Verifun : A Theorem Prover Using Lazy Proof Explication

Rajeev Joshi

NASA/JPL Laboratory for Reliable Software

Joint work done at Compaq/HP SRC with

Cormac Flanagan, **Jim Saxe** and **Xinming Ou**

Theorem Provers for Static Checking

- Should require little or no user interaction
- Should produce counterexamples
- Should support various theories
  - EUF, linear arithmetic, theory of arrays
  - quantifiers, if possible
- Efficiency is more important than completeness

Theorem Provers using Cooperating Decision Procedures

- Introduced by Nelson and Oppen [TOPLAS 1979]
- Combines decision procedures for a set of disjoint theories, producing a procedure for their union
- Key ideas
  - introduce auxiliary variables to remove mixed application of function symbols
  - theories propagate discovered equalities to each other

Example

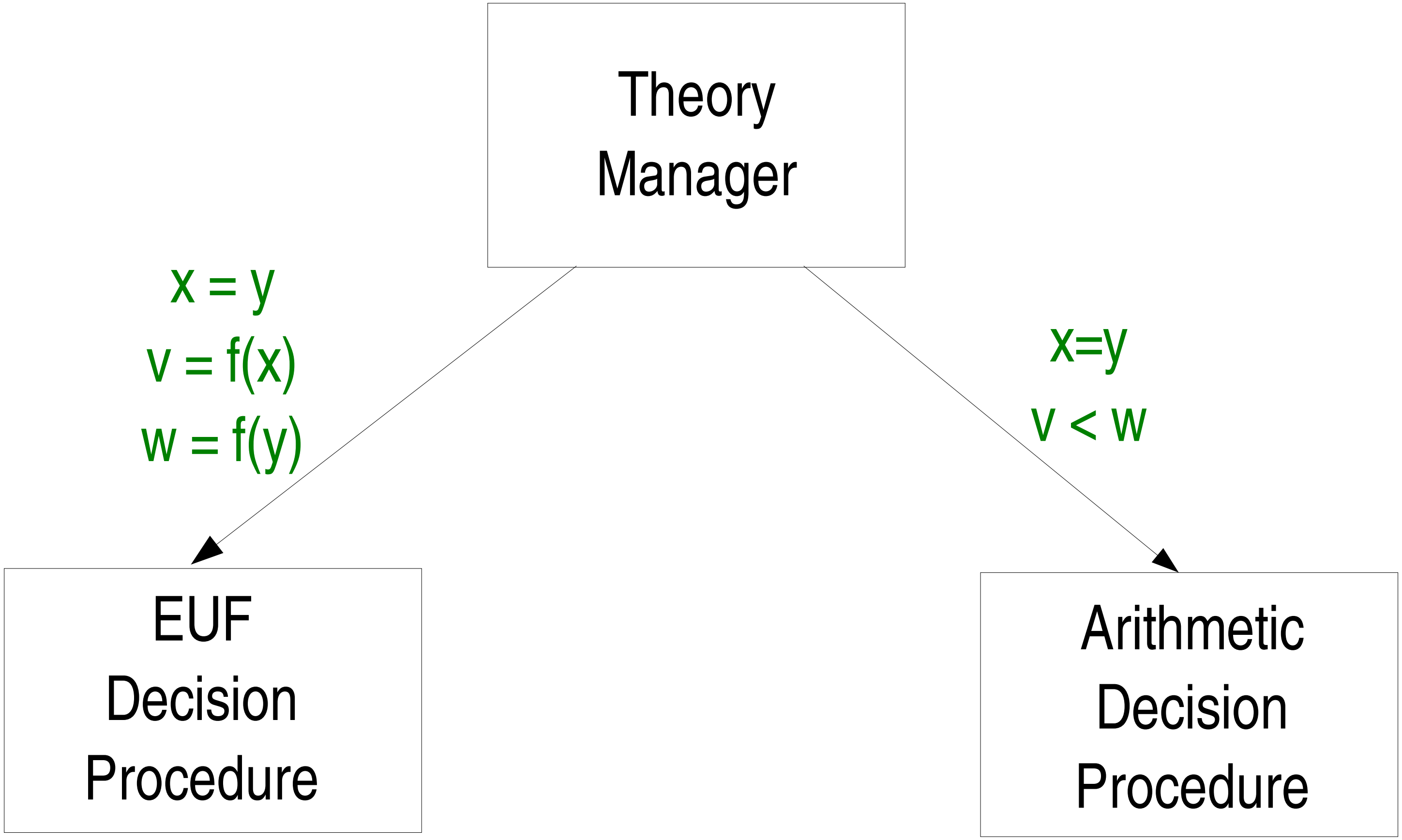
- Suppose we want to check satisfiability of

$$(x = y) \quad \wedge \quad (f(x) < f(y))$$

- Introduce auxiliary variables v, w

$$\begin{aligned} & (x = y) \quad \wedge \quad (v < w) \\ \wedge \quad & (v = f(x)) \quad \wedge \quad (w = f(y)) \end{aligned}$$

