Integrating zChaff with Isabelle/HOL

A Fast Decision Procedure for Propositional Logic

Tjark Weber

webertj@in.tum.de

TECHNISCHE UNIVERSITÄT MÜNCHEN

Club2, November 17, 2004

Motivation

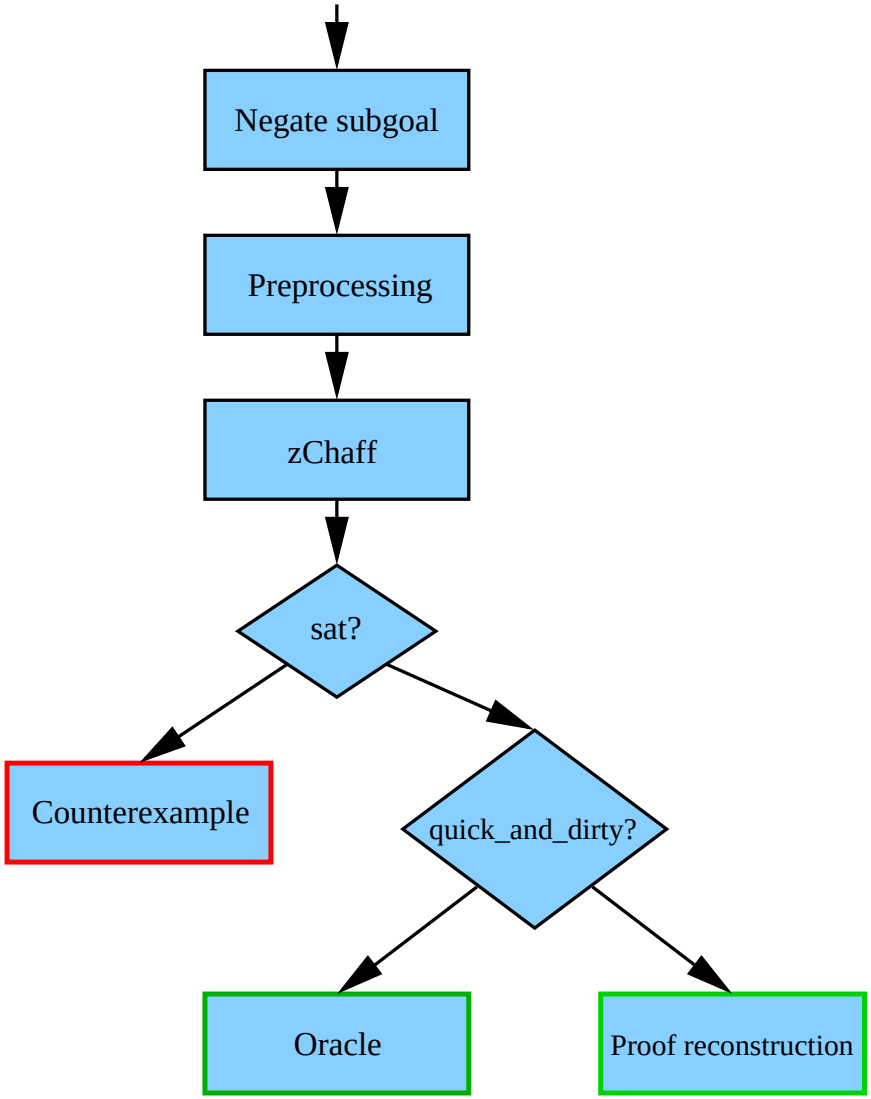
- Verification problems can often be reduced to Boolean satisfiability.
- Recent SAT solver advances have made this feasible in practice.

Can Isabelle benefit from these advances?

