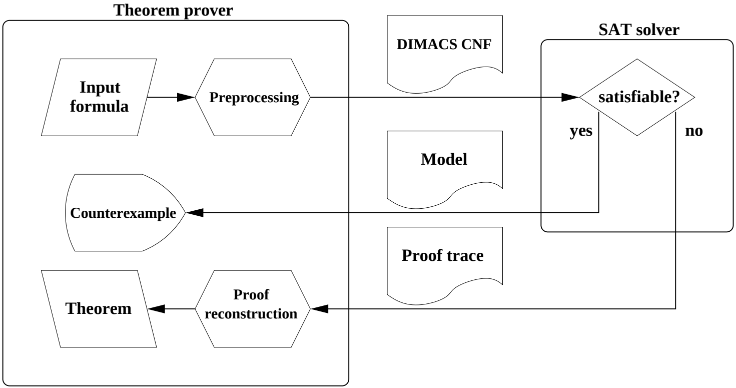
Propositional Resolution

$$\frac{P \cup \{x\} \qquad Q \cup \{\neg x\}}{P \cup Q}$$

Theorem

Propositional resolution is sound and refutation complete.

System Overview



DIMACS CNF Format

DIMACS CNF is the standard input format for SAT solvers.

Example: $(\neg x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee x_3)$

DIMACS CNF File

```
c This is just a comment line.
p cnf 3 4
-1 2 0
-2 -3 0
1 2 0
-2 3 0
```

zChaff Proof Format

No standard proof format for SAT solvers exists.

Example:

$$\dfrac{\dfrac{(3)\ \neg x_2 \vee x_3 \qquad \dfrac{(2)\ x_1 \vee x_2 \qquad (0)\ \neg x_1 \vee x_2}{(4)\ x_2}}{x_3} \qquad \dfrac{(1)\ \neg x_2 \vee \neg x_3 \qquad \dfrac{x_1 \vee x_2 \qquad \neg x_1 \vee x_2}{x_2}}{\neg x_3}}{\bot}$$

zChaff Proof File

```
CL: 4 <= 2 0
VAR: 2 L: 0 V: 1 A: 4 Lits:  4
VAR: 3 L: 1 V: 0 A: 1 Lits:  5 7
CONF: 3 == 5 6
```

Proof Reconstruction: Basics

The proof is a DAG. Each node represents an inference step and is connected to its premises.

Nodes contain information parsed from the proof file (initially), or the derived theorem (after reconstruction).

A designated root node derives False.

Depth-first (postorder) traversal determines the order of proof reconstruction.

Representation of SAT Problems

Bad: use logical connectives $\wedge$, $\vee$

Good: use sets of clauses and literals

Representation of SAT Problems

Bad: use logical connectives $\wedge$, $\vee$

Good: use sets of clauses and literals

1. The whole CNF problem is assumed: $\{\bigwedge_{i=1}^{k} C_i\} \vdash \bigwedge_{i=1}^{k} C_i$.
2. Each clause is derived: $\{\bigwedge_{i=1}^{k} C_i\} \vdash C_1, \ldots, \{\bigwedge_{i=1}^{k} C_i\} \vdash C_k$.
3. Then a sequent representation is used:

$$\{\bigwedge_{i=1}^{k} C_i, \overline{p_1}, \ldots, \overline{p_n}\} \vdash \text{False}.$$

Representation of SAT Problems

Bad: use logical connectives $\wedge$, $\vee$

Good: use sets of clauses and literals

1. The whole CNF problem is assumed: $\{\bigwedge_{i=1}^{k} C_i\} \vdash \bigwedge_{i=1}^{k} C_i$.
2. Each clause is derived: $\{\bigwedge_{i=1}^{k} C_i\} \vdash C_1, \ldots, \{\bigwedge_{i=1}^{k} C_i\} \vdash C_k$.
3. Then a sequent representation is used:

$$\{\bigwedge_{i=1}^{k} C_i, \overline{p_1}, \ldots, \overline{p_n}\} \vdash \text{False}.$$

The problem is an array of clauses. Clauses are sets of literals.