Checking $(x = y) \land (f(x) < f(y))$



Theory Manager

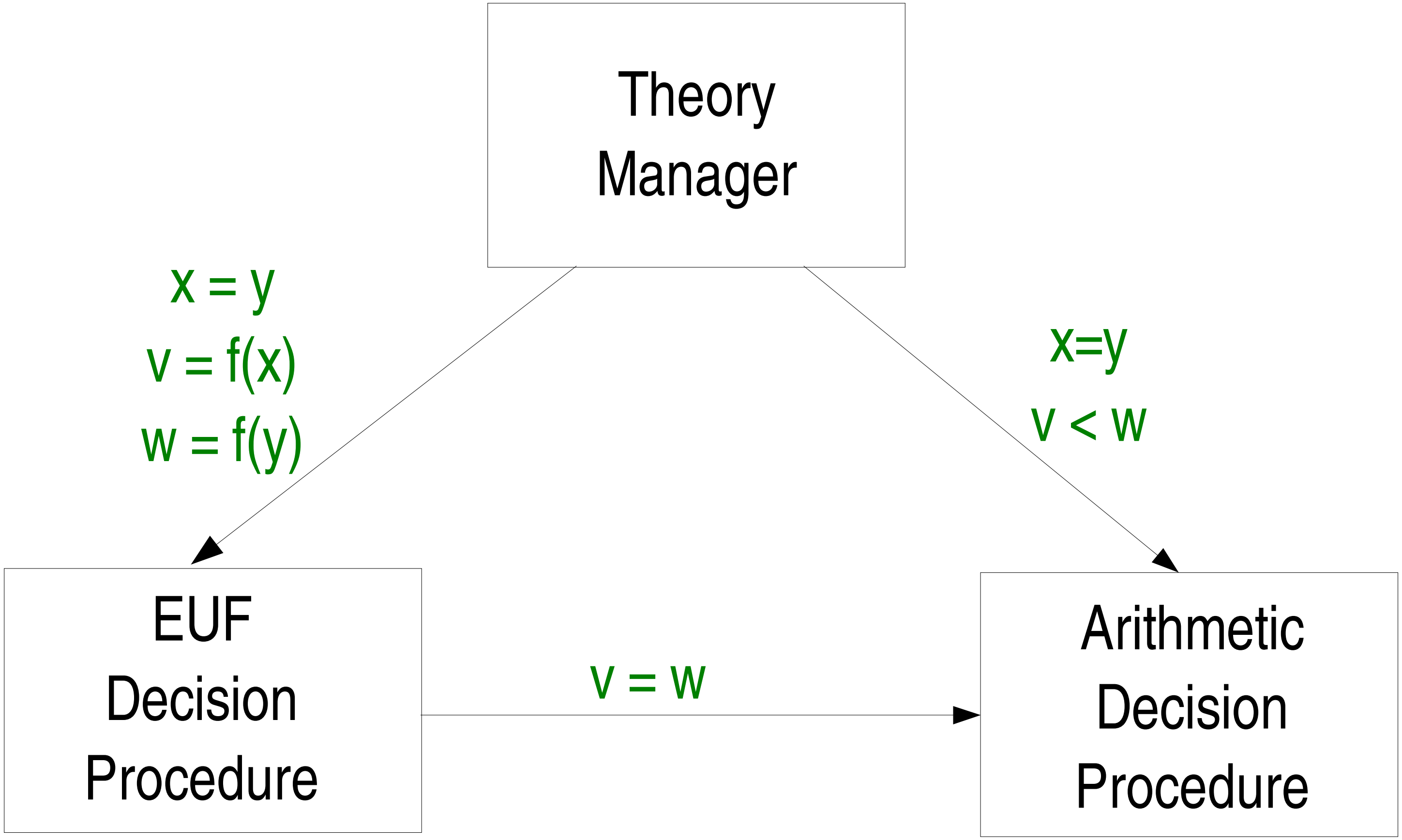
$x = y$
$v = f(x)$
$w = f(y)$

$x=y$
$v < w$

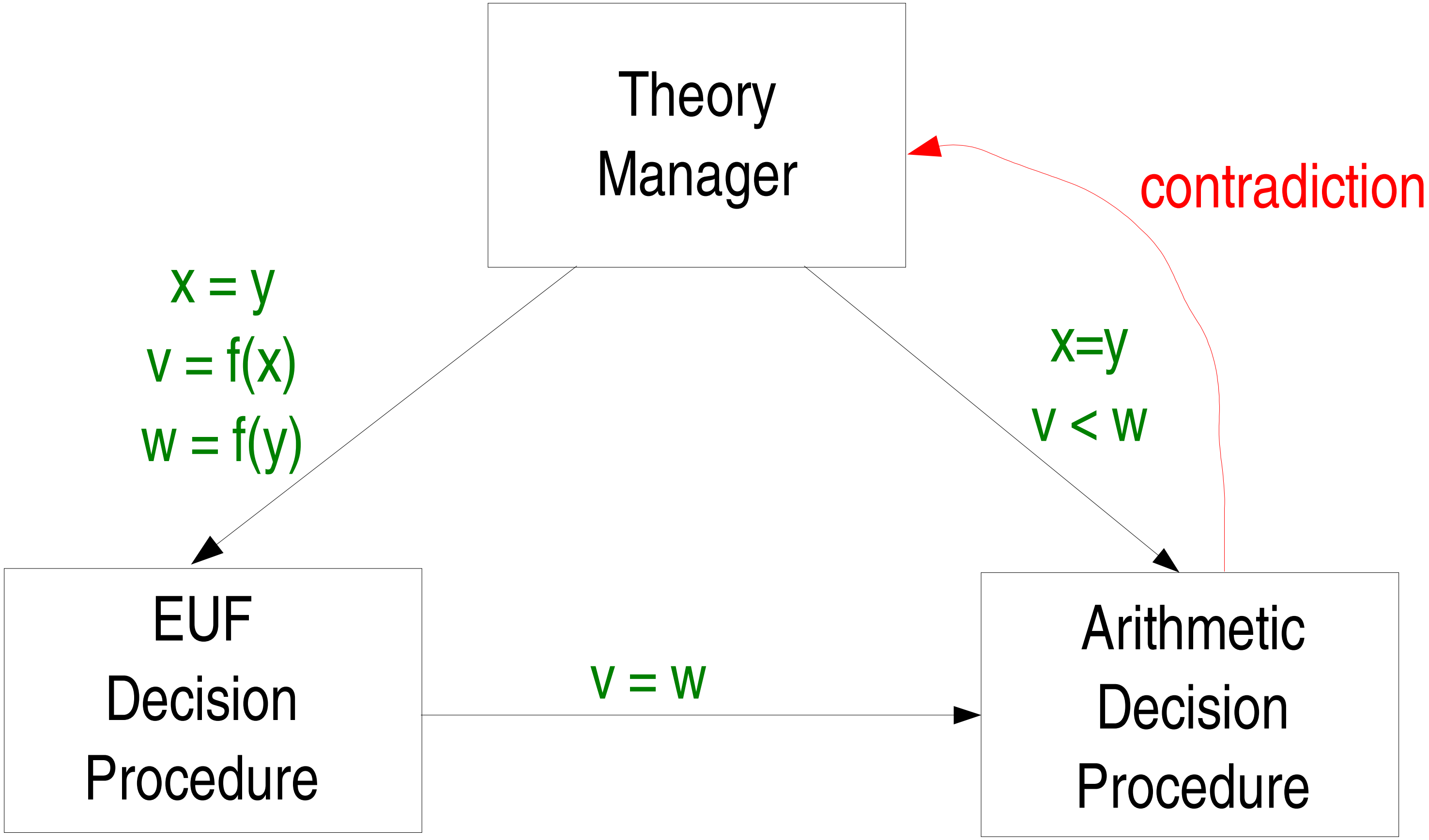
EUF Decision Procedure

Arithmetic Decision Procedure

Checking $(x = y) \wedge (f(x) < f(y))$

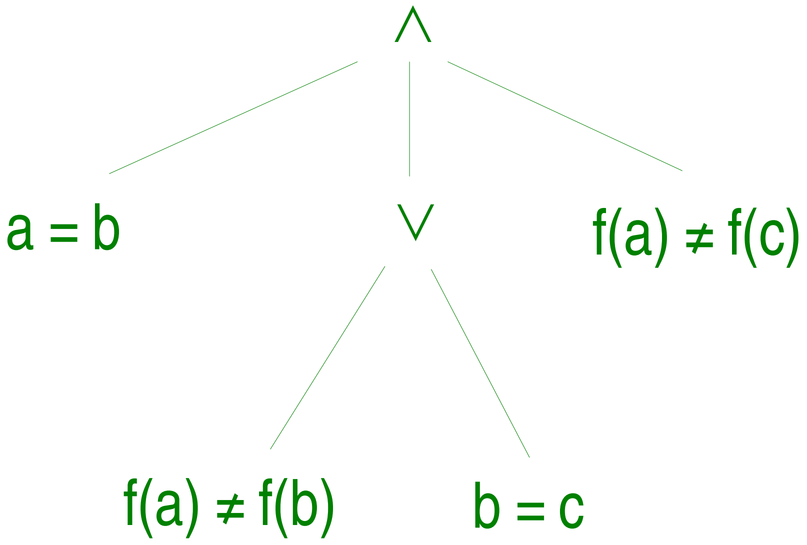


Checking $(x = y) \wedge (f(x) < f(y))$



Backtracking in Nelson-Oppen

- Consider



$\wedge$

a = b

$\vee$

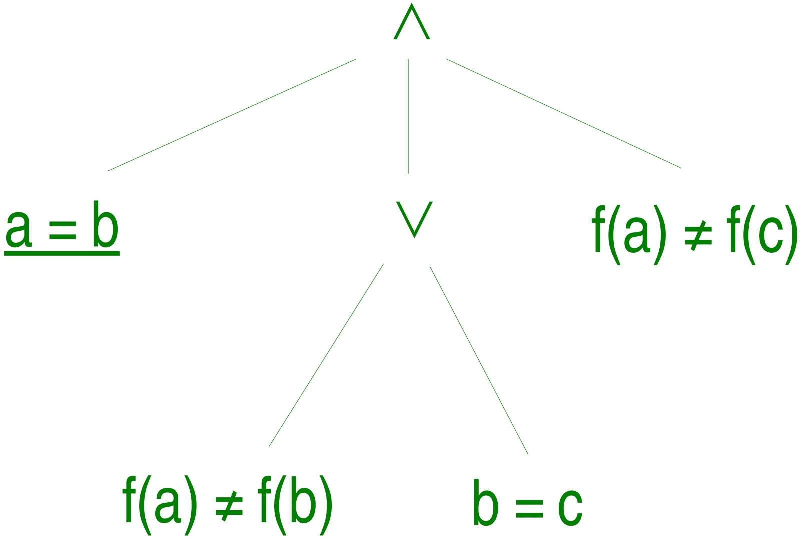
f(a) ≠ f(c)

f(a) ≠ f(b)

b = c

Backtracking in Nelson-Oppen

- Consider



$\wedge$

a = b

$\vee$

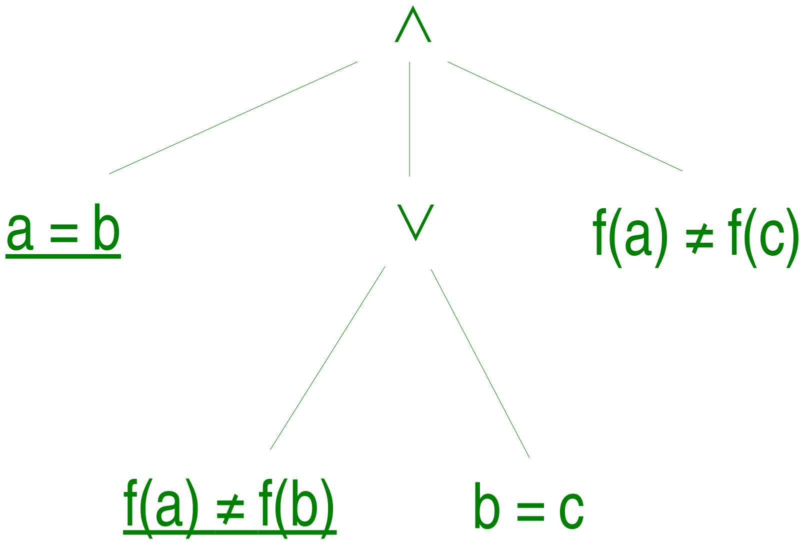
f(a) ≠ f(c)

f(a) ≠ f(b)

b = c

Backtracking in Nelson-Oppen

- Consider



$\wedge$

$a = b$

$\vee$

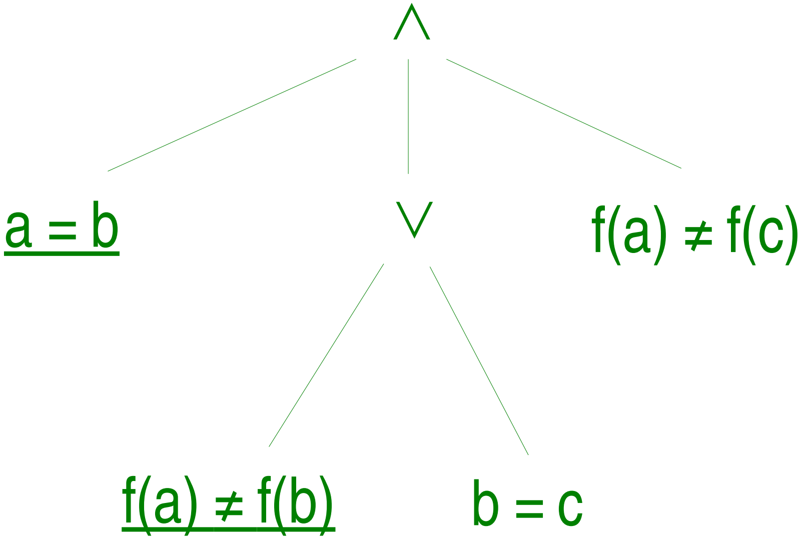
$f(a) \neq f(c)$

$f(a) \neq f(b)$

$b = c$

Backtracking in Nelson-Oppen

- Consider


∧

a = b

∨

f(a) ≠ f(c)

f(a) ≠ f(b)

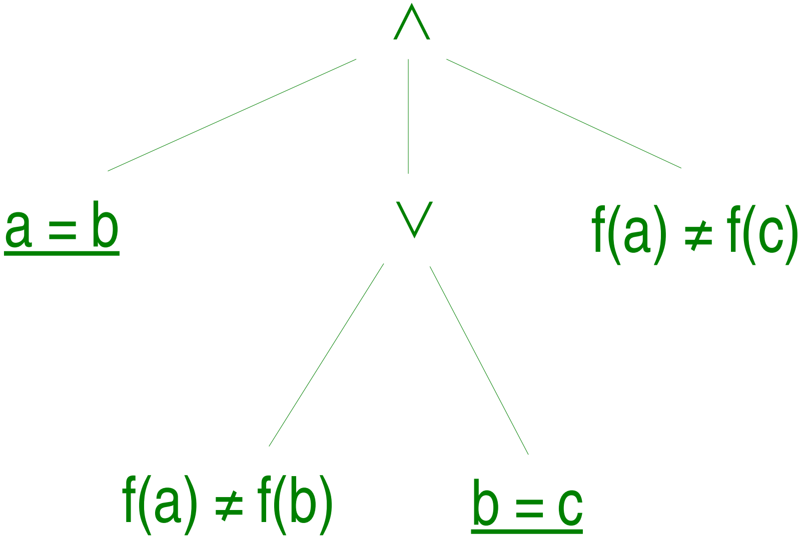
b = c

Inconsistency detected by the EUF procedure.

So backtrack, and try other branch.

Backtracking in Nelson-Oppen

- Consider



∧

a = b

∨

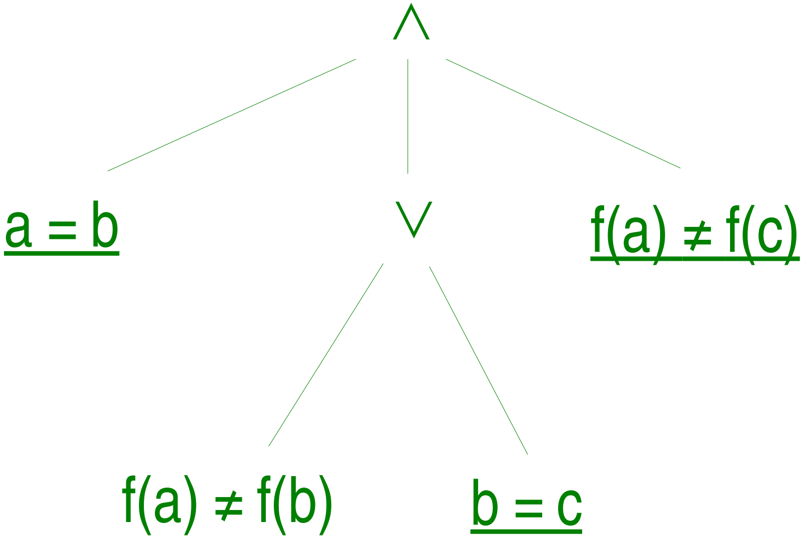
f(a) ≠ f(c)

f(a) ≠ f(b)

b = c

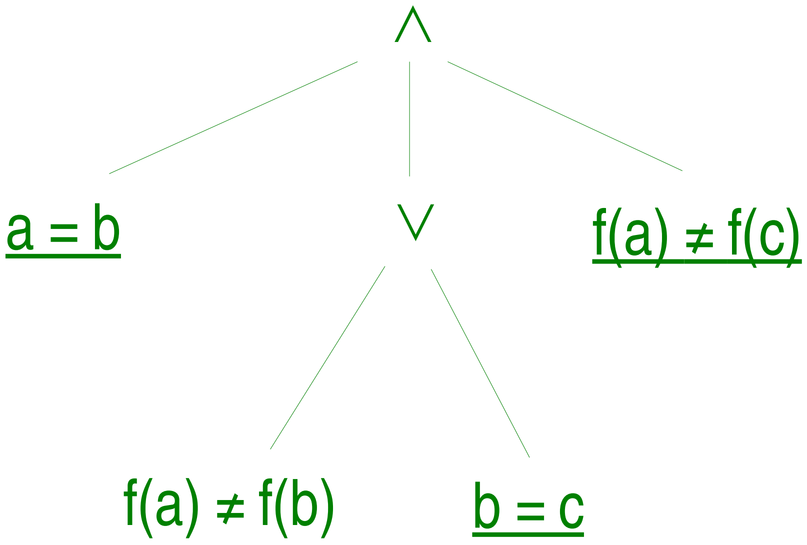
Backtracking in Nelson-Oppen

- Consider



Backtracking in Nelson-Oppen

- Consider



∧

a = b

∨

f(a) ≠ f(c)

f(a) ≠ f(b)

b = c

This assignment is also inconsistent with EUF.

