Efficiently Checking Propositional Resolution Proofs in Isabelle/HOL

Tjark Weber
webertj@in.tum.de

TECHNISCHE UNIVERSITÄT MÜNCHEN

6th International Workshop on the Implementation of Logics
November 12, 2006

Isabelle/HOL

- Isabelle is a generic theorem prover.

Isabelle/HOL

- Isabelle is a generic theorem prover.
- Isabelle/HOL provides a rich specification language.

Isabelle/HOL

- Isabelle is a generic theorem prover.
- Isabelle/HOL provides a rich specification language.
- Isabelle/HOL offers a reasonable degree of automation.

Isabelle/HOL

- Isabelle is a generic theorem prover.
- Isabelle/HOL provides a rich specification language.
- Isabelle/HOL offers a reasonable degree of automation.
- Isabelle/HOL is used for hardware and software verification.

Motivation

- Verification problems can often be reduced to Boolean satisfiability.

Motivation

- Verification problems can often be reduced to Boolean satisfiability.
- Recent SAT solver advances have made this approach feasible in practice.

Motivation

- Verification problems can often be reduced to Boolean satisfiability.
- Recent SAT solver advances have made this approach feasible in practice.

Can an LCF-style theorem prover benefit from these advances?

Motivation

- Verification problems can often be reduced to Boolean satisfiability.
- Recent SAT solver advances have made this approach feasible in practice.

Can an LCF-style theorem prover benefit from these advances?

Can we increase the degree of automation in Isabelle/HOL while keeping the trusted code base small?

In Tools We Trust?

The Oracle Approach

A formula is *accepted* as a theorem if the external tool claims it to be provable.

In Tools We Trust?

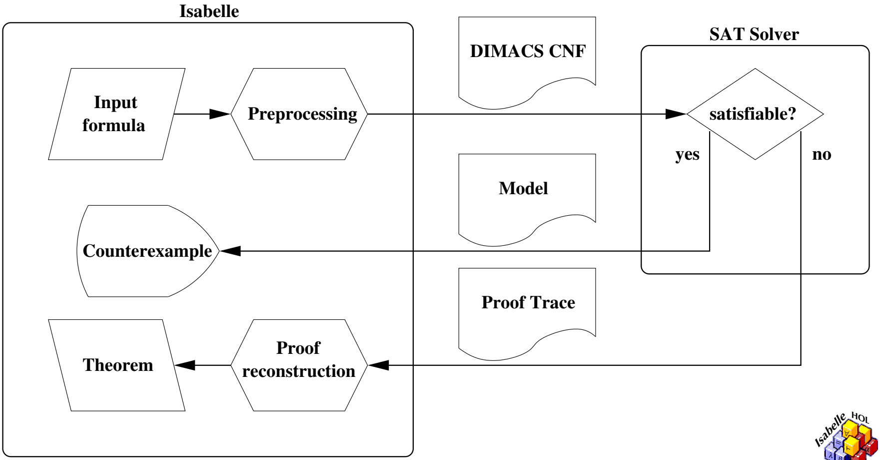
The Oracle Approach

A formula is *accepted* as a theorem if the external tool claims it to be provable.

The LCF-style Approach

The external tool provides a *certificate* of its answer that is translated into an Isabelle proof.

System Overview



Preprocessing

- The input formula is negated.

Preprocessing

- The input formula is negated.
- The negated input formula is transformed into CNF.
  - Naive CNF transformation
  - Definitional CNF

Preprocessing

- The input formula is negated.
- The negated input formula is transformed into CNF.
  - Naive CNF transformation
  - Definitional CNF

The CNF transformation must be proof-producing. The result is not just a CNF formula ϕ^*, but a theorem $\vdash \phi = \phi^*$.