Propositional Resolution, LCF-Style

With the sequent representation, resolution is fast:

$\{\bigwedge_{i=1}^{k} C_i, \overline{p_1}, \ldots, \overline{p_n}\} \vdash \mathrm{False}$, $\{\bigwedge_{i=1}^{k} C_i, \overline{q_1}, \ldots, \overline{q_m}\} \vdash \mathrm{False}$

Propositional Resolution, LCF-Style

With the sequent representation, resolution is fast:

$\{\bigwedge_{i=1}^{k} C_i, \overline{p_1}, \ldots, \overline{p_n}\} \vdash \mathsf{False}$, $\{\bigwedge_{i=1}^{k} C_i, \overline{q_1}, \ldots, \overline{q_m}\} \vdash \mathsf{False}$

1. $\textsc{impI}$: $\{\bigwedge_{i=1}^{k} C_i, \overline{p_1}, \ldots, \overline{p_n}\} \setminus \{x\} \vdash x \Rightarrow \mathsf{False}$

Propositional Resolution, LCF-Style

With the sequent representation, resolution is fast:

$\{\bigwedge_{i=1}^{k} C_i, \overline{p_1}, \ldots, \overline{p_n}\} \vdash \text{False}$, $\{\bigwedge_{i=1}^{k} C_i, \overline{q_1}, \ldots, \overline{q_m}\} \vdash \text{False}$

1. IMPI: $\{\bigwedge_{i=1}^{k} C_i, \overline{p_1}, \ldots, \overline{p_n}\} \setminus \{x\} \vdash x \Rightarrow \text{False}$
2. IMPI: $\{\bigwedge_{i=1}^{k} C_i, \overline{q_1}, \ldots, \overline{q_m}\} \setminus \{\neg x\} \vdash \neg x \Rightarrow \text{False}$

Propositional Resolution, LCF-Style

With the sequent representation, resolution is fast:

$\{\bigwedge_{i=1}^{k} C_i, \overline{p_1}, \ldots, \overline{p_n}\} \vdash \mathsf{False}$, $\{\bigwedge_{i=1}^{k} C_i, \overline{q_1}, \ldots, \overline{q_m}\} \vdash \mathsf{False}$

1. IMPI: $\{\bigwedge_{i=1}^{k} C_i, \overline{p_1}, \ldots, \overline{p_n}\} \setminus \{x\} \vdash x \Rightarrow \mathsf{False}$
2. IMPI: $\{\bigwedge_{i=1}^{k} C_i, \overline{q_1}, \ldots, \overline{q_m}\} \setminus \{\neg x\} \vdash \neg x \Rightarrow \mathsf{False}$
3. INST: $\vdash (x \Rightarrow \mathsf{False}) \Rightarrow (\neg x \Rightarrow \mathsf{False}) \Rightarrow \mathsf{False}$

Propositional Resolution, LCF-Style

With the sequent representation, resolution is fast:

$\{\bigwedge_{i=1}^{k} C_i, \overline{p_1}, \ldots, \overline{p_n}\} \vdash \mathsf{False}$, $\{\bigwedge_{i=1}^{k} C_i, \overline{q_1}, \ldots, \overline{q_m}\} \vdash \mathsf{False}$

1. IMPI: $\{\bigwedge_{i=1}^{k} C_i, \overline{p_1}, \ldots, \overline{p_n}\} \setminus \{x\} \vdash x \Rightarrow \mathsf{False}$
2. IMPI: $\{\bigwedge_{i=1}^{k} C_i, \overline{q_1}, \ldots, \overline{q_m}\} \setminus \{\neg x\} \vdash \neg x \Rightarrow \mathsf{False}$
3. INST: $\vdash (x \Rightarrow \mathsf{False}) \Rightarrow (\neg x \Rightarrow \mathsf{False}) \Rightarrow \mathsf{False}$
4. MP: $\{\bigwedge_{i=1}^{k} C_i, \overline{p_1}, \ldots, \overline{p_n}\} \setminus \{x\} \vdash (\neg x \Rightarrow \mathsf{False}) \Rightarrow \mathsf{False}$

Propositional Resolution, LCF-Style

With the sequent representation, resolution is fast:

$\{\bigwedge_{i=1}^{k} C_i, \overline{p_1}, \ldots, \overline{p_n}\} \vdash \text{False}$, $\{\bigwedge_{i=1}^{k} C_i, \overline{q_1}, \ldots, \overline{q_m}\} \vdash \text{False}$

1. IMPI: $\{\bigwedge_{i=1}^{k} C_i, \overline{p_1}, \ldots, \overline{p_n}\} \setminus \{x\} \vdash x \Rightarrow \text{False}$
2. IMPI: $\{\bigwedge_{i=1}^{k} C_i, \overline{q_1}, \ldots, \overline{q_m}\} \setminus \{\neg x\} \vdash \neg x \Rightarrow \text{False}$
3. INST: $\vdash (x \Rightarrow \text{False}) \Rightarrow (\neg x \Rightarrow \text{False}) \Rightarrow \text{False}$
4. MP: $\{\bigwedge_{i=1}^{k} C_i, \overline{p_1}, \ldots, \overline{p_n}\} \setminus \{x\} \vdash (\neg x \Rightarrow \text{False}) \Rightarrow \text{False}$
5. MP: $\{\bigwedge_{i=1}^{k} C_i, \overline{p_1}, \ldots, \overline{p_n}, \overline{q_1}, \ldots, \overline{q_m}\} \setminus \{x, \neg x\} \vdash \text{False}$

Performance

Evaluation on SATLIB problems:

Problem	Variables	Clauses	Resolutions	zChaff (s)	Isabelle (s)
c7552mul.miter	11282	69529	242509	45	24
6pipe	15800	394739	310813	137	55
6pipe_6_ooo	17064	545612	782903	265	156
7pipe	23910	751118	497019	440	169

Evaluation on pigeonhole instances:

Problem	Variables	Clauses	Resolutions	zChaff (s)	Isabelle (s)
pigeon-9	90	415	73472	1	2
pigeon-10	110	561	215718	6	4
pigeon-11	132	738	601745	24	12
pigeon-12	156	949	3186775	247	68

An LCF-Style Integration of Proof-Producing SMT Solvers

Satisfiability Modulo Theories

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

$$\varphi ::= \mathcal{A} \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi$$

Applications:

- Formal verification
- Scheduling
- Compiler optimization
- ...

Example

Theories:

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

Problem: Is

$$x \leq y \wedge y \leq x + \mathsf{hd}\,(\mathsf{cons}\,0\,\mathsf{nil}) \wedge P\,(f\,x - f\,y) \wedge \neg\, P\,0$$

satisfiable in $\mathcal{I} \cup \mathcal{L} \cup \mathcal{E}$?

Algorithms

SMT solvers typically use a combination of SAT solving and theory-specific decision procedures.

- DPLL: standard decision procedure for SAT (based on splitting and unit propagation)
- Nelson-Oppen: a decision procedure for the union of decidable theories (using variable abstraction and equality propagation)
- DPLL(T): tight integration of a theory-specific decision procedure with the DPLL algorithm

SMT-LIB Format

SMT-LIB is the standard input format for SMT solvers.

- LISP-like syntax
- Based on first-order logic
- Modular: different “theories” and “logics”
- Version 2.0 is due real soon now
- `http://goedel.cs.uiowa.edu/smtlib/`

Greatly helped to unify the field!

Translation from Higher-Order Logic

	SMT-LIB	Yices		SMT-LIB	Yices
int, real	✓	✓	let	(✓)	✓
nat, bool, $\rightarrow$		✓	λ-terms		✓
prop. logic	✓	✓	tuples		✓
equality	✓	✓	records		✓
FOL	✓	✓	data types		✓
HOL		✓	bit vectors	✓	✓
arithmetic	✓	✓			

Abstraction is used to deal with unsupported terms/types.