There are no branches left, so the formula is unsatisfiable.

Simplify

- Written by Greg Nelson, Dave Detlefs and Jim Saxe
- Supports
  - EUF (using the E-graph data structure)
  - rational linear arithmetic (using the Simplex algorithm)
  - quantified formulae involving ∃ and ∀ (using matching)
- Very successful: used as the engine in many checkers
  - ESC/Modula-3, ESC/Java, SLAM, ...

Experience with Simplify

- Backtracking search is too slow
  - Far surpassed by recent advances in SAT solving
- Inconsistencies reveal only one bit of information
  - Theory modules repeatedly rediscover the “same” inconsistencies

A Prover using Lazy Proof Explication

- Key ideas
  - use a fast SAT solver to find candidate truth assignments to atomic formulae
  - have theory modules produce compact “proofs” that are added to the SAT problem to reject all truth assignments containing the “same” inconsistency
- Requires
  - proof-explicating theory modules

Example using lazy proof explication

- Suppose we want to check satisfiability of

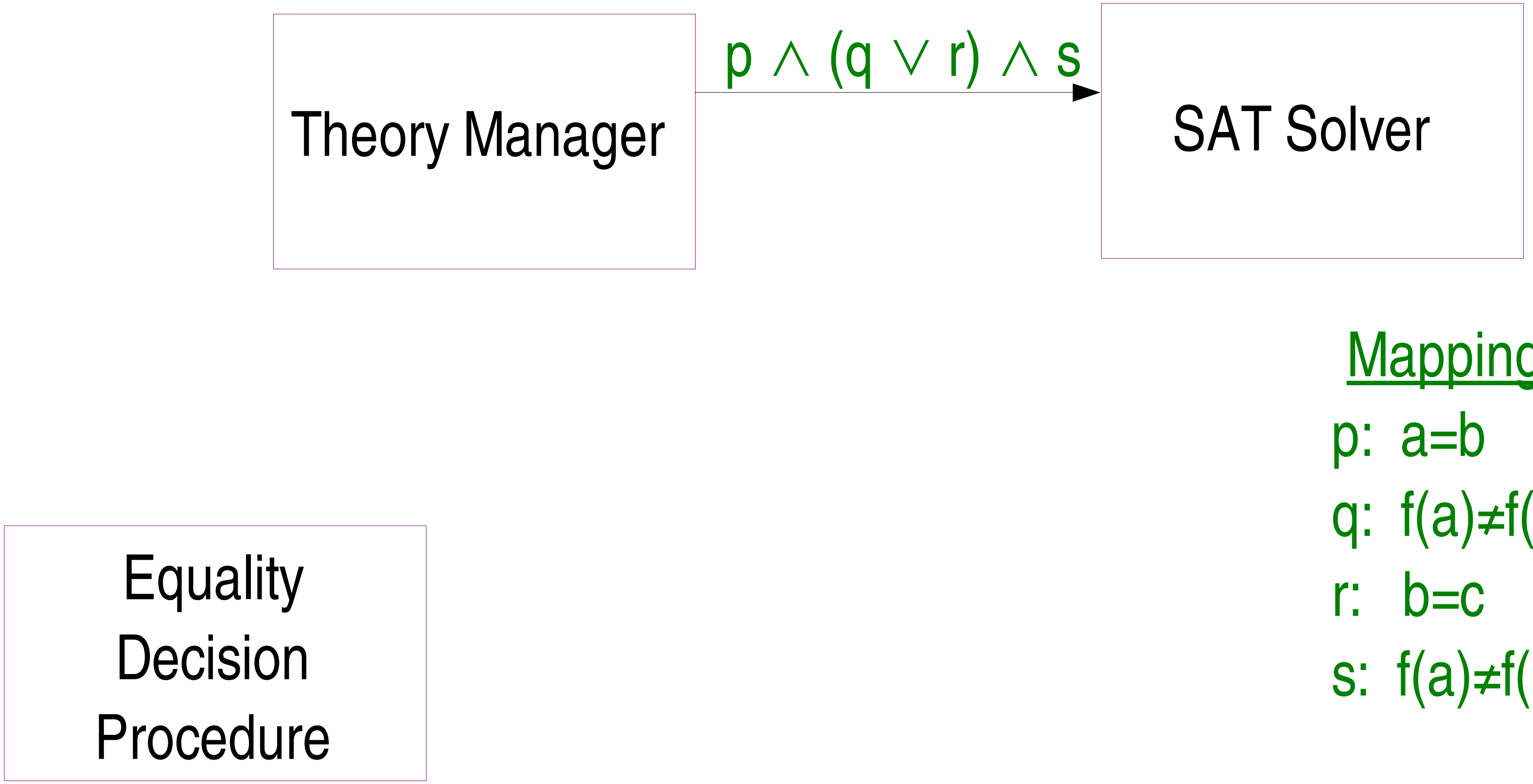
$$(a = b) \;\wedge\; (f(a) \neq f(b) \vee b = c) \;\wedge\; (f(a) \neq f(c))$$