SAT Solvers

zChaff, MiniSat

SAT Solvers

zChaff, MiniSat

- leading SAT solvers (winner of recent SAT competitions in several categories)

SAT Solvers

zChaff, MiniSat

- leading SAT solvers (winner of recent SAT competitions in several categories)
- zChaff: developed by Sharad Malik and Zhaohui Fu, Princeton University

SAT Solvers

zChaff, MiniSat

- leading SAT solvers (winner of recent SAT competitions in several categories)
- zChaff: developed by Sharad Malik and Zhaohui Fu, Princeton University
- MiniSat: developed by Niklas Eén and Niklas Sörensson, Chalmers University

SAT Solvers

zChaff, MiniSat

- leading SAT solvers (winner of recent SAT competitions in several categories)
- zChaff: developed by Sharad Malik and Zhaohui Fu, Princeton University
- MiniSat: developed by Niklas Eén and Niklas Sörensson, Chalmers University
- return a satisfying assignment, or . . .

SAT Solvers

zChaff, MiniSat

- leading SAT solvers (winner of recent SAT competitions in several categories)
- zChaff: developed by Sharad Malik and Zhaohui Fu, Princeton University
- MiniSat: developed by Niklas Eén and Niklas Sörensson, Chalmers University
- return a satisfying assignment, or . . .
- . . . a proof of unsatisfiability (since 2003 (zChaff)/2006 (MiniSat))

Proof Formats

The proofs generated by zChaff and MiniSat differ in detail, but both are based on the propositional resolution rule.

Proof Formats

The proofs generated by zChaff and MiniSat differ in detail, but both are based on the propositional resolution rule.

Propositional Resolution

$$\frac{P \vee x \quad Q \vee \neg x}{P \vee Q}$$

Proof Formats

The proofs generated by zChaff and MiniSat differ in detail, but both are based on the propositional resolution rule.

Propositional Resolution

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

Proofs: Internal Representation

```
type proof = int list Inttab.table * int
```

- "Clause n is the result of resolving clauses $n_1, \ldots, n_k$."
- "Clause m is the empty clause."

Isabelle's Previous Automation (on TPTP)

Problem	Status	auto	blast	fast
MSC007-1.008	unsat.	x	x	x
NUM285-1	sat.	x	x	x
PUZ013-1	unsat.	0.5	x	5.0
PUZ014-1	unsat.	1.4	x	6.1
PUZ015-2.006	unsat.	x	x	x
PUZ016-2.004	sat.	x	x	x
PUZ016-2.005	unsat.	x	x	x
PUZ030-2	unsat.	x	x	x
PUZ033-1	unsat.	0.2	6.4	0.1
SYN001-1.005	unsat.	x	x	x
SYN003-1.006	unsat.	0.9	x	1.6
SYN004-1.007	unsat.	0.3	822.2	2.8
SYN010-1.005.005	unsat.	x	x	x
SYN086-1.003	sat.	x	x	x
SYN087-1.003	sat.	x	x	x
SYN090-1.008	unsat.	13.8	x	x
SYN091-1.003	sat.	x	x	x
SYN092-1.003	sat.	x	x	x
SYN093-1.002	unsat.	1290.8	16.2	1126.6
SYN094-1.005	unsat.	x	x	x
SYN097-1.002	unsat.	x	19.2	x
SYN098-1.002	unsat.	x	x	x
SYN302-1.003	sat.	x	x	x

A Naive Approach (Weber, 2005)

- Start from $\vdash (\phi \Rightarrow \mathit{False}) \Rightarrow (\phi \Rightarrow \mathit{False})$.

A Naive Approach (Weber, 2005)

- Start from $\vdash (\phi \Rightarrow \mathit{False}) \Rightarrow (\phi \Rightarrow \mathit{False})$.
- Reduce the premise $\phi \Rightarrow \mathit{False}$ to True.