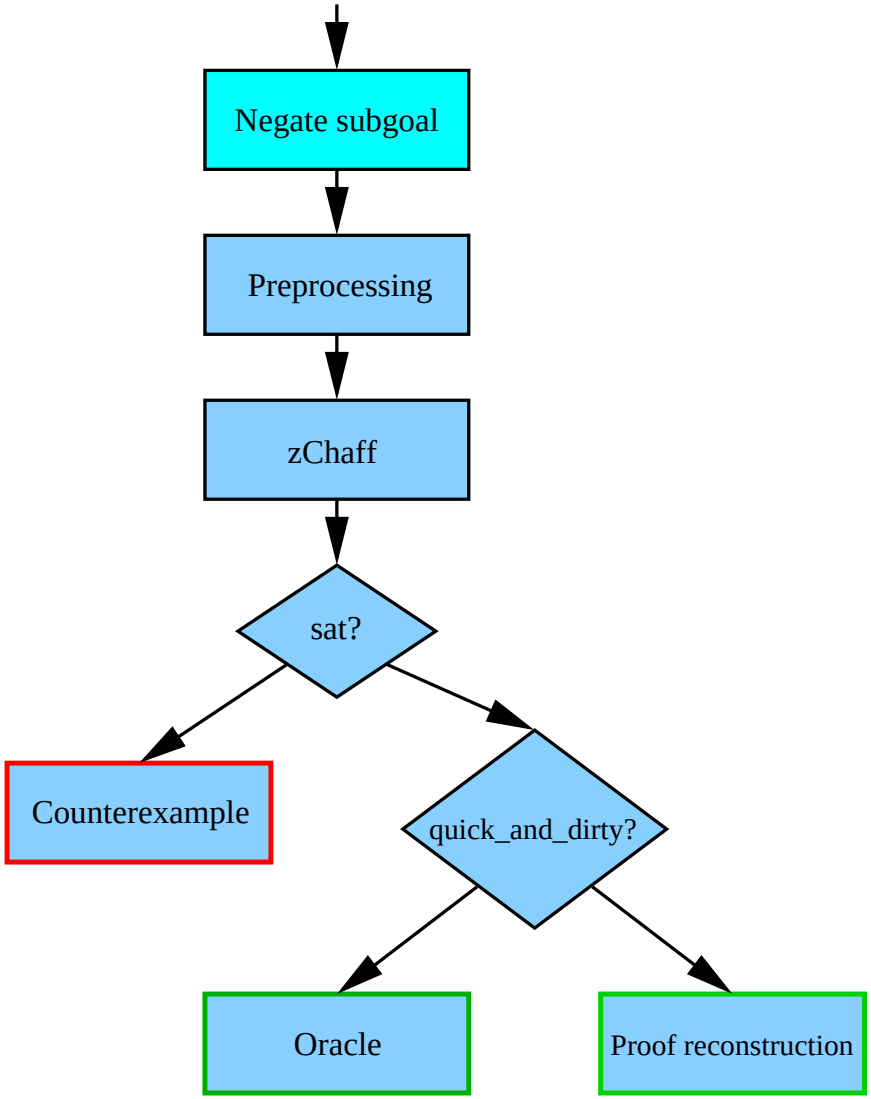
zChaff

- 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)

