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
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?
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?

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?
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: \( p, q, \ldots \)
- Formulas: \( \varphi ::= p \mid \neg \varphi \mid \varphi \lor \varphi \mid \varphi \land \varphi \)
Propositional Logic

Propositional logic:
- Boolean variables: \( p, q, \ldots \)
- Formulas: \( \varphi ::= p \mid \neg \varphi \mid \varphi \lor \varphi \mid \varphi \land \varphi \)

 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: \( p, q, \ldots \)
- Formulas:
  \[ \varphi ::= p \mid \neg \varphi \mid \varphi \lor \varphi \mid \varphi \land \varphi \]

**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.,

\[
(\forall x. P x) \lor \neg (\forall x. P x) \quad \mapsto \quad p \lor \neg p
\]
Propositional Resolution

\[ P \cup \{x\} \quad Q \cup \{\neg x\} \]

\[ \frac{P \cup \{x\}}{P \cup Q} \quad \frac{Q \cup \{\neg x\}}{P \cup Q} \]

**Theorem**

Propositional resolution is **sound** and **refutation complete**.
DIMACS CNF Format

DIMACS CNF is the **standard input format** for SAT solvers.

Example: \((\neg x_1 \lor x_2) \land (\neg x_2 \lor \neg x_3) \land (x_1 \lor x_2) \land (\neg x_2 \lor x_3)\)