Z3's Proof Format

Term language: many-sorted first-order logic

- bool, int, real
- $+$, $-$, $\cdot$, $\vee$, $\wedge$, $\neg$, $\top$, $\bot$, $\forall$, $\exists$, distinct, select, store

Z3's Proof Format

Term language: many-sorted first-order logic

- bool, int, real
- $+$, $-$, $\cdot$, $\vee$, $\wedge$, $\neg$, $\top$, $\bot$, $\forall$, $\exists$, distinct, select, store

Proofs: natural deduction

- 34 axioms and inference rules, from simple (e.g., **mp**) to complex (e.g., **rewrite**, **th-lemma**)
- Contain: inference rule used, pointers to premises, conclusion

Z3 Proof File (Example)

```
⋮
#57 := (iff #15 #34)
#58 := [rewrite]:  #57
#61 := [monotonicity #58]:  #60
⋮
```

Proof Reconstruction: Basics

The proof is a DAG. Each node represents an inference step and is connected to its premises.

Nodes contain information parsed from the proof file (initially), or the derived theorem (after reconstruction).

A designated root node derives False.

Depth-first (postorder) traversal determines the order of proof reconstruction.

Proof Reconstruction: Basics

The proof is a DAG. Each node represents an inference step and is connected to its premises.

Nodes contain information parsed from the proof file (initially), or the derived theorem (after reconstruction).

A designated root node derives False.

Depth-first (postorder) traversal determines the order of proof reconstruction.

Same as for SAT!

Proof Reconstruction: Assumptions, Skolemization

Z3's proofs may contain local (cf. **hypothesis**) and global (cf. **asserted**) assumptions:

- ASSUME: $\{\varphi\} \vdash \varphi$
- At the very end, we check that all local assumptions have been discharged.

Proof Reconstruction: Assumptions, Skolemization

Z3's proofs may contain local (cf. **hypothesis**) and global (cf. **asserted**) assumptions:

- ASSUME: $\{\varphi\} \vdash \varphi$
- At the very end, we check that all local assumptions have been discharged.

Skolem functions (introduced by **sk**) are given hypothetical definitions in terms of Hilbert's choice operator.

Rapid Prototyping

About one third of Z3's proof rules perform propositional reasoning. $\longrightarrow$ TAUTPROVE

Rapid Prototyping

About one third of Z3's proof rules perform propositional reasoning. $\longrightarrow$ TAUTPROVE

About one third of Z3's proof rules perform relatively simple first-order reasoning. $\longrightarrow$ METIS

Rapid Prototyping

About one third of Z3's proof rules perform propositional reasoning. $\longrightarrow$ TAUTPROVE

About one third of Z3's proof rules perform relatively simple first-order reasoning. $\longrightarrow$ METIS

Slow!

Rapid Prototyping

About one third of Z3's proof rules perform propositional reasoning. $\longrightarrow$ TautProve

About one third of Z3's proof rules perform relatively simple first-order reasoning. $\longrightarrow$ Metis

Slow!

Speedups of several orders of magnitude can be achieved through specialized implementations that perform the required inferences directly.

Optimizations: Propositional and First-Order Reasoning I

Nested conjunctions: equivalence of $\bigwedge_{i=1}^{n} \varphi_i$ and $\bigwedge_{i=1}^{n} \varphi_{\pi(i)}$ can be established in $O(n \log n)$ using conjunction elimination and introduction only (no associativity/commutativity theorems!)

Optimizations: Propositional and First-Order Reasoning I

Nested conjunctions: equivalence of $\bigwedge_{i=1}^{n} \varphi_i$ and $\bigwedge_{i=1}^{n} \varphi_{\pi(i)}$ can be established in $O(n \log n)$ using conjunction elimination and introduction only (no associativity/commutativity theorems!)

1. ASSUME: $\bigwedge_{i=1}^{n} \varphi_i \vdash \bigwedge_{i=1}^{n} \varphi_i$

Optimizations: Propositional and First-Order Reasoning I

Nested conjunctions: equivalence of $\bigwedge_{i=1}^{n} \varphi_i$ and $\bigwedge_{i=1}^{n} \varphi_{\pi(i)}$ can be established in $O(n \log n)$ using conjunction elimination and introduction only (no associativity/commutativity theorems!)