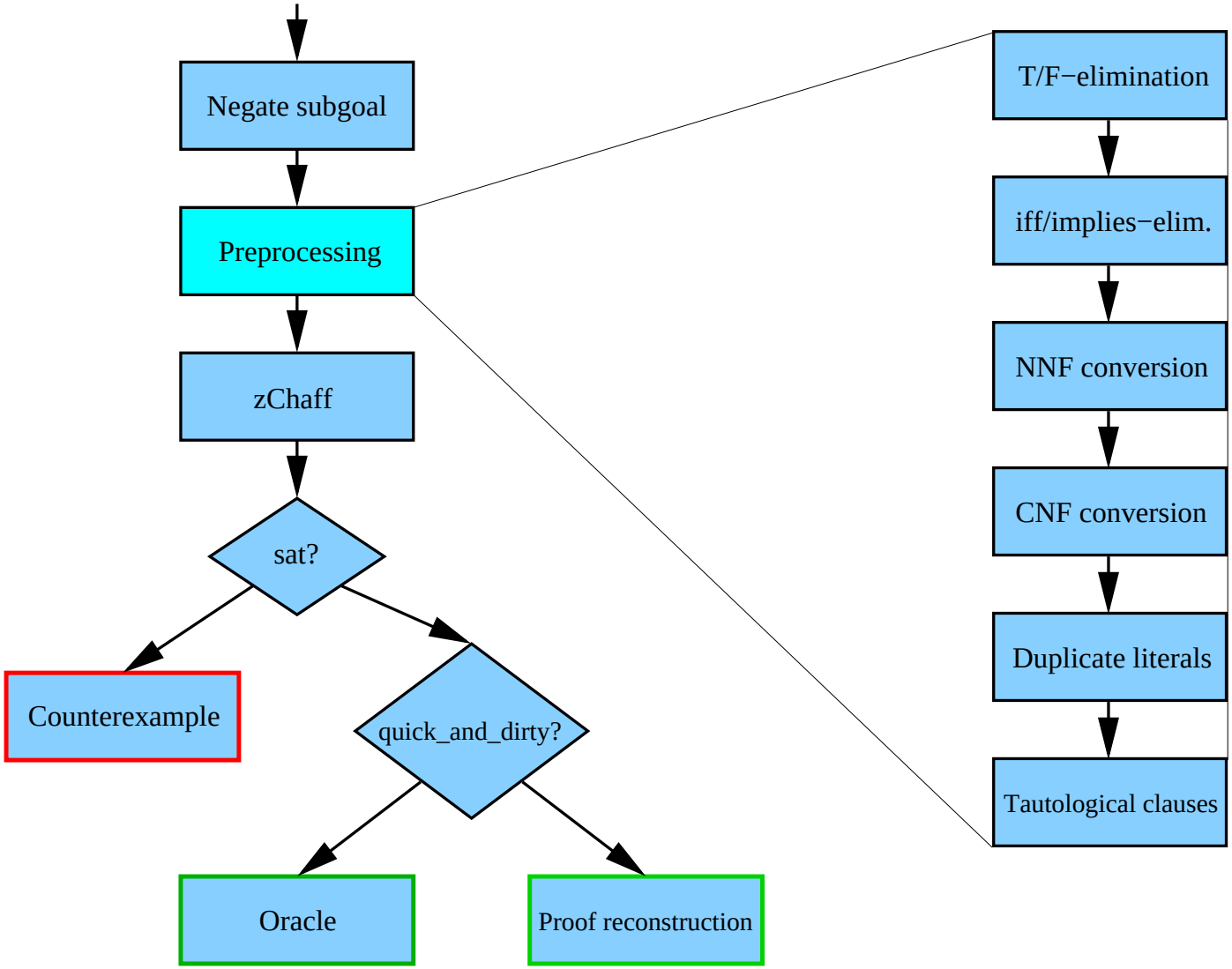
zchaff_tac


Negate subgoal
Preprocessing
zChaff
sat?
Counterexample
quick_and_dirty?
Oracle
Proof reconstruction