**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
```
No standard proof format for SAT solvers exists.

Example:

\[
\begin{align*}
(3) & \quad \neg x_2 \vee x_3 \\
& \quad \frac{\frac{(2) \ x_1 \lor x_2 \quad (0) \ \neg x_1 \lor x_2}{\quad (4) \ x_2}}{\quad (1) \ \neg x_2 \lor \neg x_3} \\
& \quad \frac{x_3}{\bot}
\end{align*}
\]

**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
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 $\land$, $\lor$

Good: use sets of clauses and literals
Representation of SAT Problems

Bad: use logical connectives $\land$, $\lor$

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: $\{\bigwedge_{i=1}^{k} C_i\} \vdash C_1$, ..., $\{\bigwedge_{i=1}^{k} C_i\} \vdash C_k$.
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 $\land$, $\lor$

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:

\[ \{ \bigwedge_{i=1}^{k} C_i, \overline{p_1}, \ldots, \overline{p_n} \} \vdash \text{False}, \{ \bigwedge_{i=1}^{k} C_i, \overline{q_1}, \ldots, \overline{q_m} \} \vdash \text{False} \]
With the sequent representation, **resolution is fast**:

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

1. **IMP1**: \( \{ \bigwedge_{i=1}^{k} C_i, \overline{p_1}, \ldots, \overline{p_n} \} \setminus \{x\} \vdash x \Rightarrow \text{False} \)
Propositional Resolution, LCF-Style

With the sequent representation, resolution is fast:

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

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:

\[
\land_{i=1}^{k} C_i, \overline{p_1}, \ldots, \overline{p_n} \vdash \text{False}, \ \land_{i=1}^{k} C_i, \overline{q_1}, \ldots, \overline{q_m} \vdash \text{False}
\]

1. \text{IMPI: } \land_{i=1}^{k} C_i, \overline{p_1}, \ldots, \overline{p_n} \setminus \{x\} \vdash x \Rightarrow \text{False}
2. \text{IMPI: } \land_{i=1}^{k} C_i, \overline{q_1}, \ldots, \overline{q_m} \setminus \{\neg x\} \vdash \neg x \Rightarrow \text{False}
3. \text{INST: } \vdash (x \Rightarrow \text{False}) \Rightarrow (\neg x \Rightarrow \text{False}) \Rightarrow \text{False}
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: \( \{ \bigwedge_{i=1}^{k} C_i, \overline{p_1}, \ldots, \overline{p_n} \} \setminus \{ x \} \vdash (\neg x \Rightarrow \text{False}) \Rightarrow \text{False} \)
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. \text{IMPI}: \{ \bigwedge_{i=1}^{k} C_i, \overline{p}_1, \ldots, \overline{p}_n \} \setminus \{x\} \vdash x \Rightarrow \text{False}
2. \text{IMPI}: \{ \bigwedge_{i=1}^{k} C_i, \overline{q}_1, \ldots, \overline{q}_m \} \setminus \{\neg x\} \vdash \neg x \Rightarrow \text{False}
3. \text{INST}: \vdash (x \Rightarrow \text{False}) \Rightarrow (\neg x \Rightarrow \text{False}) \Rightarrow \text{False}
4. \text{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. \text{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:

<table>
<thead>
<tr>
<th>Problem</th>
<th>Variables</th>
<th>Clauses</th>
<th>Resolutions</th>
<th>zChaff (s)</th>
<th>Isabelle (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c7552mul.miter</td>
<td>11282</td>
<td>69529</td>
<td>242509</td>
<td>45</td>
<td>24</td>
</tr>
<tr>
<td>6pipe</td>
<td>15800</td>
<td>394739</td>
<td>310813</td>
<td>137</td>
<td>55</td>
</tr>
<tr>
<td>6pipe_6..ooo</td>
<td>17064</td>
<td>545612</td>
<td>782903</td>
<td>265</td>
<td>156</td>
</tr>
<tr>
<td>7pipe</td>
<td>23910</td>
<td>751118</td>
<td>497019</td>
<td>440</td>
<td>169</td>
</tr>
</tbody>
</table>

#### Evaluation on pigeonhole instances:

<table>
<thead>
<tr>
<th>Problem</th>
<th>Variables</th>
<th>Clauses</th>
<th>Resolutions</th>
<th>zChaff (s)</th>
<th>Isabelle (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>pigeon-9</td>
<td>90</td>
<td>415</td>
<td>73472</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>pigeon-10</td>
<td>110</td>
<td>561</td>
<td>215718</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>pigeon-11</td>
<td>132</td>
<td>738</td>
<td>601745</td>
<td>24</td>
<td>12</td>
</tr>
<tr>
<td>pigeon-12</td>
<td>156</td>
<td>949</td>
<td>3186775</td>
<td>247</td>
<td>68</td>
</tr>
</tbody>
</table>
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.

\[ \varphi ::= A \mid \neg \varphi \mid \varphi \lor \varphi \mid \varphi \land \varphi \]

Applications:
- Formal verification
- Scheduling
- Compiler optimization
- ...
Example

Theories:

- \( \mathcal{I} \): theory of integers
  \[ \Sigma_{\mathcal{I}} = \{ \leq, +, -, 0, 1 \} \]

- \( \mathcal{L} \): theory of lists
  \[ \Sigma_{\mathcal{L}} = \{ =, \text{hd}, \text{tl}, \text{nil}, \text{cons} \} \]

- \( \mathcal{E} \): theory of equality

\( \Sigma \): free function and predicate symbols

Problem: Is

\[ x \leq y \land y \leq x + \text{hd}(\text{cons}0\text{nil}) \land P(fx - fy) \land \neg P0 \]

satisfiable in \( \mathcal{I} \cup \mathcal{L} \cup \mathcal{E} \)?
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 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/](http://goedel.cs.uiowa.edu/smtlib/)

Greatly helped to **unify** the field!
### Translation from Higher-Order Logic

<table>
<thead>
<tr>
<th>SMT-LIB</th>
<th>Yices</th>
<th>SMT-LIB</th>
<th>Yices</th>
</tr>
</thead>
<tbody>
<tr>
<td>int, real</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>nat, bool, →</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>prop. logic</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>equality</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>FOL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>HOL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>arithmetic</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

*Abstraction* is used to deal with unsupported terms/types.
Z3’s Proof Format

**Term language:** many-sorted first-order logic

- `bool`, `int`, `real`
- `+`, `−`, `⋅`, `∨`, `∧`, `¬`, `⊤`, `⊥`, `∀`, `∃`, `distinct`, `select`, `store`
Z3’s Proof Format

**Term language:** many-sorted first-order logic

- `bool`, `int`, `real`
- `+`, `−`, `·`, `∨`, `∧`, `¬`, `⊤`, `⊥`, `∀`, `∃`, `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)