1. ASSUME: $\bigwedge_{i=1}^{n} \varphi_i \vdash \bigwedge_{i=1}^{n} \varphi_i$
2. Repeated CONJE: $\bigwedge_{i=1}^{n} \varphi_i \vdash \varphi_1, \ldots, \bigwedge_{i=1}^{n} \varphi_i \vdash \varphi_n$

Optimizations: Propositional and First-Order Reasoning I

Nested conjunctions: equivalence of $\bigwedge_{i=1}^{n} \varphi_i$ and $\bigwedge_{i=1}^{n} \varphi_{\pi(i)}$ can be established in $O(n \log n)$ using conjunction elimination and introduction only (no associativity/commutativity theorems!)

1. ASSUME: $\bigwedge_{i=1}^{n} \varphi_i \vdash \bigwedge_{i=1}^{n} \varphi_i$
2. Repeated CONJE: $\bigwedge_{i=1}^{n} \varphi_i \vdash \varphi_1, \ldots, \bigwedge_{i=1}^{n} \varphi_i \vdash \varphi_n$
3. Store these theorems in a red-black tree, indexed by their conclusion.

Optimizations: Propositional and First-Order Reasoning I

Nested conjunctions: equivalence of $\bigwedge_{i=1}^{n} \varphi_i$ and $\bigwedge_{i=1}^{n} \varphi_{\pi(i)}$ can be established in $O(n \log n)$ using conjunction elimination and introduction only (no associativity/commutativity theorems!)

1. Assume: $\bigwedge_{i=1}^{n} \varphi_i \vdash \bigwedge_{i=1}^{n} \varphi_i$
2. Repeated conjE: $\bigwedge_{i=1}^{n} \varphi_i \vdash \varphi_1, \ldots, \bigwedge_{i=1}^{n} \varphi_i \vdash \varphi_n$
3. Store these theorems in a red-black tree, indexed by their conclusion.
4. Repeated conjI: $\bigwedge_{i=1}^{n} \varphi_i \vdash \bigwedge_{i=1}^{n} \varphi_{\pi(i)}$

Optimizations: Propositional and First-Order Reasoning II

Nested disjunctions: dual to nested conjunctions, but trickier

Optimizations: Propositional and First-Order Reasoning II

Nested disjunctions: dual to nested conjunctions, but trickier

Unit resolution: $\frac{\Gamma \vdash \bigvee_{i \in I} \varphi_i \quad \langle \Gamma_i \vdash \neg \varphi_i \rangle_{i \in J}}{\Gamma \cup \bigcup_{i \in J} \Gamma_i \vdash \bigvee_{i \in I \setminus J} \varphi_i}$ similar to nested disjunctions

Optimizations: Propositional and First-Order Reasoning II

Nested disjunctions: dual to nested conjunctions, but trickier

Unit resolution: $\frac{\Gamma \vdash \bigvee_{i \in I} \varphi_i \quad \langle \Gamma_i \vdash \neg \varphi_i \rangle_{i \in J}}{\Gamma \cup \bigcup_{i \in J} \Gamma_i \vdash \bigvee_{i \in I \setminus J} \varphi_i}$ similar to nested disjunctions

Quantifier instantiations: determined by first-order term matching

Optimizations: Theory-Specific Reasoning

Proforma theorems: more than 230 proforma theorems allow about 76% of all terms given to **rewrite** to be proved by instantiation

Optimizations: Theory-Specific Reasoning

Proforma theorems: more than 230 proforma theorems allow about 76% of all terms given to **rewrite** to be proved by instantiation

Theorem caching: theorems proved by **rewrite** and **th-lemma** are cached (indexed by a term net) for later re-use

Optimizations: Theory-Specific Reasoning

Proforma theorems: more than 230 proforma theorems allow about 76% of all terms given to **rewrite** to be proved by instantiation

Theorem caching: theorems proved by **rewrite** and **th-lemma** are cached (indexed by a term net) for later re-use

Generalization: terms passed to HOL4's arithmetic decision procedures are generalized first (for faster preprocessing)

Implementation Techniques (Overview)

Primitive inference, proforma theorem: **asserted**, **commutativity**, **hypothesis**, **iff-false**, **iff-true**, **mp**, **mp$\sim$**, **refl**, **symm**, **trans**

Combination of primitive inferences/instantiations: **and-elim**, **def-axiom**, **elim-unused**, **lemma**, **monotonicity**, **nnf-neg**, **nnf-pos**, **not-or-elim**, **pull-quant**, **quant-inst**, **quant-intro**, **sk**, **unit-resolution**