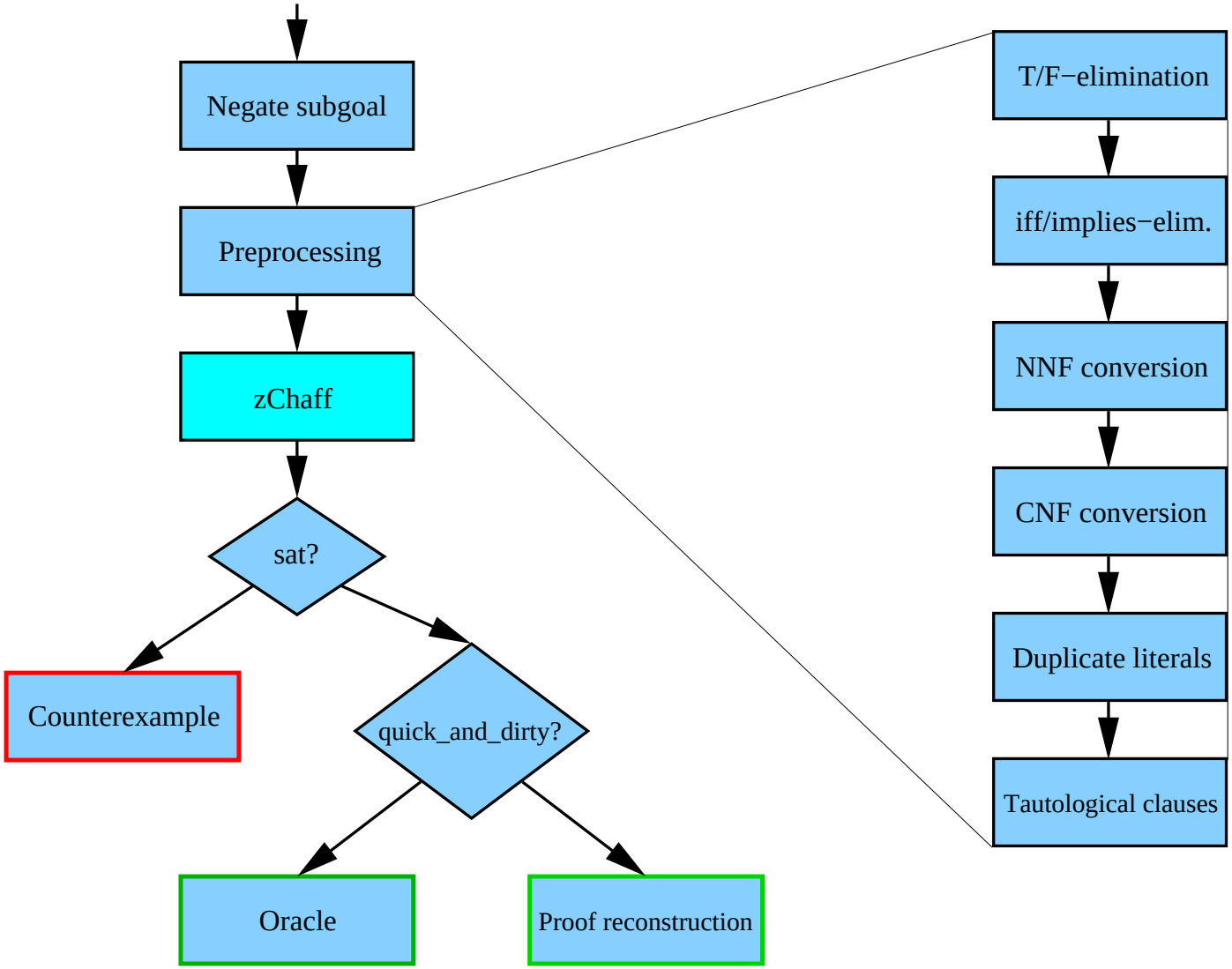
zchaff_tac


Negate subgoal
Preprocessing
zChaff
sat?
Counterexample
quick_and_dirty?
Oracle
Proof reconstruction

