Integrating a SAT Solver with an LCF-style Theorem Prover

A Fast Decision Procedure for Propositional Logic for the Isabelle System

Tjark Weber

webertj@in.tum.de

TECHNISCHE UNIVERSITÄT MÜNCHEN

PDPAR'05, July 12, 2005

Motivation

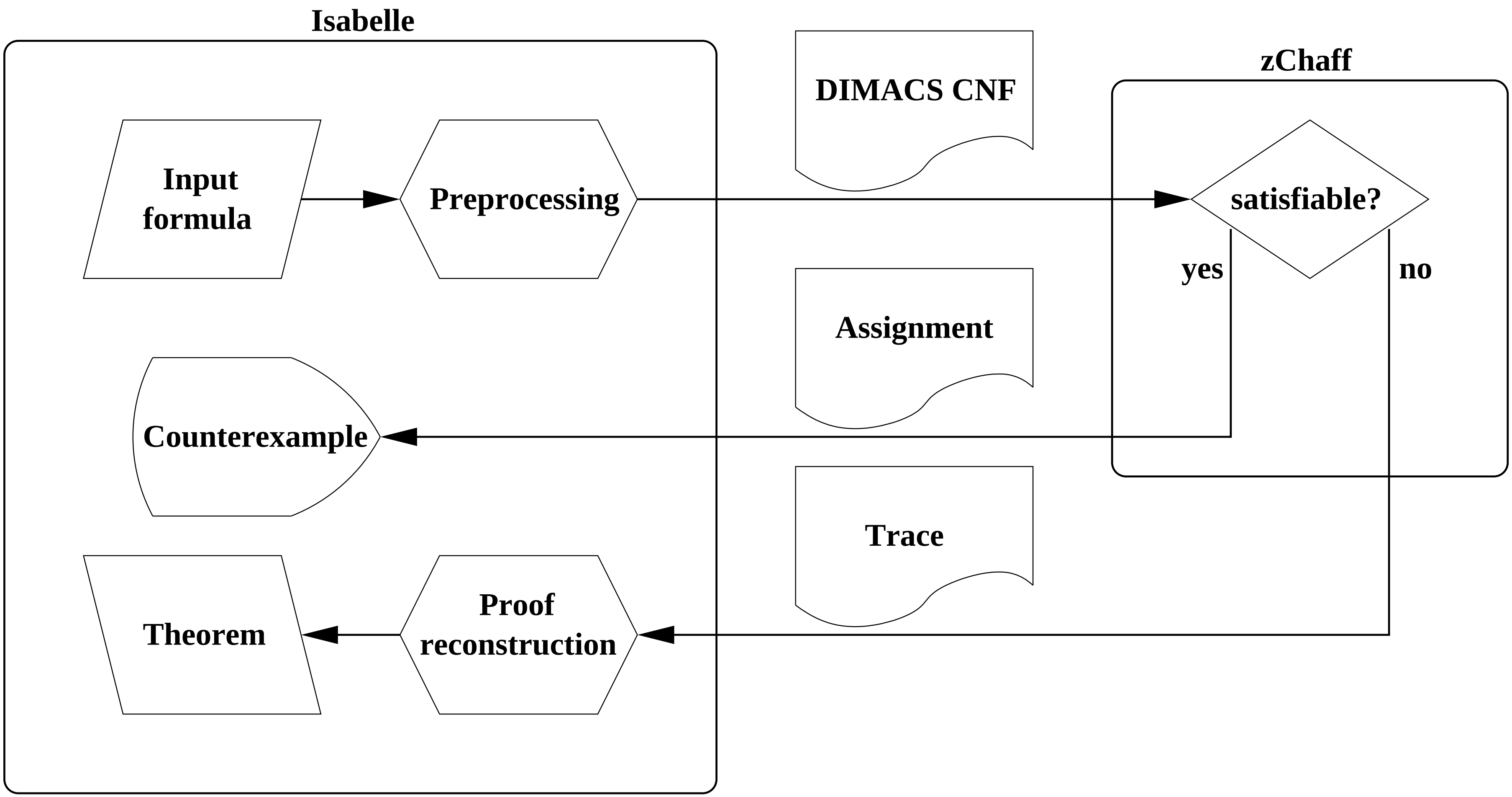
- Verification problems can often be reduced to Boolean satisfiability.
- Recent SAT solver advances have made this approach feasible in practice.

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 $\phi = \phi^*$

```
thm_of decomp t =
  let
    (ts, recomb) = decomb t
  in recomb (map (thm_of decomp) ts)
```

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
[...] variable id          antecedent
CONF: 206 == 80 82 64 70 37
  conflict clause id
```

Proof Reconstruction (1)

- `resolution : Thm.thm list -> Thm.thm`
- `prove_clause : int -> Thm.thm`
- `prove_literal : int -> Thm.thm`

Proof Reconstruction (1)

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

  Input: $[X \longrightarrow P \vee Q \vee R, X \longrightarrow S \vee \neg Q \vee T]$

  Result: $X \longrightarrow P \vee R \vee S \vee T$

- `prove_clause : int -> Thm.thm`

- `prove_literal : int -> Thm.thm`

Proof Reconstruction (1)

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

  Input: $[X \longrightarrow Q, X \longrightarrow \neg Q]$

  Result: $X \longrightarrow \texttt{False}$

- `prove_clause : int -> Thm.thm`

- `prove_literal : int -> Thm.thm`

Proof Reconstruction (1)

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

  Input: $[X \longrightarrow Q, X \longrightarrow \neg Q]$

  Result: $X \longrightarrow$ `False`

- `prove_clause : int -> Thm.thm`

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

- `prove_literal : int -> Thm.thm`

Proof Reconstruction (1)

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

  Input: $[X \longrightarrow Q, X \longrightarrow \neg Q]$

  Result: $X \longrightarrow$ `False`

- `prove_clause : int -> Thm.thm`

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

- `prove_literal : int -> Thm.thm`

```
prove_literal var_id =
  let th_ante = prove_clause (antecedent_of var_id)
      var_ids = filter (λi. i ≠ var_id)
                  (var_ids_in_clause th_ante)
  in resolution
      (th_ante :: map prove_literal var_ids)
```

Proof Reconstruction (2)

- Many clauses may be redundant.
- Clauses and literals may be needed many times.

Proof Reconstruction (2)

- Many clauses may be redundant.
- Clauses and literals may be needed many times.

Two *arrays* store . . .

- each clause's resolvents *or* its proof,
- each variable's antecedent *or* its proof

. . . and are *updated* during proof reconstruction.

Proof Reconstruction (2)

- Many clauses may be redundant.
- Clauses and literals may be needed many times.

Two *arrays* store . . .

- each clause's resolvents *or* its proof,
- each variable's antecedent *or* its proof

. . . and are *updated* during proof reconstruction.

1. Initialize arrays with information from the trace.
2. Prove conflict clause C.
3. Perform resolution with prove_literal for each literal in C.

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

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

Conclusions and Future Work

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

- Huge SAT problems are still out of scope
- Extension to (fragments of) richer logics
- Integration of first-order provers