- Encode it in propositional logic

$$p \;\wedge\; (\; q \;\vee\; r \;) \;\wedge\; s$$

where p denotes (a=b), and so on

Example using lazy proof explication



Theory Manager

$p \wedge (q \vee r) \wedge s$

SAT Solver

Equality Decision Procedure

<u>Mapping</u>

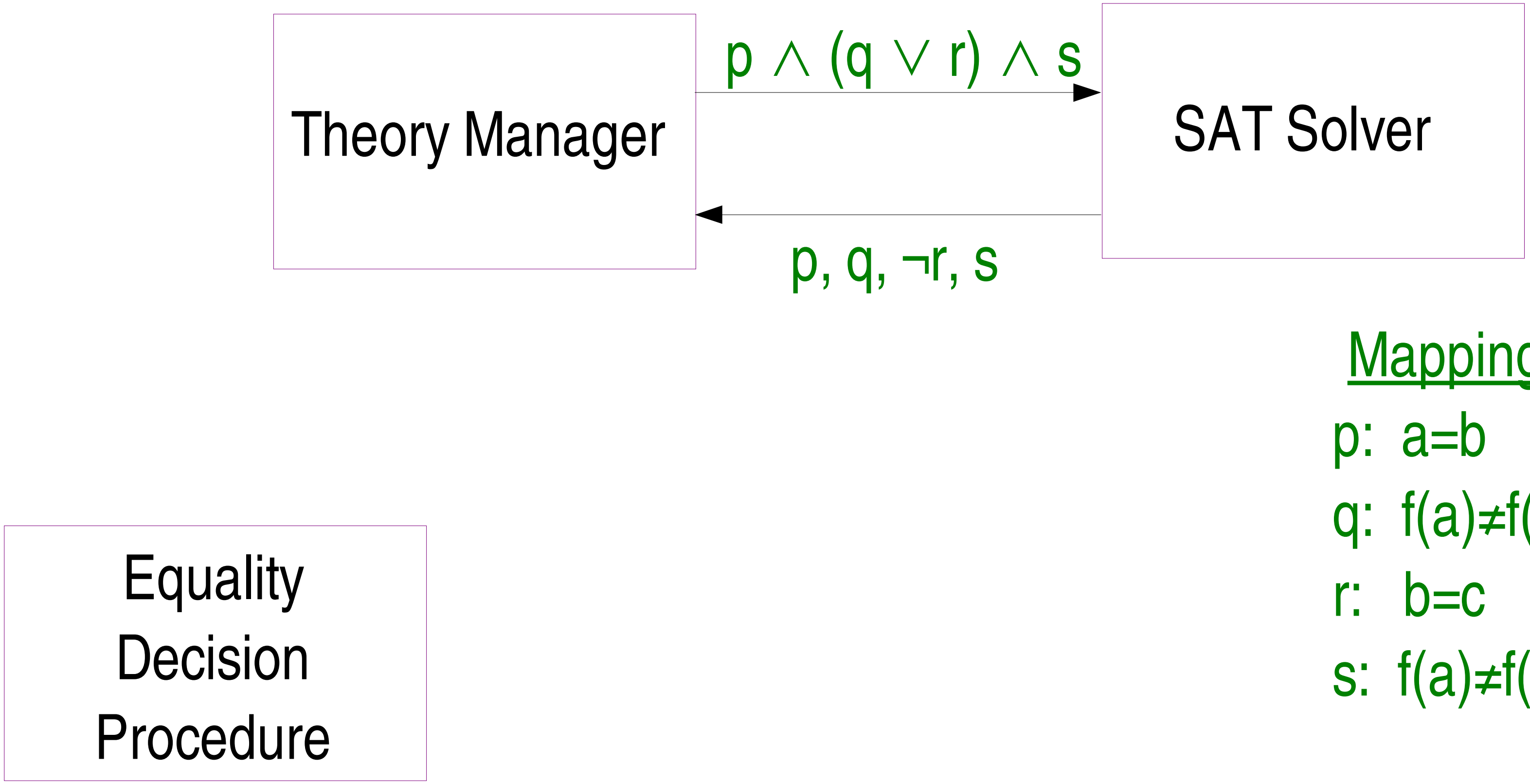
p: $a=b$

q: $f(a) \neq f(b)$

r: $b=c$

s: $f(a) \neq f(c)$

Example using lazy proof explication



Theory Manager

$p \wedge (q \vee r) \wedge s$

SAT Solver

p, q, ¬r, s

Equality Decision Procedure

<u>Mapping</u>

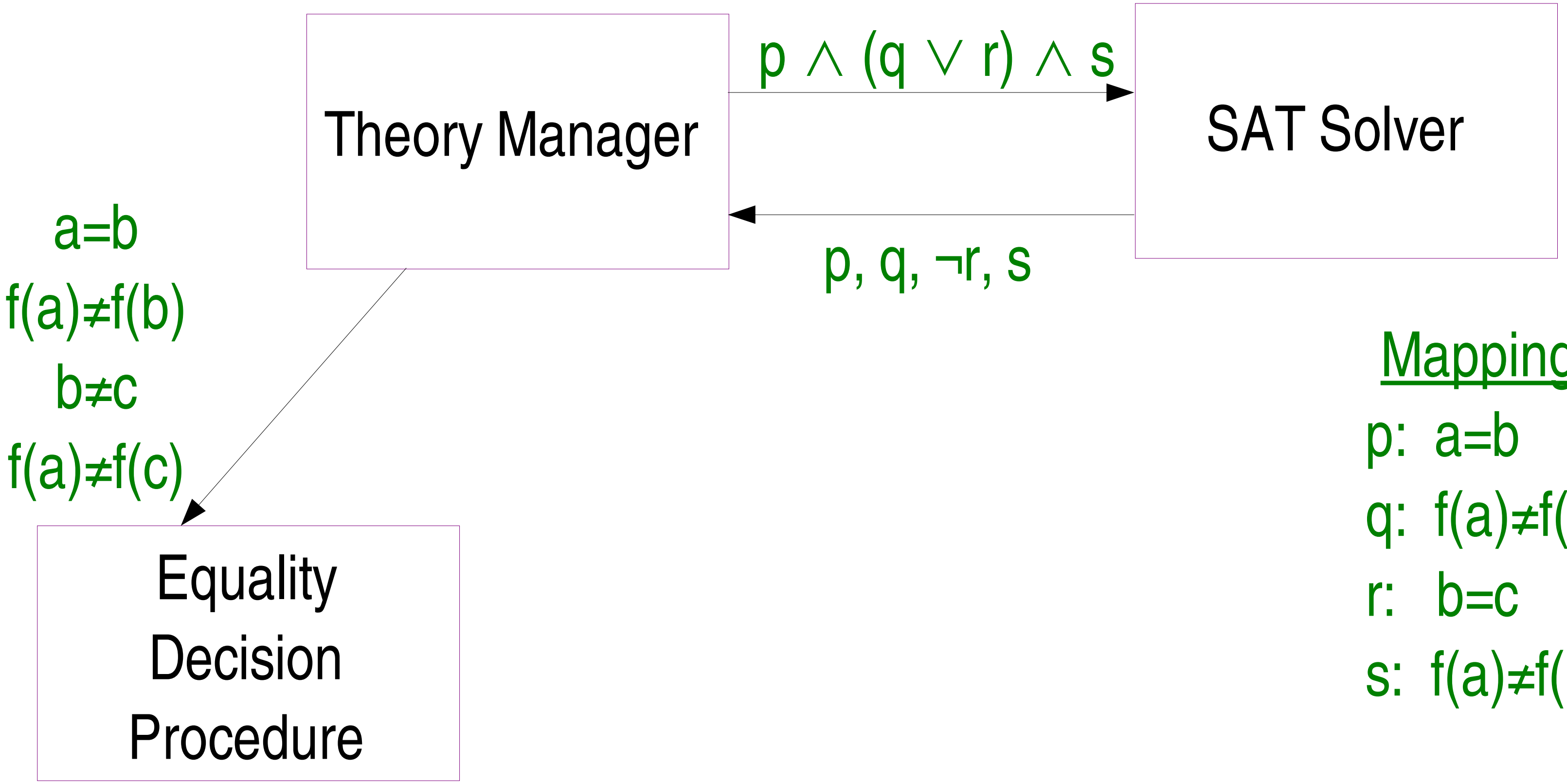
p: a=b

q: f(a)≠f(b)

r: b=c

s: f(a)≠f(c)

Example using lazy proof explication



Theory Manager

$p \wedge (q \vee r) \wedge s$

SAT Solver

$p, q, \neg r, s$

$a=b$

$f(a) \neq f(b)$

$b \neq c$

$f(a) \neq f(c)$

Equality Decision Procedure

Mapping

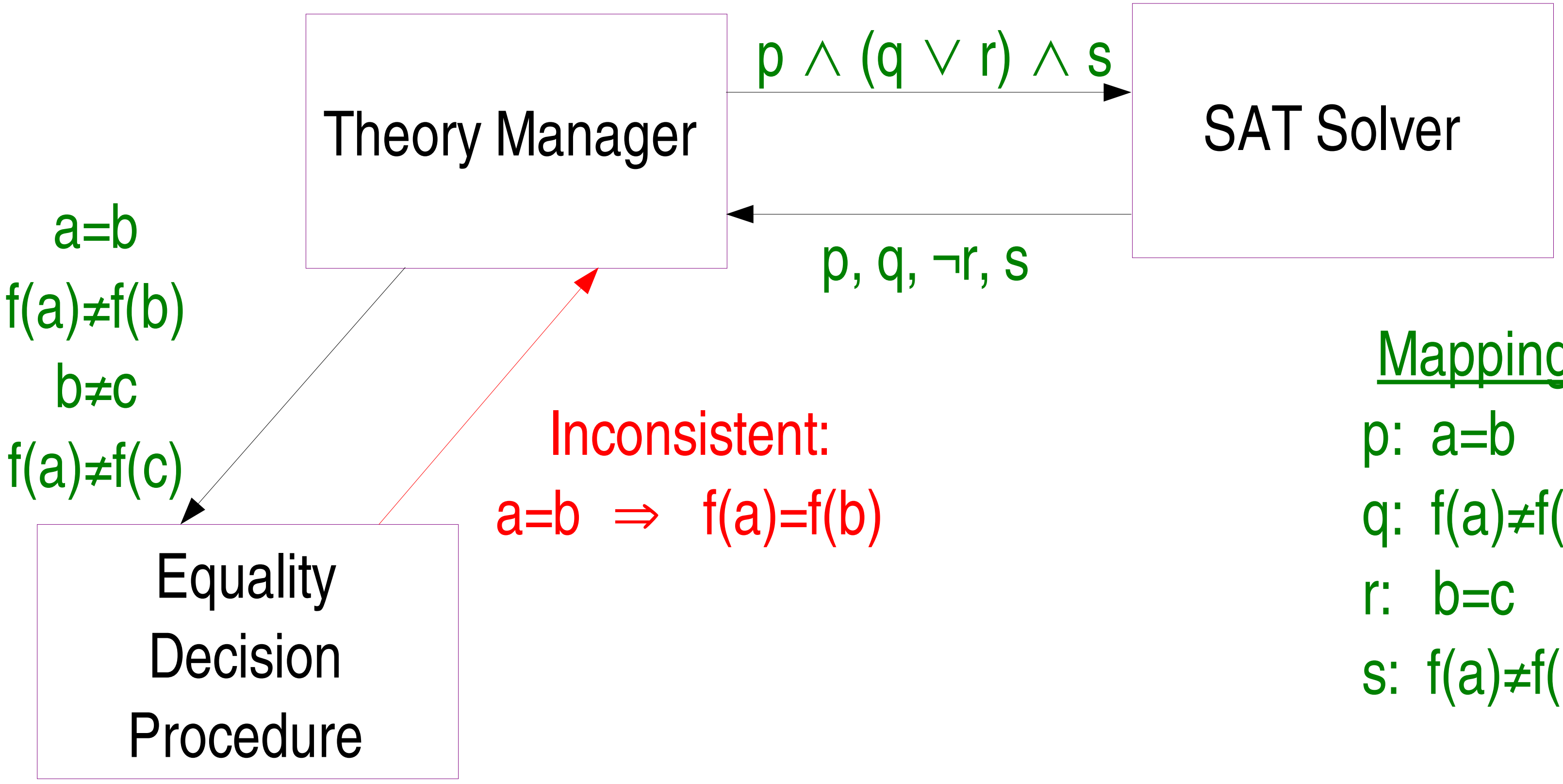
p: $a=b$

q: $f(a) \neq f(b)$

r: $b=c$

s: $f(a) \neq f(c)$

Example using lazy proof explication



Theory Manager

$p \wedge (q \vee r) \wedge s$

SAT Solver

p, q, ¬r, s

$a=b$
$f(a) \neq f(b)$
$b \neq c$
$f(a) \neq f(c)$

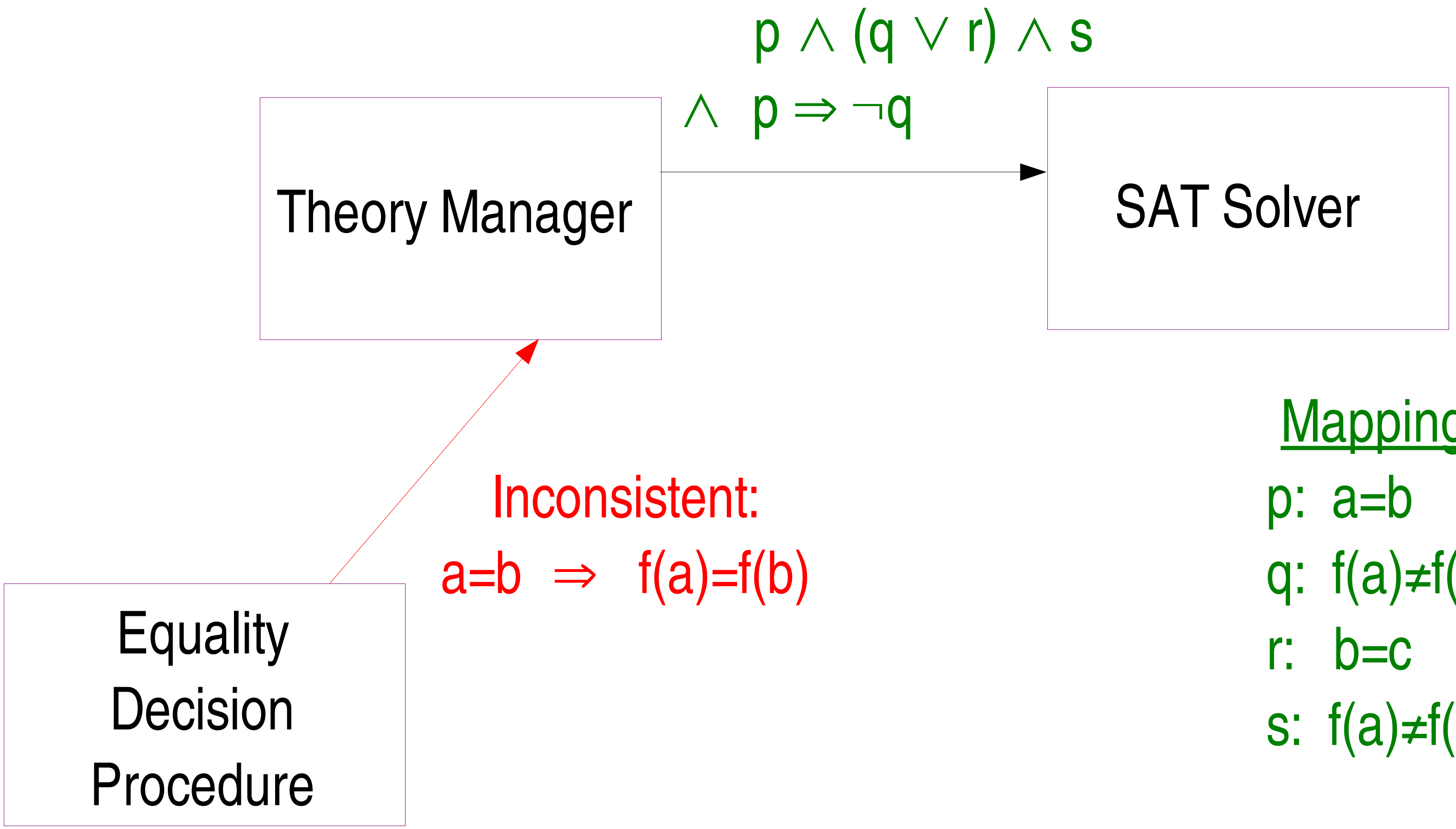
Equality Decision Procedure

Inconsistent:
$a=b \;\Rightarrow\; f(a)=f(b)$

Mapping

p: $a=b$
q: $f(a) \neq f(b)$
r: $b=c$
s: $f(a) \neq f(c)$

Example using lazy proof explication



$p \wedge (q \vee r) \wedge s$

$\wedge \ \ p \Rightarrow \neg q$

Theory Manager

SAT Solver

Inconsistent:

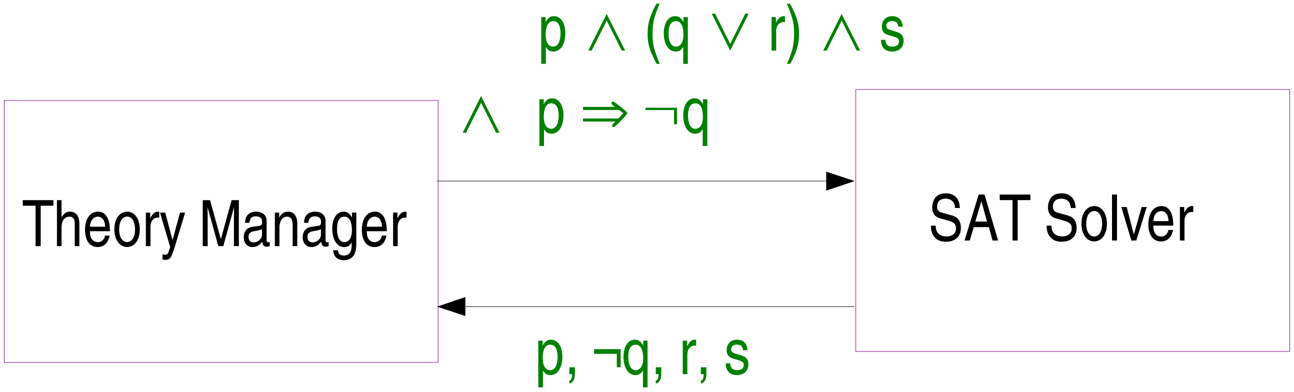
$a=b \ \Rightarrow \ f(a)=f(b)$

Equality Decision Procedure

<u>Mapping</u>

p: $a=b$
q: $f(a) \neq f(b)$
r: $b=c$
s: $f(a) \neq f(c)$

Example using lazy proof explication



$p \wedge (q \vee r) \wedge s$

$\wedge\ p \Rightarrow \neg q$

Theory Manager

SAT Solver

$p, \neg q, r, s$

Equality
Decision
Procedure

<u>Mapping</u>

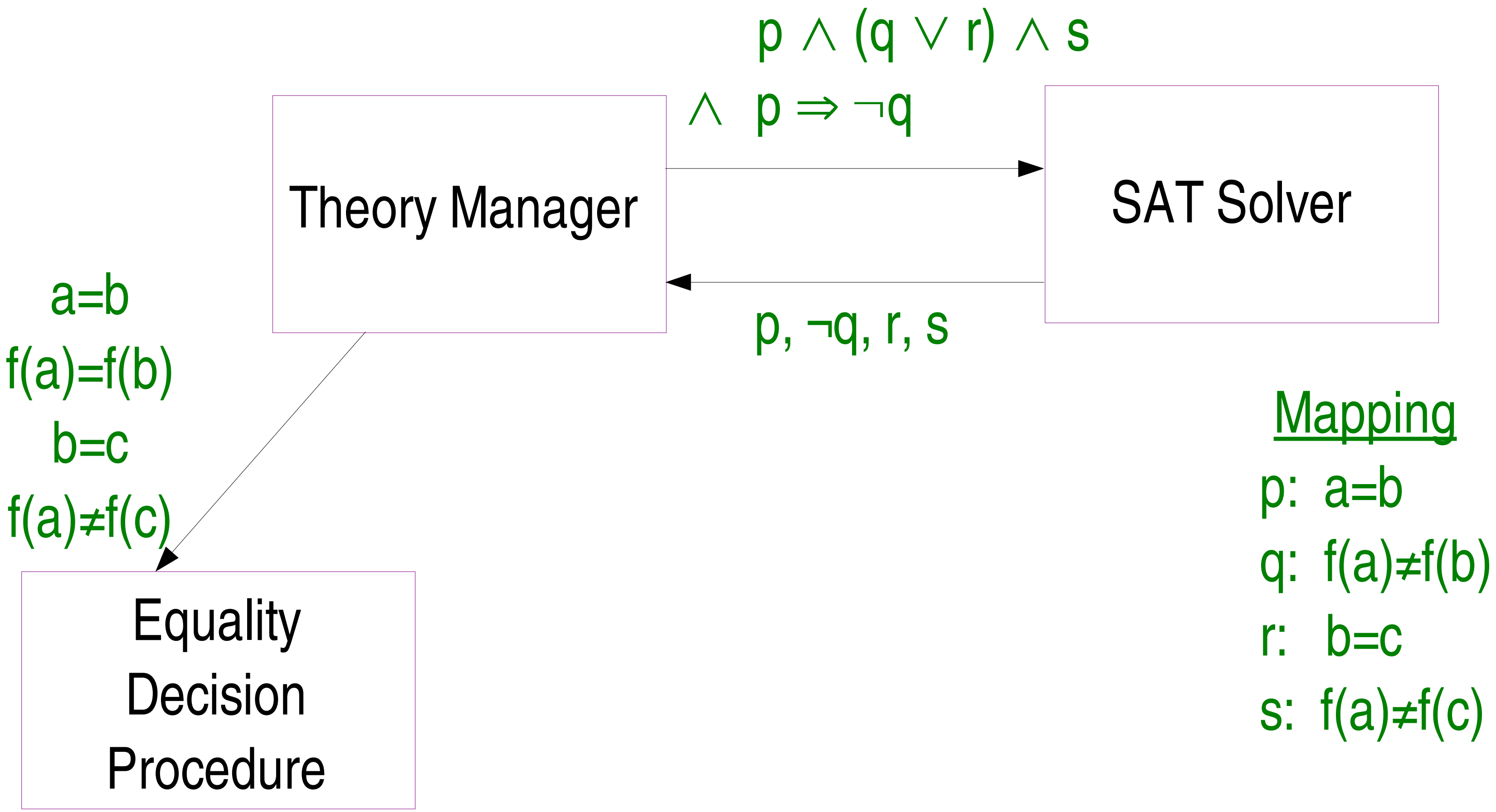
p: $a=b$

q: $f(a) \neq f(b)$

r: $b=c$

s: $f(a) \neq f(c)$

Example using lazy proof explication



$p \wedge (q \vee r) \wedge s$

$\wedge\ \ p \Rightarrow \neg q$

Theory Manager

SAT Solver

$p, \neg q, r, s$

$a=b$

$f(a)=f(b)$

$b=c$

$f(a)\neq f(c)$

Equality Decision Procedure

<u>Mapping</u>

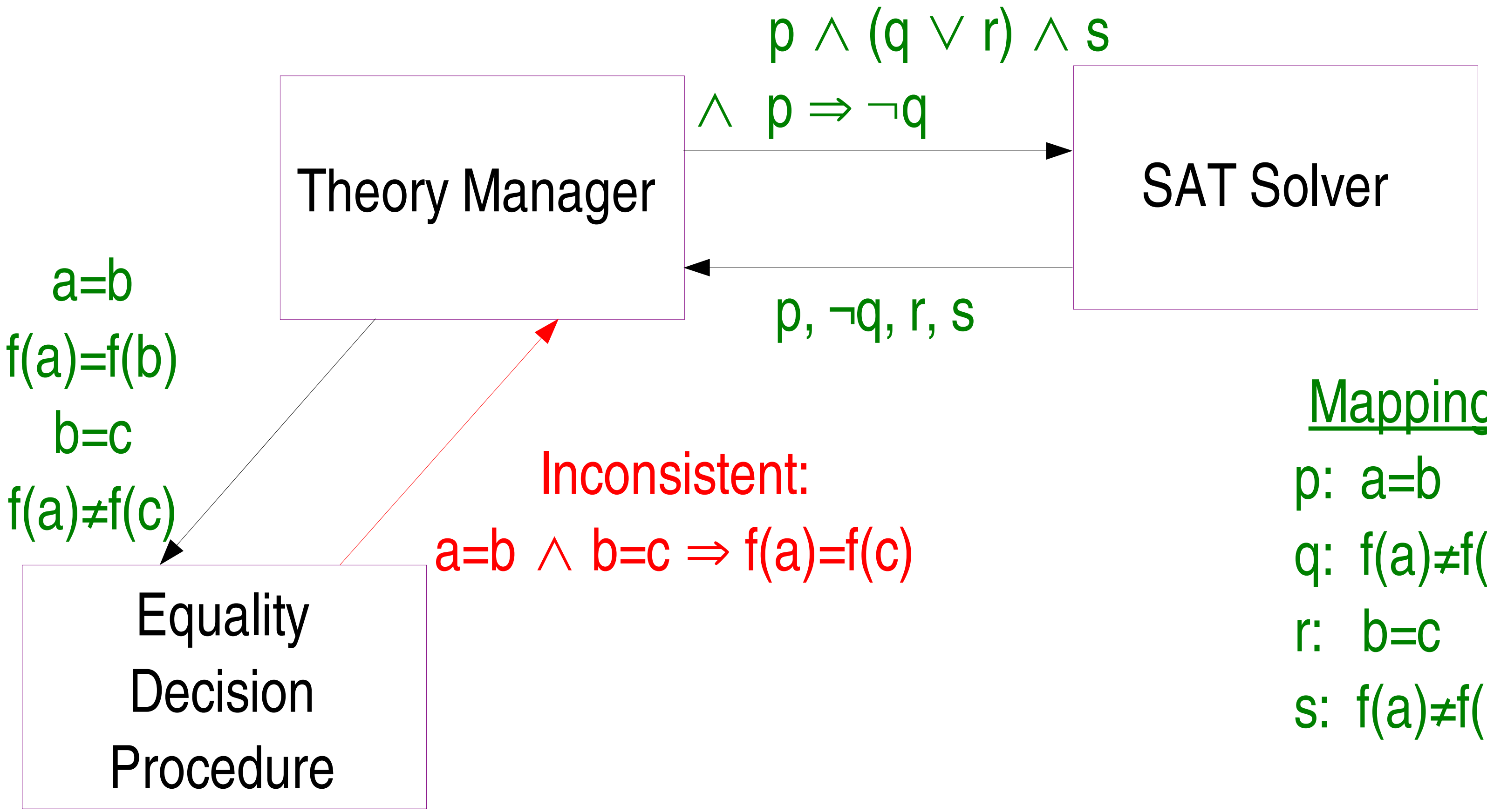
p: $a=b$

q: $f(a)\neq f(b)$

r: $b=c$

s: $f(a)\neq f(c)$

Example using lazy proof explication



$p \wedge (q \vee r) \wedge s$

$\wedge\ \ p \Rightarrow \neg q$

Theory Manager

SAT Solver

$p, \neg q, r, s$

$a=b$

$f(a)=f(b)$

$b=c$

$f(a) \neq f(c)$

Inconsistent:

$a=b \wedge b=c \Rightarrow f(a)=f(c)$

Equality Decision Procedure

Mapping

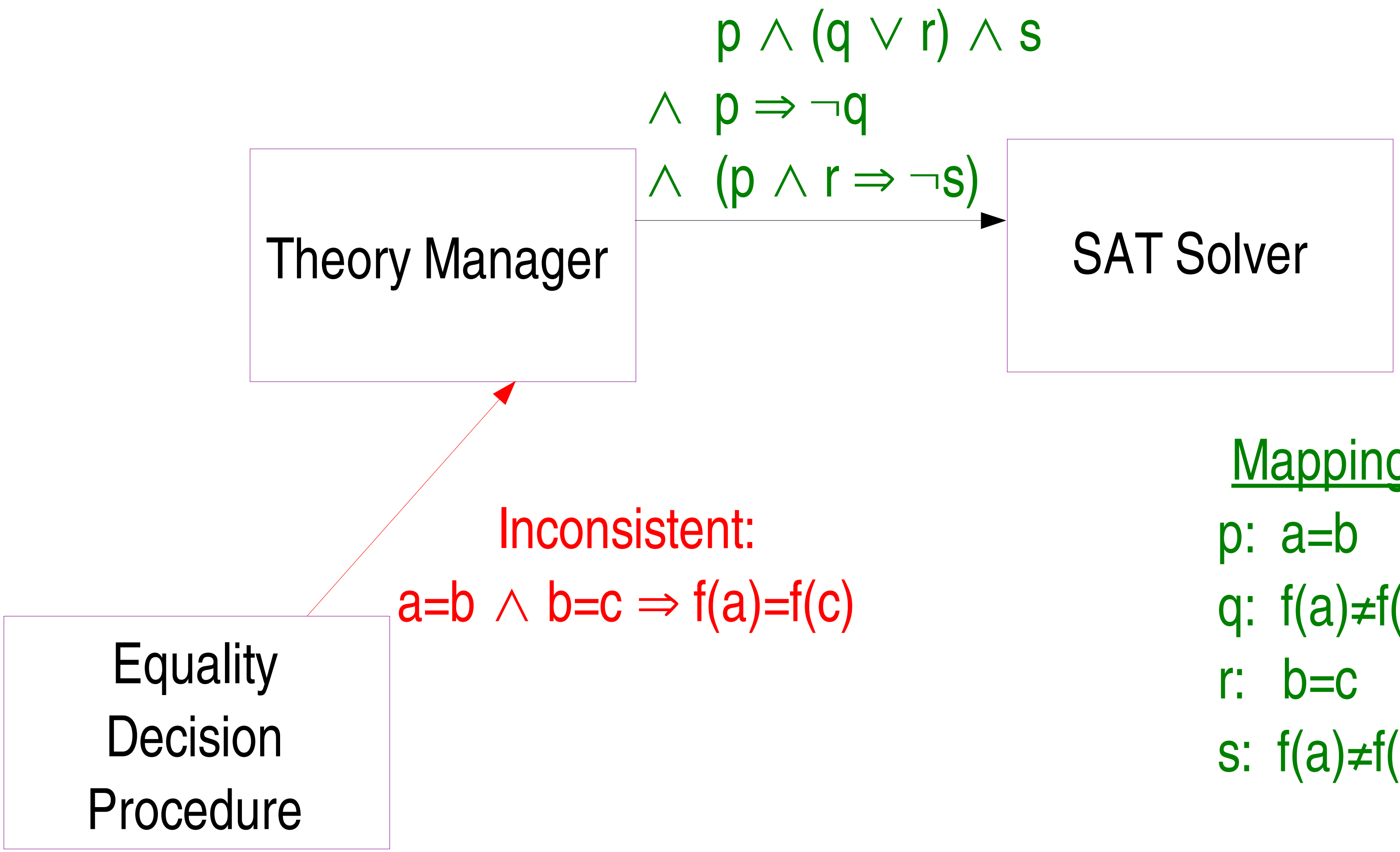
p: $a=b$

q: $f(a) \neq f(b)$

r: $b=c$

s: $f(a) \neq f(c)$

Example using lazy proof explication



$p \wedge (q \vee r) \wedge s$

$\wedge \;\; p \Rightarrow \neg q$

$\wedge \;\; (p \wedge r \Rightarrow \neg s)$

Theory Manager

SAT Solver

Inconsistent:

$a=b \wedge b=c \Rightarrow f(a)=f(c)$

Equality Decision Procedure

Mapping

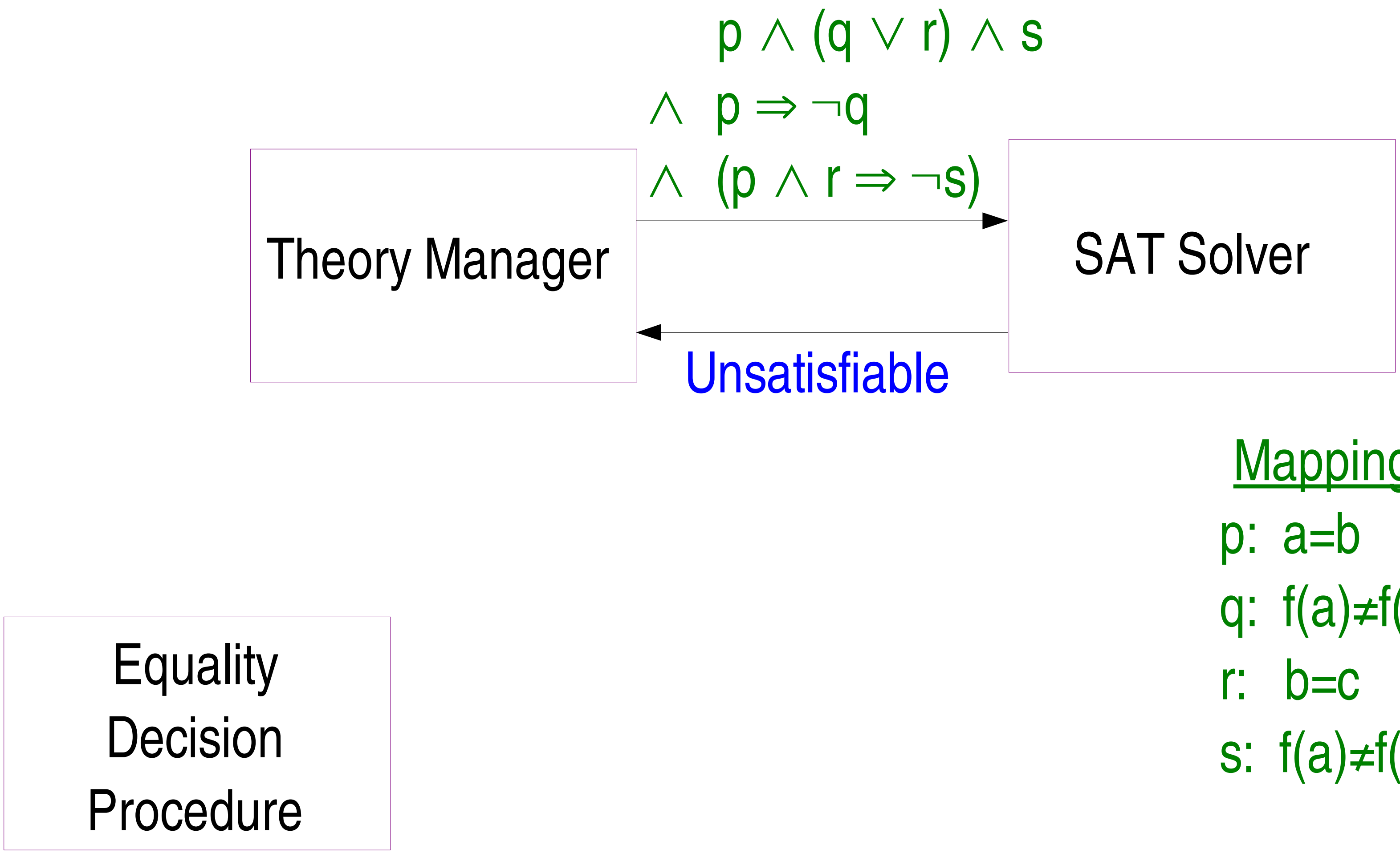
p: $a=b$

q: $f(a) \neq f(b)$

r: $b=c$

s: $f(a) \neq f(c)$

Example using lazy proof explication



$p \wedge (q \vee r) \wedge s$

$\wedge \; p \Rightarrow \neg q$

$\wedge \; (p \wedge r \Rightarrow \neg s)$

<u>Mapping</u>

p: $a=b$

q: $f(a) \neq f(b)$

r: $b=c$

s: $f(a) \neq f(c)$

Definitions

- A literal is an atomic formula or its negation, e.g, $(a<b)$
- A quantified formula is either a $\forall$-formula or its negation e.g., $\neg\forall y.F$ where F is a formula (we also write this as $\exists y.\neg F$)
- A formula is an arbitrary boolean combination of atomic formulae and quantified formulae, e.g, $(b > 0 \Rightarrow \forall x.(P(x) \vee \exists y.\neg Q(x,y)))$
- A monome is a set of literals and quantified formulae, e.g., $\{ b > 0, \neg Q(a,b), \forall x.(P(x) \vee \exists y.\neg Q(x,y)) \}$

Two key procedures

- *satisfyProp(F)*
  - returns either UNSAT, or
  - a monome *m* representing a satisfying boolean assignment to the atomic formulae and outermost quantified formulae in *F*
- *satisfyTheories(m)*
  - returns either SAT, or
  - a formula *F* such that

    F is a tautology wrt the underlying theories, and

    $(F \wedge m)$ is **propositionally** unsatisfiable

Algorithm for quantifier-free formulae

```
satisfy(F) {   /* returns UNSAT or a monome satisfying F */
    E := true
    while (true) {
        m := satisfyProp(F ∧ E)
        if (m = UNSAT)  { return UNSAT }
        else {
            R := satisfyTheories(m)
            if (R = SAT) { return m }
            else { E := E ∧ R }
        }
    }
```

Algorithm for formulae with quantifiers

```
• satisfy(F) {   /* returns UNSAT or a monome satisfying F */
    E := true
    while (true) {
        m := satisfyProp(F ∧ E)
        if (m = UNSAT)  { return UNSAT }
        else {
            R := checkMonome(m)
            if (R = SAT) { return m }
            else { E := E ∧ R }
        }
    }
```

Procedure *checkMonome(..)*

```
checkMonome(m) {  /* returns SAT or an explicated proof */
    R := satisfyTheories(m)
    if  (R $\neq$ SAT) {  return R }
    if  m contains  $\exists x.F(x)$
        such that  $(m \wedge \neg F(V_F))$  is propositionally satisfiable
            { return  $(\exists x.F(x)) \Rightarrow F(V_F)$ }
    if  m  contains  $\forall x.F(x)$  such that for some substitution $\sigma$,
        $(m \wedge \neg\sigma(F))$  is propositionally satisfiable
            { return  $(\forall x.F(x)) \Rightarrow \sigma(F)$ }
    return SAT
}
```

where V_F is a fresh, unique variable for given formula F

Quantified formula example

- Suppose we want to check satisfiability of

$$
\begin{aligned}
& b \geq 1 \\
\wedge\ & b > 0 \ \Rightarrow\ \forall x.(P(x) \vee \exists y.\neg Q(x,y)) \\
\wedge\ & \neg P(a) \\
\wedge\ & \forall z.Q(a,z)
\end{aligned}
$$

Quantified formula example

- Suppose that the SAT solver assigns true to the green atomic formulae, and false to the red atomic formulae

$$
\begin{aligned}
& b \geq 1 \\
\wedge\ & b > 0 \Rightarrow \forall x.(P(x) \vee \exists y.\neg Q(x,y)) \\
\wedge\ & \neg P(a) \\
\wedge\ & \forall z.Q(a,z)
\end{aligned}
$$

But this is inconsistent with arithmetic

Suppose satisfyTheories(..) explicates the proof

$$(b \geq 1 \Rightarrow b > 0)$$

Quantified formula example

- We add the explicated proof to the original problem, and invoke the SAT solver again. It assigns true to all atomic formulae:

  $$
  \begin{aligned}
  & b \geq 1 \\
  \wedge\ & b > 0 \Rightarrow \forall x.\big(P(x) \vee \exists y.\neg Q(x,y)\big) \\
  \wedge\ & \neg P(a) \\
  \wedge\ & \forall z.Q(a,z) \\
  \wedge\ & (b \geq 1 \Rightarrow b > 0)
  \end{aligned}
  $$

  The theories do not detect any inconsistency, and there is no existentially quantified formula, so we invoke the matcher. Suppose the matcher produces the instance $x := a$

