Integrating a SAT Solver with Isabelle/HOL

Tjark Weber (joint work with Alwen Tiu et al.)

webertj@in.tum.de

First Munich-Nancy Workshop on

Decision Procedures for Theorem Provers

March 6th & 7th, 2006

Motivation

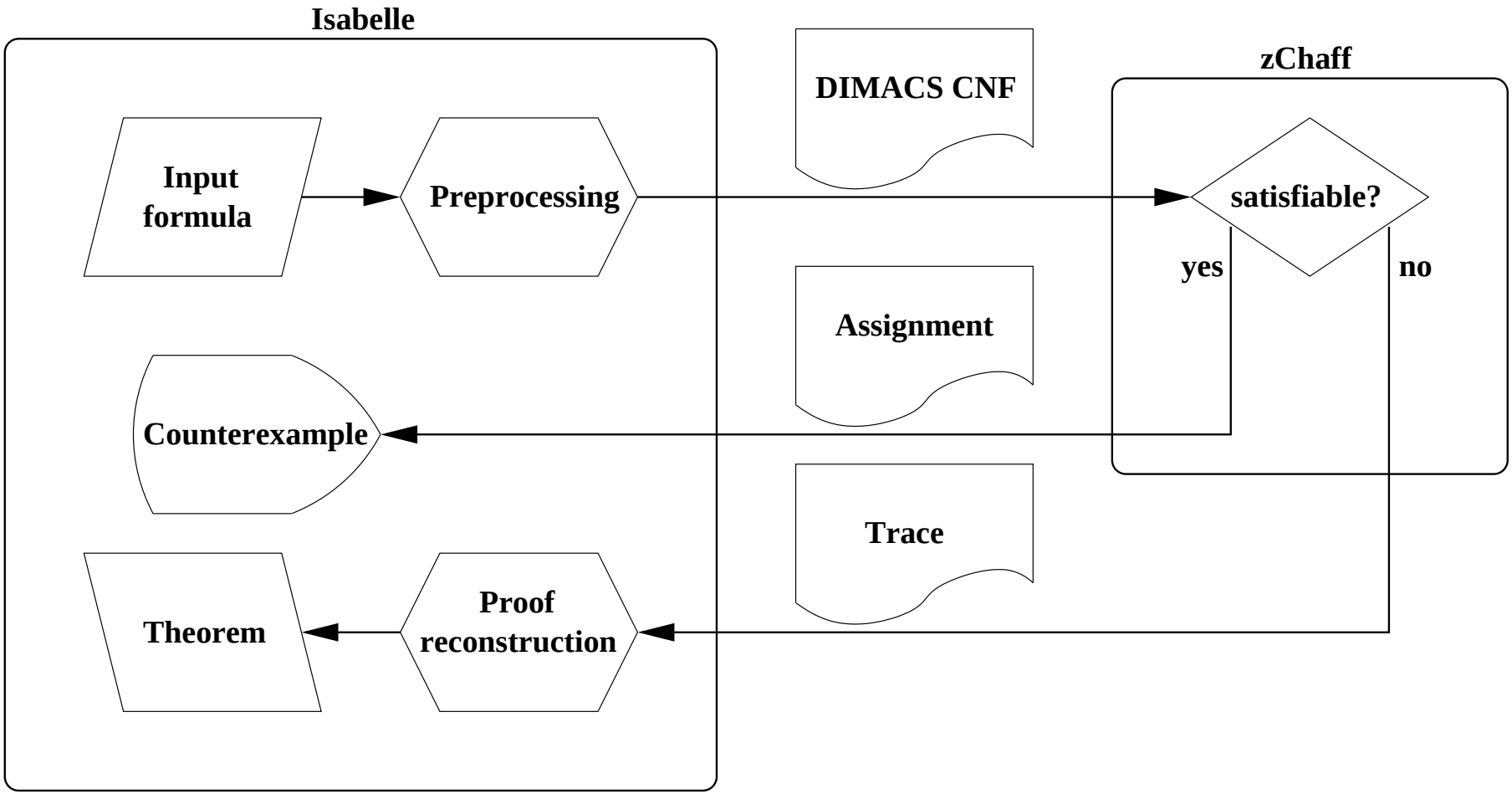
- Verification problems can often be reduced to Boolean satisfiability.
- Recent SAT solver advances have made this approach feasible in practice.

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

zChaff

- A leading SAT solver (winner of the SAT 2002 and SAT 2004 competitions in several categories)
- Developed by Sharad Malik and Zhaohui Fu, Princeton University
- Returns a satisfying assignment, or . . .
- . . . a proof of unsatisfiability (since 2003)

System Overview



Preprocessing

Input: propositional formula ϕ

- CNF conversion
- Normalization
- Removal of duplicate literals
- Removal of tautological clauses

Output: a theorem of the form $\phi = \phi^*$

The SAT Solver's Trace

```
CL: 184 <= 173 28 35 142 154
CL: 185 <= 43 4 11 59 55
[...]
VAR: 16 L: 35 V: 0 A: 55 Lits: 29 33
VAR: 26 L: 28 V: 1 A: 202 Lits: 52 98 57
[...]
CONF: 206 == 80 82 64 70 37
```