```plaintext
#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.
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!**
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.
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. → **TautProve**
Rapid Prototyping

About one third of Z3’s proof rules perform propositional reasoning. → **TautProve**

About one third of Z3’s proof rules perform relatively simple first-order reasoning. → **Metis**
Rapid Prototyping

About one third of Z3’s proof rules perform propositional reasoning. → **TautProve**

About one third of Z3’s proof rules perform relatively simple first-order reasoning. → **Metis**

Slow!
Rapid Prototyping

About one third of Z3’s proof rules perform propositional reasoning. → **TAUTPROVE**

About one third of Z3’s proof rules perform relatively simple first-order reasoning. → **METIS**

Slow!

Speedups of several orders of magnitude can be achieved through specialized implementations that perform the required inferences directly.
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!)
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 CONJ:** $\bigwedge_{i=1}^{n} \varphi_i \vdash \varphi_1, \ldots, \bigwedge_{i=1}^{n} \varphi_i \vdash \varphi_n$
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 CONJ E:** $\bigwedge_{i=1}^{n} \varphi_i \vdash \varphi_1, \ldots, \bigwedge_{i=1}^{n} \varphi_i \vdash \varphi_n$
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: $\bigwedge_{i=1}^{n} \varphi_i \vdash \bigwedge_{i=1}^{n} \varphi_{\pi(i)}$
Nested disjunctions: dual to nested conjunctions, but trickier
Nested disjunctions: **dual** to nested conjunctions, but trickier

Unit resolution:

\[
\frac{\Gamma \vdash \bigvee_{i \in I} \varphi_i}{\Gamma \cup \bigcup_{i \in J} \Gamma_i \vdash \bigvee_{i \in I \setminus J} \varphi_i}
\]

similar to nested disjunctions
Nested disjunctions: dual to nested conjunctions, but trickier

Unit resolution: \[ \frac{\Gamma \vdash \bigvee_{i \in I} \varphi_i}{\bigcup_{i \in J} \Gamma \vdash \bigvee_{i \in I \setminus J} \varphi_i} \] 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∼, 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

<table>
<thead>
<tr>
<th>Logic</th>
<th>Solved (Z3)</th>
<th>Reconstructed</th>
<th>Failed</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#</td>
<td>Time</td>
<td>#</td>
<td>Time</td>
</tr>
<tr>
<td>AUFLIA</td>
<td>100</td>
<td>0.180 s</td>
<td>100</td>
<td>0.407 s</td>
</tr>
<tr>
<td>AUFLIRA</td>
<td>100</td>
<td>0.051 s</td>
<td>97</td>
<td>0.038 s</td>
</tr>
<tr>
<td>QF_UF</td>
<td>96</td>
<td>2.992 s</td>
<td>74</td>
<td>16.618 s</td>
</tr>
<tr>
<td>QF_UFLIA</td>
<td>99</td>
<td>0.534 s</td>
<td>92</td>
<td>5.889 s</td>
</tr>
<tr>
<td>QF_UFLRA</td>
<td>100</td>
<td>0.189 s</td>
<td>100</td>
<td>1.673 s</td>
</tr>
</tbody>
</table>
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

- blue: rewrite
- orange: def_axiom
- yellow: parse_proof_file
- green: th_lemma
- maroon: monotonicity
- other

Tjark Weber
Integrating SAT and SMT Solvers with ITPs
Profiling: AUFLIRA

- rewrite
- parse_proof_file
- sk
- quant_intro
- monotonicity
- other
Profiling: QF_UF

- unit_resolution
- parse_proof_file
- lemma
- monotonicity
- trans
- other
Profiling: QF_UFLIA

- th_lemma
- parse_proof_file
- other
Profiling: QF_UFLRA

- **rewrite**
- **parse_proof_file**
- **monotonicity**
- **th_lemma**
- **and_elim**
- **other**
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
Thank you for your attention.