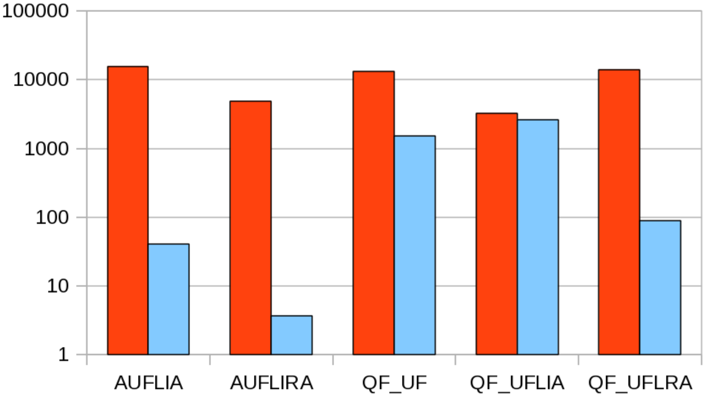
Automated proof procedure: —

Combination of the above: **rewrite**, **th-lemma**

Experimental Results

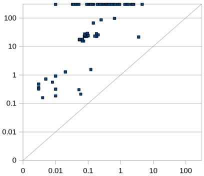
Logic	Solved (Z3)		Reconstructed		Failed		Factor
	#	Time	#	Time	#T	#Z	
AUFLIA	100	0.180 s	100	0.407 s	0	0	2.3
AUFLIRA	100	0.051 s	97	0.038 s	0	3	0.7
QF_UF	96	2.992 s	74	16.618 s	1	21	5.6
QF_UFLIA	99	0.534 s	92	5.889 s	7	0	11.0
QF_UFLRA	100	0.189 s	100	1.673 s	0	0	8.9

Total Run-Times

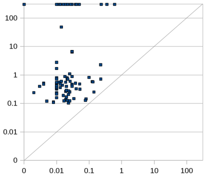


Isabelle/HOL (Böhme, SMT ’09) vs. HOL4 (average speedup: 12.4)