The SAT Solver's Trace

clause id resolvents

```
CL: 184 <= 173 28 35 142 154
CL: 185 <= 43 4 11 59 55
[...]
VAR: 16 L: 35 V: 0 A: 55 Lits: 29 33
VAR: 26 L: 28 V: 1 A: 202 Lits: 52 98 57
[...]
CONF: 206 == 80 82 64 70 37
```

variable id antecedent

conflict clause id

The SAT Solver's Trace

clause id resolvents

```
CL: 184 <= 173 28 35 142 154
CL: 185 <= 43 4 11 59 55
[...]
VAR: 16 L: 35 V: 0 A: 55 Lits: 29 33
VAR: 26 L: 28 V: 1 A: 202 Lits: 52 98 57
[...]
CONF: 206 == 80 82 64 70 37
```

variable id antecedent

conflict clause id

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

The Intermediate Proof Format

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

- Each integer is a clause identifier.
- Each list of integers gives the resolvents for the associated key.
- No circular dependencies between clauses.
- (At least) one clause must be the empty clause.
- Its ID is given by the second component of the proof.
- Clauses of the original problem are numbered consecutively, starting from 0.

Proof Reconstruction

- `resolution : Thm.thm list -> Thm.thm`

- `prove_clause : int -> Thm.thm`

- `replay_proof : (Thm.thm option) array`
  `-> SatSolver.proof -> Thm.thm`

Proof Reconstruction

- `resolution : Thm.thm list -> Thm.thm`

  Input: $[\![x_1; \ldots; a; \ldots; x_n]\!] \Longrightarrow$ `False`

  $[\![y_1; \ldots; \neg a; \ldots; y_m]\!] \Longrightarrow$ `False`

  Result: $[\![x_1; \ldots; x_n; y_1; \ldots; y_m]\!] \Longrightarrow$ `False`

- `prove_clause : int -> Thm.thm`

- `replay_proof : (Thm.thm option) array`
  `-> SatSolver.proof -> Thm.thm`

Proof Reconstruction

- `resolution : Thm.thm list -> Thm.thm`

  Input: $[\![x_1; \ldots; a; \ldots; x_n]\!] \Longrightarrow$ `False`

  $[\![y_1; \ldots; \neg a; \ldots; y_m]\!] \Longrightarrow$ `False`

  Result: $[\![x_1; \ldots; x_n; y_1; \ldots; y_m]\!] \Longrightarrow$ `False`

- `prove_clause : int -> Thm.thm`

```
prove_clause clause_id =
  resolution (map prove_clause
    (resolvents_of clause_id))
```

- `replay_proof : (Thm.thm option) array -> SatSolver.proof -> Thm.thm`

Proof Reconstruction

- `resolution : Thm.thm list -> Thm.thm`

  Input: $[\![x_1; \ldots; a; \ldots; x_n]\!] \Longrightarrow$ `False`

  $[\![y_1; \ldots; \neg a; \ldots; y_m]\!] \Longrightarrow$ `False`

  Result: $[\![x_1; \ldots; x_n; y_1; \ldots; y_m]\!] \Longrightarrow$ `False`

- `prove_clause : int -> Thm.thm`

```
prove_clause clause_id =
  resolution (map prove_clause
    (resolvents_of clause_id))
```

- `replay_proof : (Thm.thm option) array -> SatSolver.proof -> Thm.thm`

```
prove_clause empty_clause_id
```

Evaluation

- Isabelle is several orders of magnitude slower than zverify_df.
- However, zChaff vs. auto/blast/fast . . .
  - 42 propositional problems in TPTP, v2.6.0
    - 19 “easy” problems, solved in less than a second each by auto, blast, fast, and zChaff
    - 23 harder problems

Performance (1)

Problem	Status	auto	blast	fast	sat
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

Improvements

- Sequent representation of clauses:

  $l_1 \vee \ldots \vee l_n$ is equivalent to $[\![\neg l_1; \ldots; \neg l_n]\!] \Longrightarrow \texttt{False}$

- Proof reconstruction for needed clauses only:

  starting with the conflict clause, resolvents are proved recursively and stored in an array

- Minor implementation optimizations:

  bookkeeping instead of “trial and error” during resolution

Performance (2)

Problem	Status	sat (naive)	sat (sequents)	sat (now)
MSC007-1.008	unsat.	726.5	11.5	7.9
PUZ015-2.006	unsat.	10.5	2.4	0.7
PUZ016-2.005	unsat.	1.6	1.2	0.6
PUZ030-2	unsat.	0.7	0.5	0.4
PUZ033-1	unsat.	0.1	0.1	0.1
SYN090-1.008	unsat.	0.5	0.5	0.3
SYN093-1.002	unsat.	0.1	0.1	0.1
SYN094-1.005	unsat.	0.8	0.7	0.5

Conclusions and Future Work

- A fast decision procedure for propositional logic
- Counterexamples for unprovable formulae

- Huge SAT problems are still out of scope
- Integration of other SAT solvers
- Integration of an incremental SAT solver
- Extension to (fragments of) richer logics