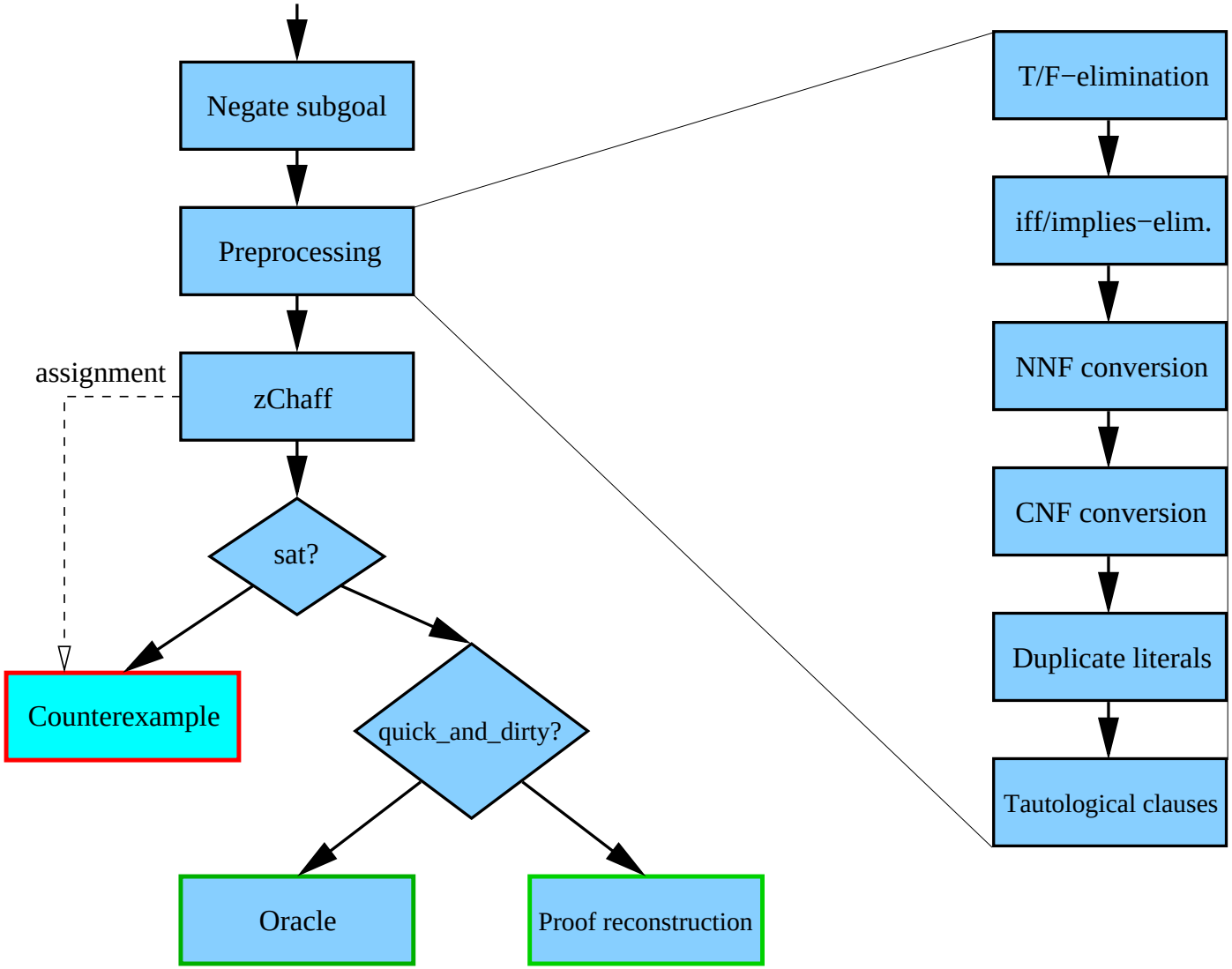
zchaff_tac


Negate subgoal
Preprocessing
zChaff
sat?
Counterexample
quick_and_dirty?
Oracle
Proof reconstruction
T/F−elimination
iff/implies−elim.
NNF conversion
CNF conversion
Duplicate literals
Tautological clauses