Quantified formula example

- We add the new instance to the problem as a tautology:

$$
\begin{aligned}
& b \geq 1 \\
\wedge\ & b > 0 \ \Rightarrow\ \forall x.(P(x) \vee \exists y.\neg Q(x,y)) \\
\wedge\ & \neg P(a) \\
\wedge\ & \forall z.Q(a,z) \\
\wedge\ & (b \geq 1 \ \Rightarrow\ b > 0) \\
\wedge\ & \forall x.(P(x) \vee \exists y.\neg Q(x,y)) \ \Rightarrow\ P(a) \vee \exists y.\neg Q(a,y)
\end{aligned}
$$

Quantified formula example

- Invoking the SAT solver now yields the following assignment

$$b \geq 1$$
$$\land\ b > 0 \ \Rightarrow\ \forall x.(P(x) \lor \exists y.\neg Q(x,y))$$
$$\land\ \neg P(a)$$
$$\land\ \forall z.Q(a,z)$$
$$\land\ (b \geq 1 \ \Rightarrow\ b > 0)$$
$$\land\ \forall x.(P(x) \lor \exists y.\neg Q(x,y)) \ \Rightarrow\ P(a) \ \lor\ \exists y.\neg Q(a,y)$$

The theories detect no inconsistency, so we assert $\exists y.\neg Q(a,y)$

This leads to creation of a skolem constant v_0 and explication of

$$\exists y.\neg Q(a,y) \ \Rightarrow\ \neg Q(a,v_0)$$

Quantified formula example

- We add the explicated proof

$$
\begin{aligned}
& b \geq 1 \\
\wedge\ & b > 0 \ \Rightarrow\ \forall x.\big(P(x) \vee \exists y.\neg Q(x,y)\big) \\
\wedge\ & \neg P(a) \\
\wedge\ & \forall z.Q(a,z) \\
\wedge\ & (b \geq 1 \ \Rightarrow\ b > 0) \\
\wedge\ & \forall x.\big(P(x) \vee \neg\forall y.Q(x,y)\big) \ \Rightarrow\ P(a) \vee \exists y.\neg Q(a,y) \\
\wedge\ & \exists y.\neg Q(a,y) \ \Rightarrow\ \neg Q(a,v_0)
\end{aligned}
$$

Quantified formula example

- Invoking the SAT solver now yields the following assignment

$$
\begin{aligned}
& b \geq 1 \\
\wedge\ & b > 0 \ \Rightarrow\ \forall x.(P(x) \vee \exists y.\neg Q(x,y)) \\
\wedge\ & \neg P(a) \\
\wedge\ & \forall z.Q(a,z) \\
\wedge\ & (b \geq 1 \ \Rightarrow\ b > 0) \\
\wedge\ & \forall x.(P(x) \vee \exists y.\neg Q(x,y)) \ \Rightarrow\ P(a) \vee \exists y.\neg Q(a,y) \\
\wedge\ & \exists y.\neg Q(a,y) \ \Rightarrow\ \neg Q(a,v_0)
\end{aligned}
$$

This is also consistent with the theories, so we invoke the matcher, which instantiates $\forall z.Q(a,z)$ with $z := v_0$

Quantified formula example

- This results in the following formula

$$
\begin{aligned}
& b \geq 1 \\
\wedge\ & b > 0 \ \Rightarrow\ \forall x.(P(x) \vee \exists y.\neg Q(x,y)) \\
\wedge\ & \neg P(a) \\
\wedge\ & \forall z.Q(a,z) \\
\wedge\ & (b \geq 1 \ \Rightarrow\ b > 0) \\
\wedge\ & \forall x.(P(x) \vee \exists y.\neg Q(x,y)) \ \Rightarrow\ P(a) \vee \exists y.\neg Q(a,y) \\
\wedge\ & \exists y.\neg Q(a,y) \ \Rightarrow\ \neg Q(a,v_0) \\
\wedge\ & \forall z.Q(a,z) \ \Rightarrow\ Q(a,v_0)
\end{aligned}
$$

which is propositionally unsatisfiable

Verifun

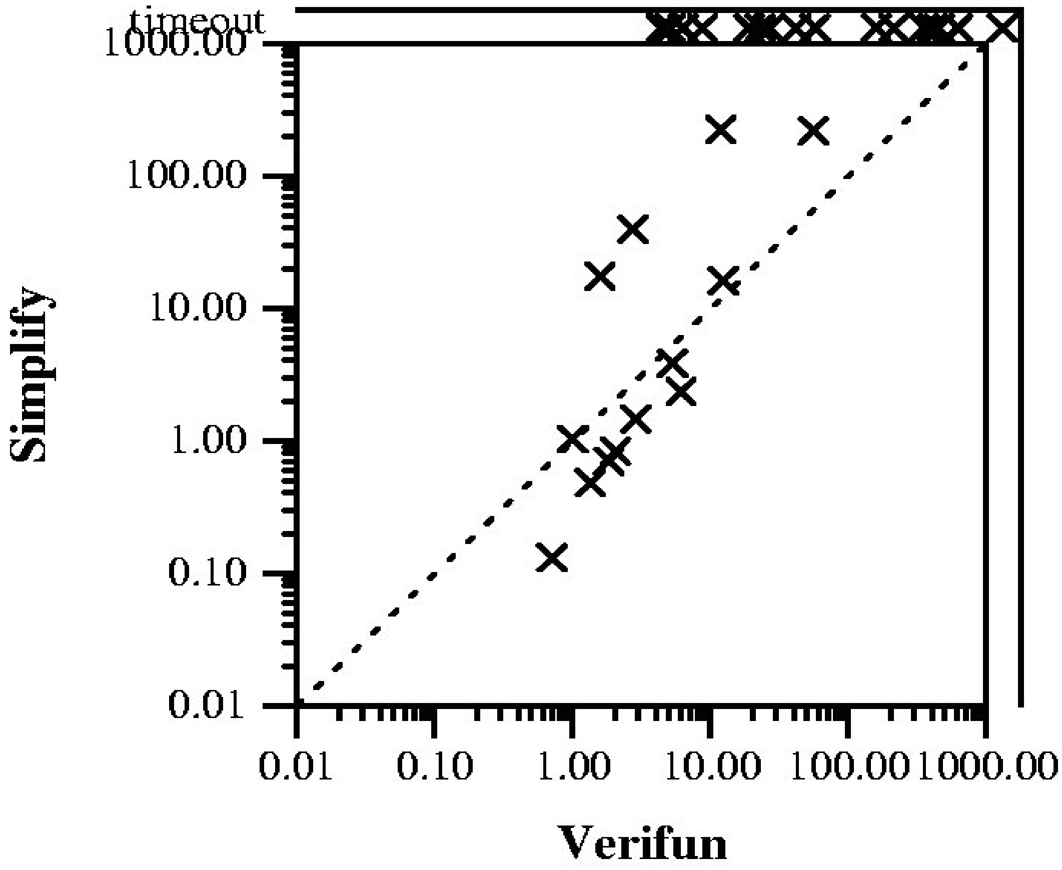
- Intended to be a replacement for Simplify
- Written in Java (~10,500 lines) and in C (~800 lines)
- Supports
  - equality with uninterpreted function symbols (implemented using the E-graph data structure)
  - rational linear arithmetic (based on Nelson's adaptation of the Simplex algorithm; extended with proof-generation by summer intern Xinming Ou, Princeton)
  - quantifiers (based on *matching* upto equivalence)

Verifun performance

- Benchmark suite:
  - 38 processor & cache verification problems (provided by the UCLID group at CMU)
  - 41 timed automata verification problems in the *postoffice* suite (provided by the Math-SAT designers)
- None of the benchmarks included quantified formulae