A Naive Approach (Weber, 2005)

Problem	Status	auto	blast	fast	zChaff
MSC007-1.008	unsat.	x	x	x	726.5
NUM285-1	sat.	x	x	x	0.2
PUZ013-1	unsat.	0.5	x	5.0	0.1
PUZ014-1	unsat.	1.4	x	6.1	0.1
PUZ015-2.006	unsat.	x	x	x	10.5
PUZ016-2.004	sat.	x	x	x	0.3
PUZ016-2.005	unsat.	x	x	x	1.6
PUZ030-2	unsat.	x	x	x	0.7
PUZ033-1	unsat.	0.2	6.4	0.1	0.1
SYN001-1.005	unsat.	x	x	x	0.4
SYN003-1.006	unsat.	0.9	x	1.6	0.1
SYN004-1.007	unsat.	0.3	822.2	2.8	0.1
SYN010-1.005.005	unsat.	x	x	x	0.4
SYN086-1.003	sat.	x	x	x	0.1
SYN087-1.003	sat.	x	x	x	0.1
SYN090-1.008	unsat.	13.8	x	x	0.5
SYN091-1.003	sat.	x	x	x	0.1
SYN092-1.003	sat.	x	x	x	0.1
SYN093-1.002	unsat.	1290.8	16.2	1126.6	0.1
SYN094-1.005	unsat.	x	x	x	0.8
SYN097-1.002	unsat.	x	19.2	x	0.2
SYN098-1.002	unsat.	x	x	x	0.4
SYN302-1.003	sat.	x	x	x	0.4

A Naive Approach (Weber, 2005)

- Start from $\vdash (\phi \Rightarrow \mathit{False}) \Rightarrow (\phi \Rightarrow \mathit{False})$.
- Reduce the premise $\phi \Rightarrow \mathit{False}$ to True.

- A huge improvement over Isabelle's previous automation ...

A Naive Approach (Weber, 2005)

- Start from $\vdash (\phi \Rightarrow \mathit{False}) \Rightarrow (\phi \Rightarrow \mathit{False})$.
- Reduce the premise $\phi \Rightarrow \mathit{False}$ to True.

- A huge improvement over Isabelle's previous automation . . .
- . . . but still not satisfactory:

Problem	Status	auto	blast	fast	zChaff
MSC007-1.008	unsat.	x	x	x	726.5

A Naive Approach (Weber, 2005)

- Start from $\vdash (\phi \Rightarrow \mathit{False}) \Rightarrow (\phi \Rightarrow \mathit{False})$.
- Reduce the premise $\phi \Rightarrow \mathit{False}$ to True.

- A huge improvement over Isabelle's previous automation ...
- ... but still not satisfactory:

Problem	Status	auto	blast	fast	zChaff
MSC007-1.008	unsat.	**x**	**x**	**x**	726.5

- Explicit treatment of associativity and commutativity for $\vee$, $\wedge$ required.

The Main Question

How to check propositional resolution proofs in Isabelle/HOL efficiently?

The Main Question

How to check propositional resolution proofs in Isabelle/HOL efficiently?

Theorems in Isabelle/HOL

A theorem is a sequent $\Gamma \vdash \phi$, where Γ is a finite set of hypotheses.

The Main Question

How to check propositional resolution proofs in Isabelle/HOL efficiently?

Theorems in Isabelle/HOL

A theorem is a sequent $\Gamma \vdash \phi$, where Γ is a finite set of hypotheses.

$$\frac{}{\{\phi\} \vdash \phi}\text{Assume} \qquad \frac{\Gamma \vdash \psi}{\Gamma \setminus \phi \vdash \phi \Longrightarrow \psi}\text{impI} \qquad \frac{\Gamma \vdash \phi \Longrightarrow \psi \quad \Gamma' \vdash \phi}{\Gamma \cup \Gamma' \vdash \psi}\text{impE}$$

Separate Clauses (Alwen Tiu et al., 2006)

Each clause $p_1 \vee \ldots \vee p_n$ is encoded as a single theorem

$$\{p_1 \vee \ldots \vee p_n\} \vdash \overline{p_1} \Rightarrow \ldots \Rightarrow \overline{p_n} \Rightarrow \mathrm{False}$$

Separate Clauses (Alwen Tiu et al., 2006)

Each clause $p_1 \vee \ldots \vee p_n$ is encoded as a single theorem

$$\{p_1 \vee \ldots \vee p_n\} \vdash \overline{p_1} \Rightarrow \ldots \Rightarrow \overline{p_n} \Rightarrow \mathrm{False}$$

Resolution is based on a derived Isabelle tactic which performs cuts.