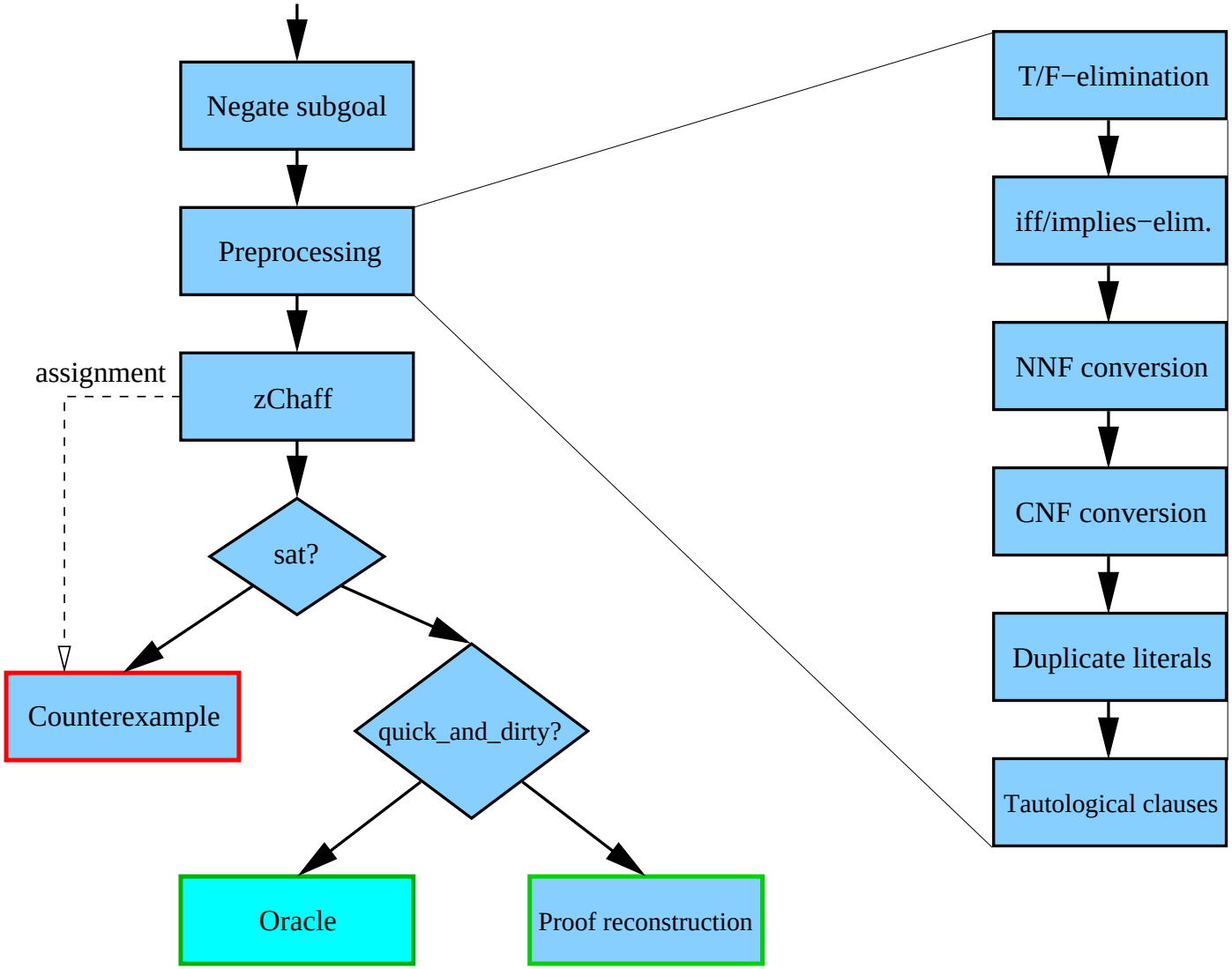
zchaff_tac


Negate subgoal
Preprocessing
zChaff
sat?
Counterexample
quick_and_dirty?
Oracle
Proof reconstruction
T/F−elimination
iff/implies−elim.
NNF conversion
CNF conversion
Duplicate literals
Tautological clauses