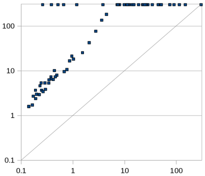
Run-Times: Individual Problems



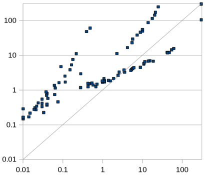
AUFLIA



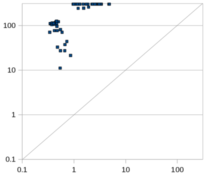
AUFLIRA



QF_UF

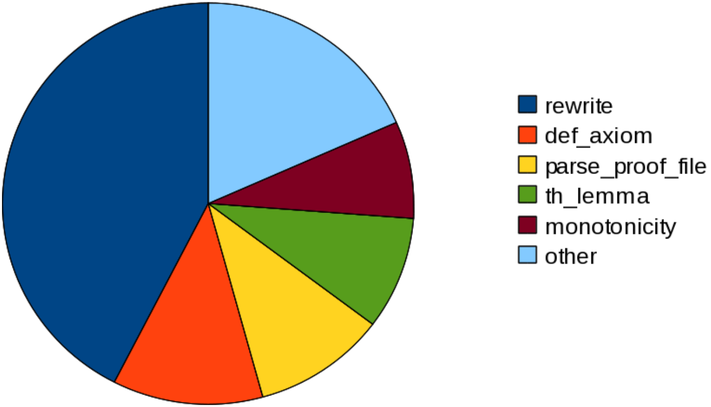


QF_UFLIA

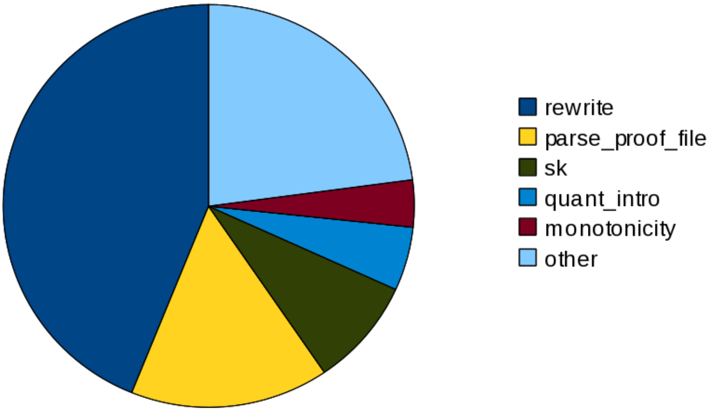


QF_UFLRA

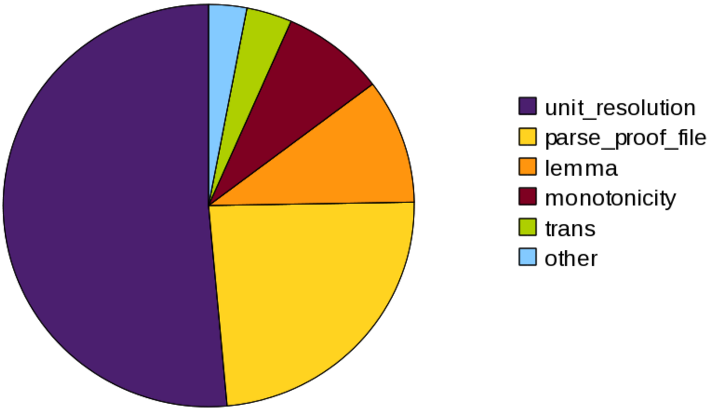
Profiling: AUFLIA



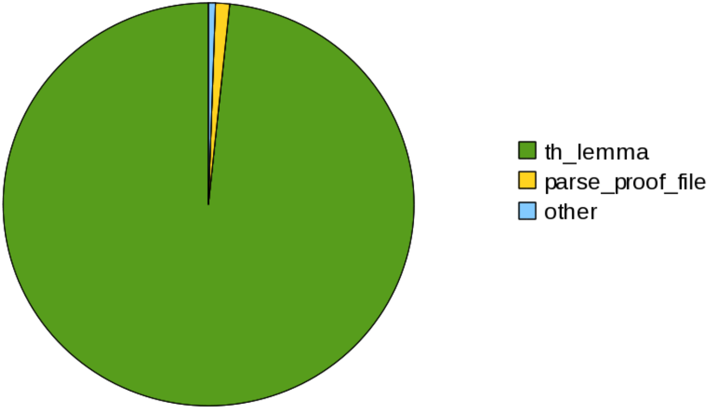
Profiling: AUFLIRA



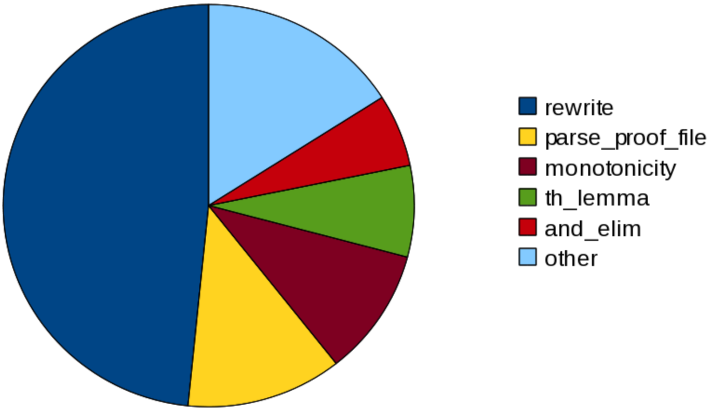
Profiling: QF_UF



Profiling: QF_UFLIA



Profiling: QF_UFLRA



Conclusions

- LCF-style proof checking for SAT and SMT is feasible.
- In LCF-style theorem provers, specialized implementations can be much faster than automated generic proof procedures.
- Z3's proof format is reasonably easy to check. Only **rewrite** and **th-lemma** are overly complex.

Future Work

- A standard proof format for SAT solvers
- A standard proof format for SMT solvers
- Proof reconstruction for bit vectors
- Parallel proof checking
- Proof compression

Questions?

Thank you for your attention.