Separate Clauses (Alwen Tiu et al., 2006)

Each clause $p_1 \vee \ldots \vee p_n$ is encoded as a single theorem

$$\{p_1 \vee \ldots \vee p_n\} \vdash \overline{p_1} \Rightarrow \ldots \Rightarrow \overline{p_n} \Rightarrow \text{False}$$

Resolution is based on a derived Isabelle tactic which performs cuts.

- Proof reconstruction for MSC007-1.008: 7.8 s
- The problem is a set of clauses.
- Clauses are not viewed as sets of literals.

Sequent Representation

Each clause $p_1 \vee \ldots \vee p_n$ is encoded as a single theorem

$$\{p_1 \vee \ldots \vee p_n, \overline{p_1}, \ldots, \overline{p_n}\} \vdash \mathrm{False}$$

Sequent Representation

Each clause $p_1 \vee \ldots \vee p_n$ is encoded as a single theorem

$$\{p_1 \vee \ldots \vee p_n, \overline{p_1}, \ldots, \overline{p_n}\} \vdash \mathrm{False}$$

Resolution:

1. impI: $\Gamma_1 := \{p_1 \vee \ldots \vee p_n, \overline{p_1}, \ldots, \overline{p_n}\} \setminus \{p\} \vdash p \Rightarrow \mathrm{False}$
2. impI: $\Gamma_2 := \{q_1 \vee \ldots \vee q_m, \overline{q_1}, \ldots, \overline{q_m}\} \setminus \{\neg p\} \vdash \neg p \Rightarrow \mathrm{False}$
3. instantiate: $\vdash (p \Rightarrow \mathrm{False}) \Rightarrow (\neg p \Rightarrow \mathrm{False}) \Rightarrow \mathrm{False}$
4. impE: $\Gamma_1 \vdash (\neg p \Rightarrow \mathrm{False}) \Rightarrow \mathrm{False}$
5. impE: $\Gamma_1 \cup \Gamma_2 \vdash \mathrm{False}$

Sequent Representation

Each clause $p_1 \vee \ldots \vee p_n$ is encoded as a single theorem

$$\{p_1 \vee \ldots \vee p_n, \overline{p_1}, \ldots, \overline{p_n}\} \vdash \text{False}$$

- Proof reconstruction for MSC007-1.008: 1.2 s
- The problem is a set of clauses.
- Clauses are sets of literals.
- Clause hypotheses accumulate during resolution, until the set of hypotheses eventually contains every clause used in the proof.

CNF Sequent Representation

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 the (modified) sequent representation is used: $\{\bigwedge_{i=1}^{k} C_i, \overline{p_1}, \ldots, \overline{p_n}\} \vdash \mathit{False}$.

CNF Sequent Representation

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 the (modified) sequent representation is used: $\{\bigwedge_{i=1}^{k} C_i, \overline{p_1}, \ldots, \overline{p_n}\} \vdash \mathit{False}$.

- Proof reconstruction for MSC007-1.008: 0.5 s
- The right way to do things.

Further Optimizations

- Pivot search: Instead of searching the hypotheses of clauses to determine the pivot literal for resolutions, we associate our own data structure (an ordered tree of integers) with each clause.

Further Optimizations

- Pivot search: Instead of searching the hypotheses of clauses to determine the pivot literal for resolutions, we associate our own data structure (an ordered tree of integers) with each clause.
- Backwards proof: Instead of chronologically replaying the proof trace, we perform "backwards" proof reconstruction, starting from the empty clause's identifier.