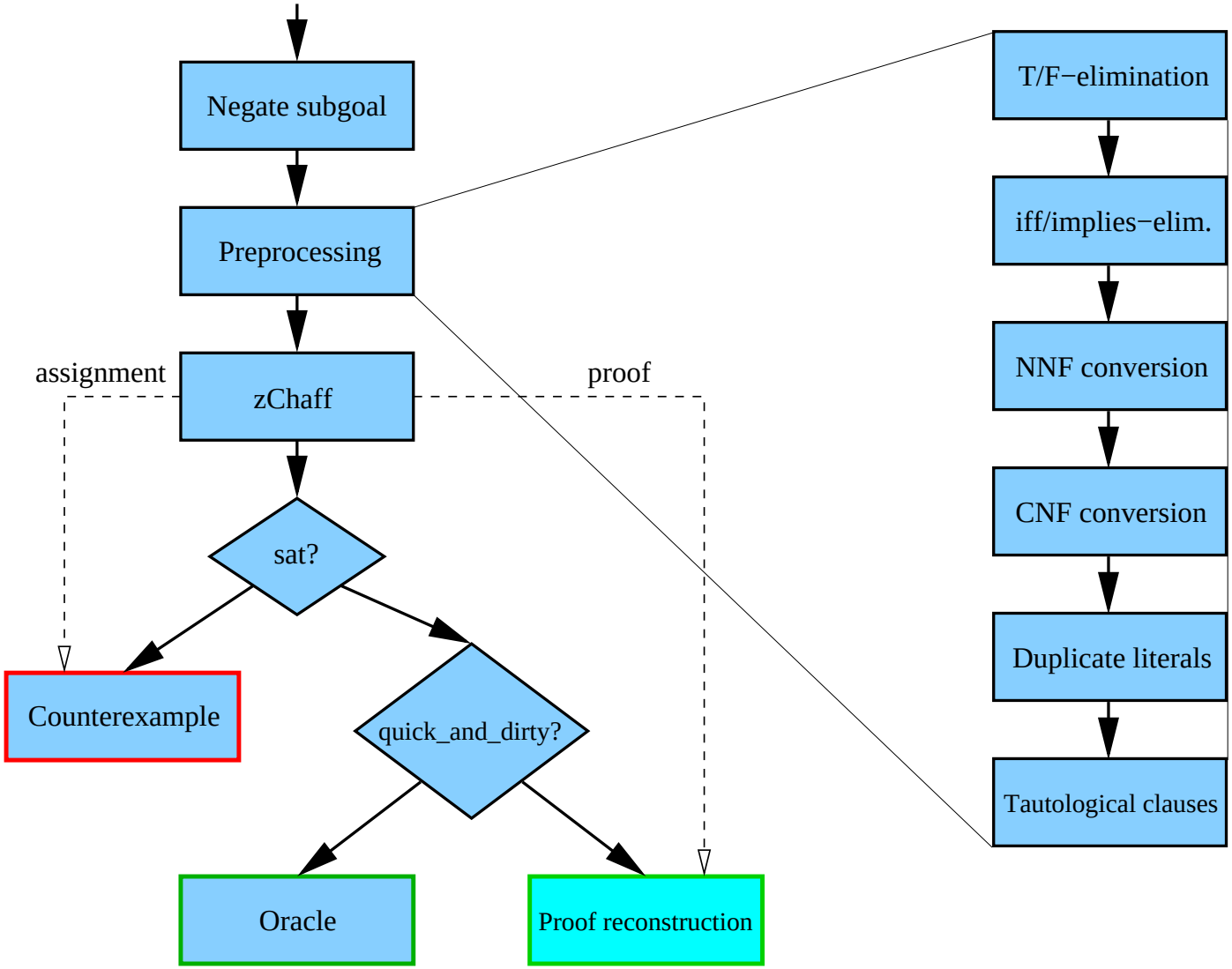
zchaff_tac


Negate subgoal
Preprocessing
assignment
zChaff
sat?
Counterexample
quick_and_dirty?
Oracle
Proof reconstruction
T/F−elimination
iff/implies−elim.
NNF conversion
CNF conversion
Duplicate literals
Tautological clauses

zchaff_tac


Negate subgoal
Preprocessing
assignment
zChaff
sat?
Counterexample
quick_and_dirty?
Oracle
Proof reconstruction
T/F−elimination
iff/implies−elim.
NNF conversion
CNF conversion
Duplicate literals
Tautological clauses

zchaff_tac


Negate subgoal
Preprocessing
assignment
zChaff
proof
sat?
Counterexample
quick_and_dirty?
Oracle
Proof reconstruction
T/F−elimination
iff/implies−elim.
NNF conversion
CNF conversion
Duplicate literals
Tautological clauses

resolve_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
```

resolve_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

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: $X \longrightarrow$ `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`

```
fun prove_clause id =
  resolution (map prove_clause (resolvents 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`

```
fun prove_clause id =
  resolution (map prove_clause (resolvents id));
```

- `prove_literal : int -> Thm.thm`

```
fun prove_literal id =
  let val ante = prove_clause (antecedent id)
      val vars = variables_in_clause ante id in
    resolution (ante :: map prove_literal vars)
  end;
```

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 resolve_trace.
2. Prove conflict clause C.
3. Perform resolution with prove_literal for each literal in C.

Performance

- 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_tac
    - 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 clauses
- Counterexamples for unprovable formulae

- A fast proof-generating CNF conversion
- Huge SAT problems are still out of scope
- Integration of first-order provers