Further Optimizations

- Pivot search: Instead of searching the hypotheses of clauses to determine the pivot literal for resolutions, we associate our own data structure (an ordered tree of integers) with each clause.
- Backwards proof: Instead of chronologically replaying the proof trace, we perform "backwards" proof reconstruction, starting from the empty clause's identifier.
- Lemmas: Instead of proving the same intermediate clause multiple times, we store proven clauses in an array and simply retrieve them from there if they are needed again.

Evaluation on SATLIB Problems

Individual SATLIB problems typically contain several ten thousand variables and several hundred thousand clauses.

SATLIB: Pushing Isabelle to its Limits

- Parser: very general, but unable to parse very large terms in reasonable time.

SATLIB: Pushing Isabelle to its Limits

- Parser: very general, but unable to parse very large terms in reasonable time.
- Solution: we implemented our own little parser for DIMACS files, which turns them directly into ML term values.

SATLIB: Pushing Isabelle to its Limits

- Parser: very general, but unable to parse very large terms in reasonable time.
- Solution: we implemented our own little parser for DIMACS files, which turns them directly into ML term values.
- User interface: nice features (e.g. syntax highlighting), but unable to display very large terms in reasonable time.

SATLIB: Pushing Isabelle to its Limits

- Parser: very general, but unable to parse very large terms in reasonable time.
- Solution: we implemented our own little parser for DIMACS files, which turns them directly into ML term values.
- User interface: nice features (e.g. syntax highlighting), but unable to display very large terms in reasonable time.
- Solution: we worked at the system's ML level, thereby avoiding the additional user interface layer.

SATLIB: Pushing Isabelle to its Limits

- Parser: very general, but unable to parse very large terms in reasonable time.
- Solution: we implemented our own little parser for DIMACS files, which turns them directly into ML term values.
- User interface: nice features (e.g. syntax highlighting), but unable to display very large terms in reasonable time.
- Solution: we worked at the system's ML level, thereby avoiding the additional user interface layer.
- Inference kernel: minor inefficiencies, which became significant when the kernel had to deal with very large terms.

SATLIB: Pushing Isabelle to its Limits

- Parser: very general, but unable to parse very large terms in reasonable time.
- Solution: we implemented our own little parser for DIMACS files, which turns them directly into ML term values.
- User interface: nice features (e.g. syntax highlighting), but unable to display very large terms in reasonable time.
- Solution: we worked at the system's ML level, thereby avoiding the additional user interface layer.
- Inference kernel: minor inefficiencies, which became significant when the kernel had to deal with very large terms.
- Solution: small fixes to the kernel.

Evaluation on SATLIB Problems

Problem	Variables	Clauses
c7552mul.miter	11282	69529
6pipe	15800	394739
6pipe_6_ooo	17064	545612
7pipe	23910	751118

Evaluation on SATLIB Problems

Problem	Variables	Clauses
c7552mul.miter	11282	69529
6pipe	15800	394739
6pipe_6_ooo	17064	545612
7pipe	23910	751118

Problem	zChaff (s)	Proof (s)	Resolutions	Total (s)
c7552mul.miter	73	70	252200	145
6pipe	167	321	268808	512
6pipe_6_ooo	308	2575	870345	3179
7pipe	495	1132	357136	1768

Conclusions

- Isabelle's automation for propositional logic has been greatly enhanced.
- Efficient proof checking for propositional logic is possible in a general LCF-style system.
- Our implementation scales well to proofs with hundreds of thousands of resolution steps.
- Our techniques are applicable to other interactive provers, e.g. to HOL 4 and HOL-Light.

Future Work

- Analysis and optimization of resolution proofs
- SAT-based decision procedures beyond propositional logic (e.g. SMT)
- Standard proof formats (for propositional logic and beyond)