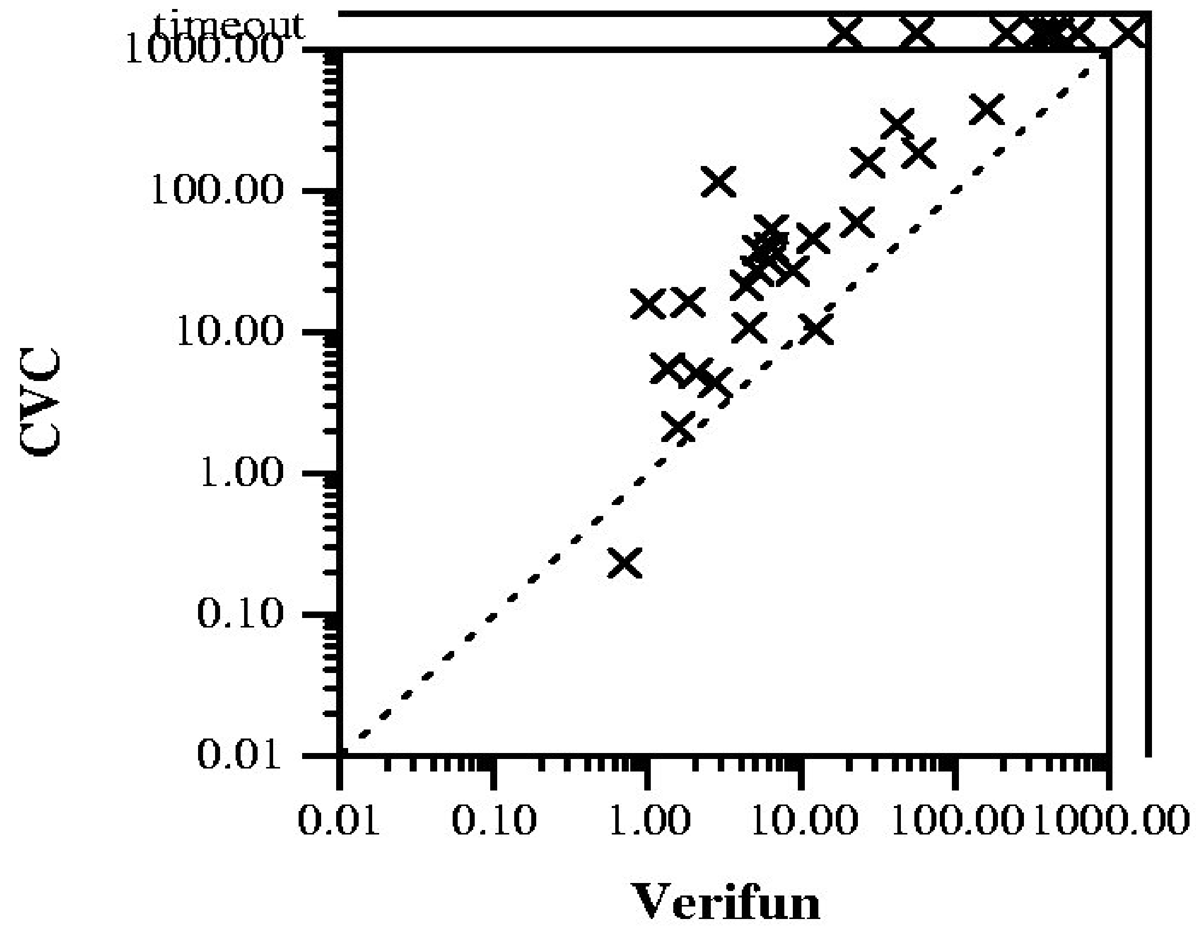
Verifun vs. Simplify

on the UCLID benchmarks



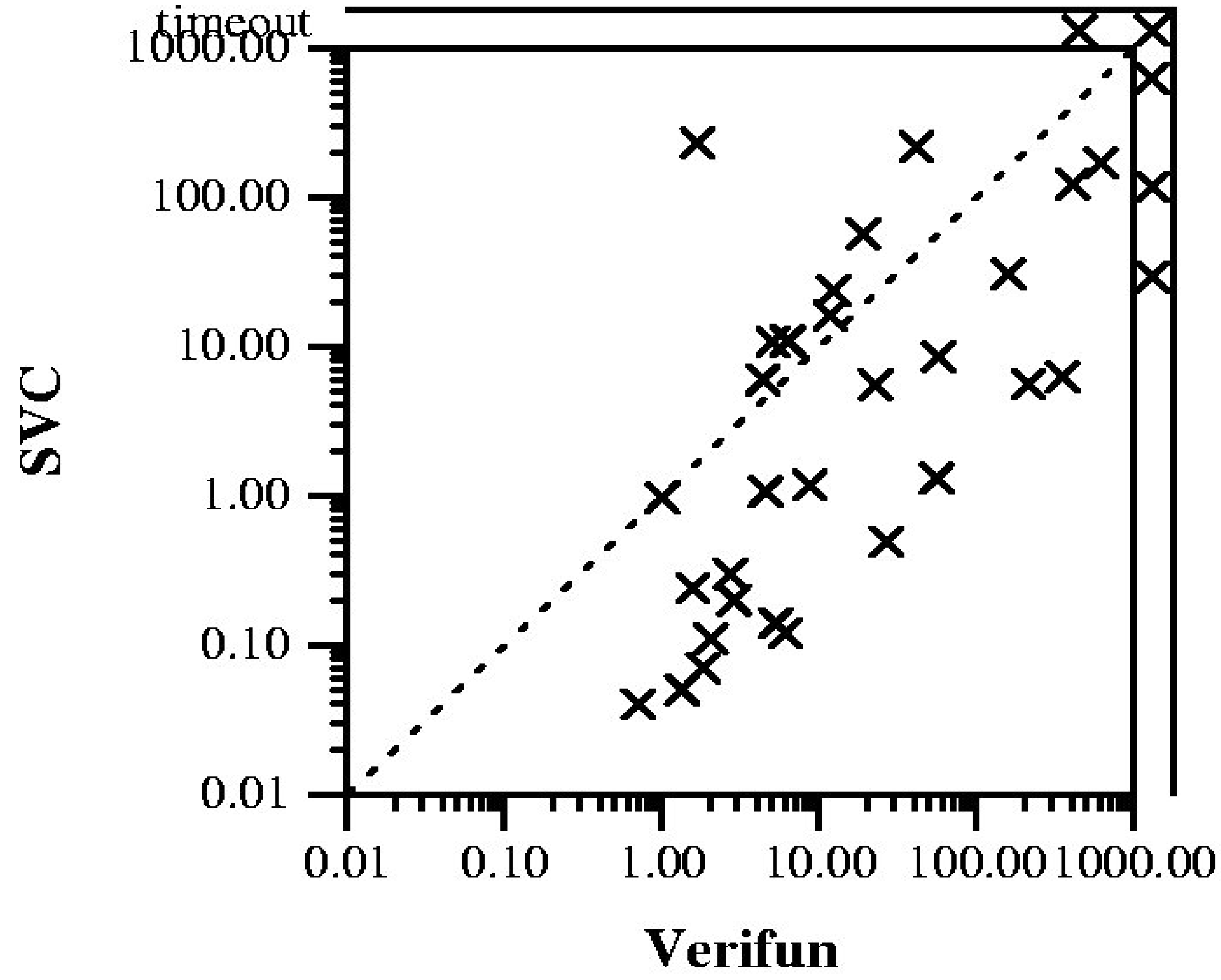
Verifun vs. CVC

on the UCLID benchmarks



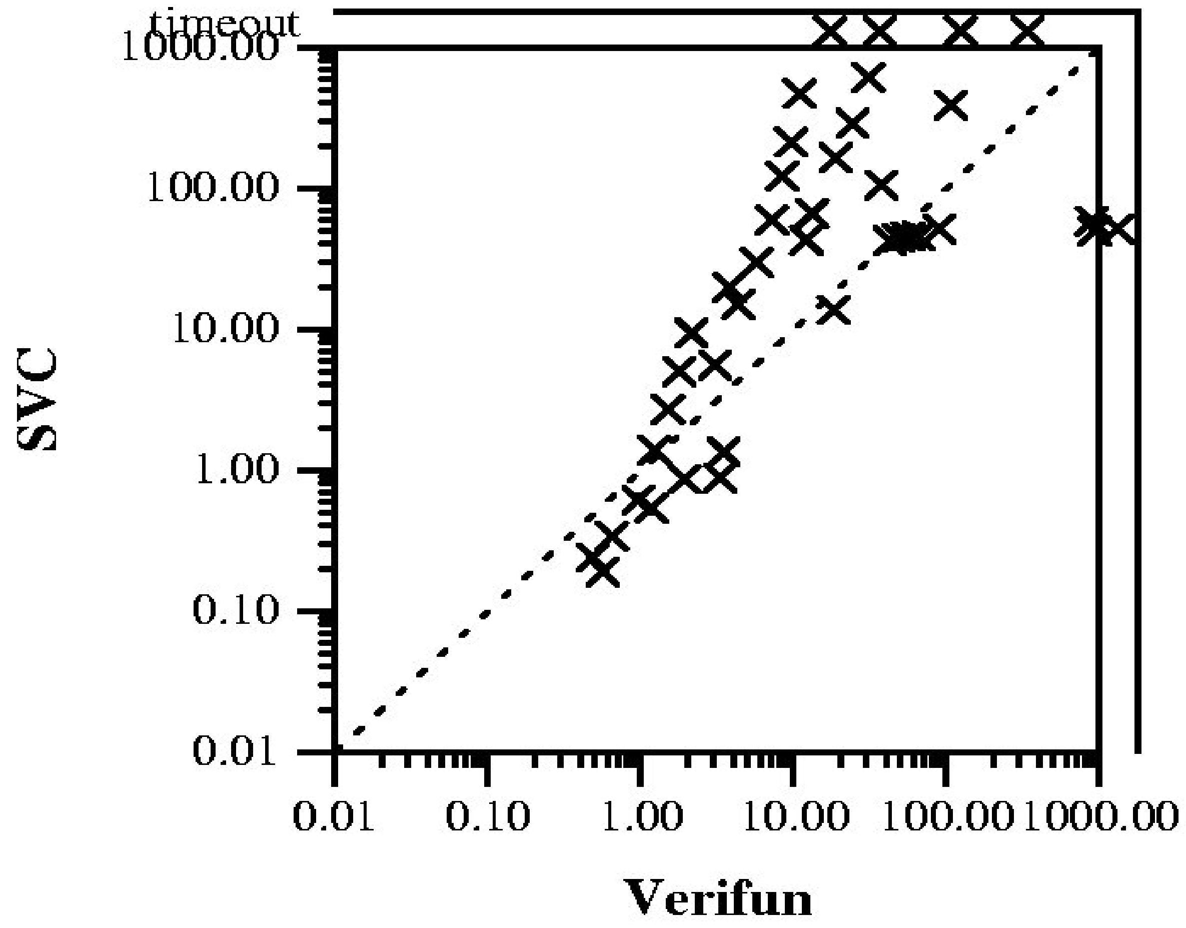
Verifun vs. SVC

on the UCLID benchmarks



Verifun vs. SVC

on the Math-SAT benchmarks



Design choices in Verifun

- Laziness in theory invocation
- Complete vs. partial truth assignments
- Detecting multiple inconsistencies
- Incremental SAT solving
- Backtrackable theories
- Eager proof introduction

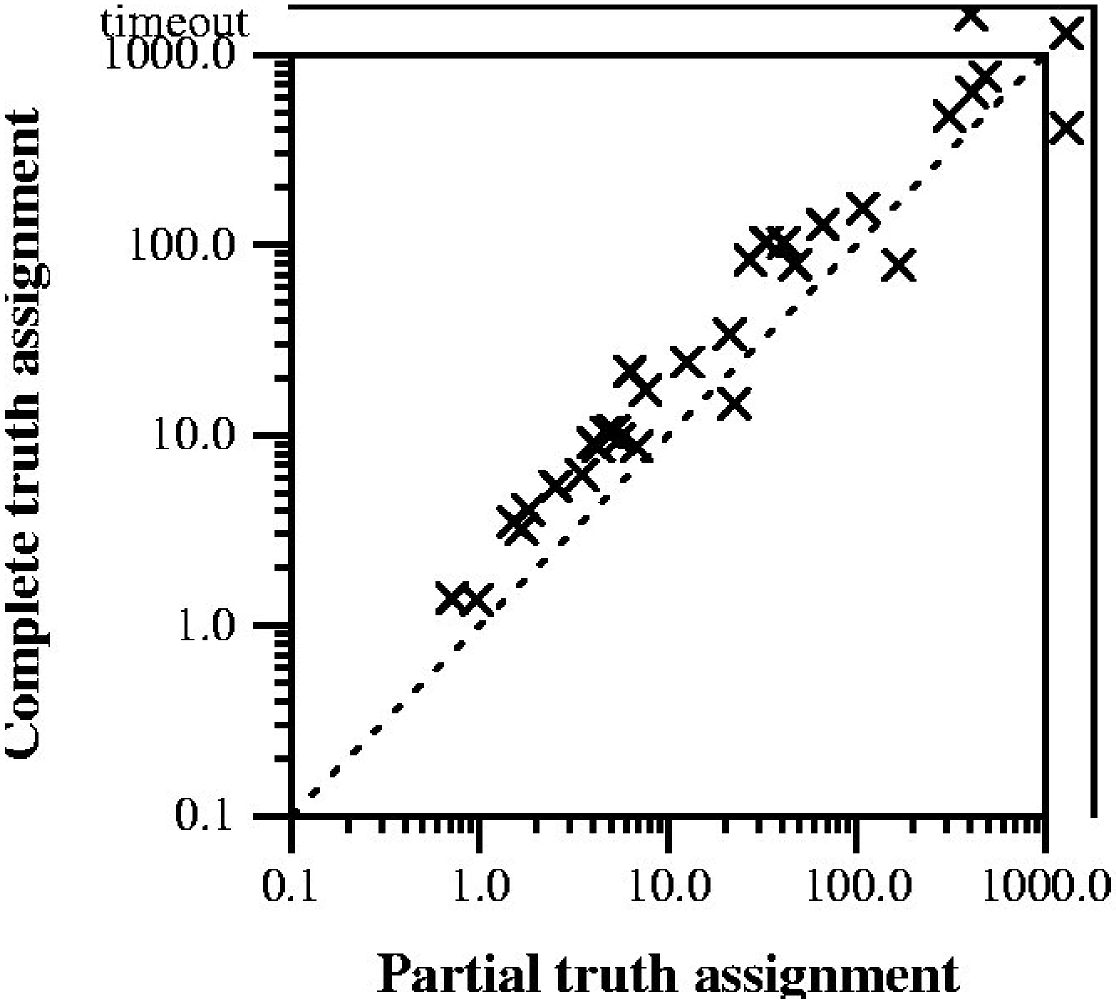
Laziness in Theory Invocation

- In Verifun, theories are invoked only after the SAT solver has found a candidate assignment
- An alternative is to invoke theories eagerly, as the SAT solver makes choices in its backtracking search
  (cf. CVC, Simplify)
- An advantage of the Verifun approach is the ability to use any off-the-shelf SAT solver (zChaff, Berkmin,...)

Complete vs. partial truth assignments

- Assignment returned by SAT solver assigns truth values to all atomic formulae
- Asserting all these formulae might cause theories to do unnecessary work
- An optimisation in Verifun is to determine a minimal subset of literals which suffices to satisfy the SAT problem, and assert only these literals to the theories

Results with partial assignments



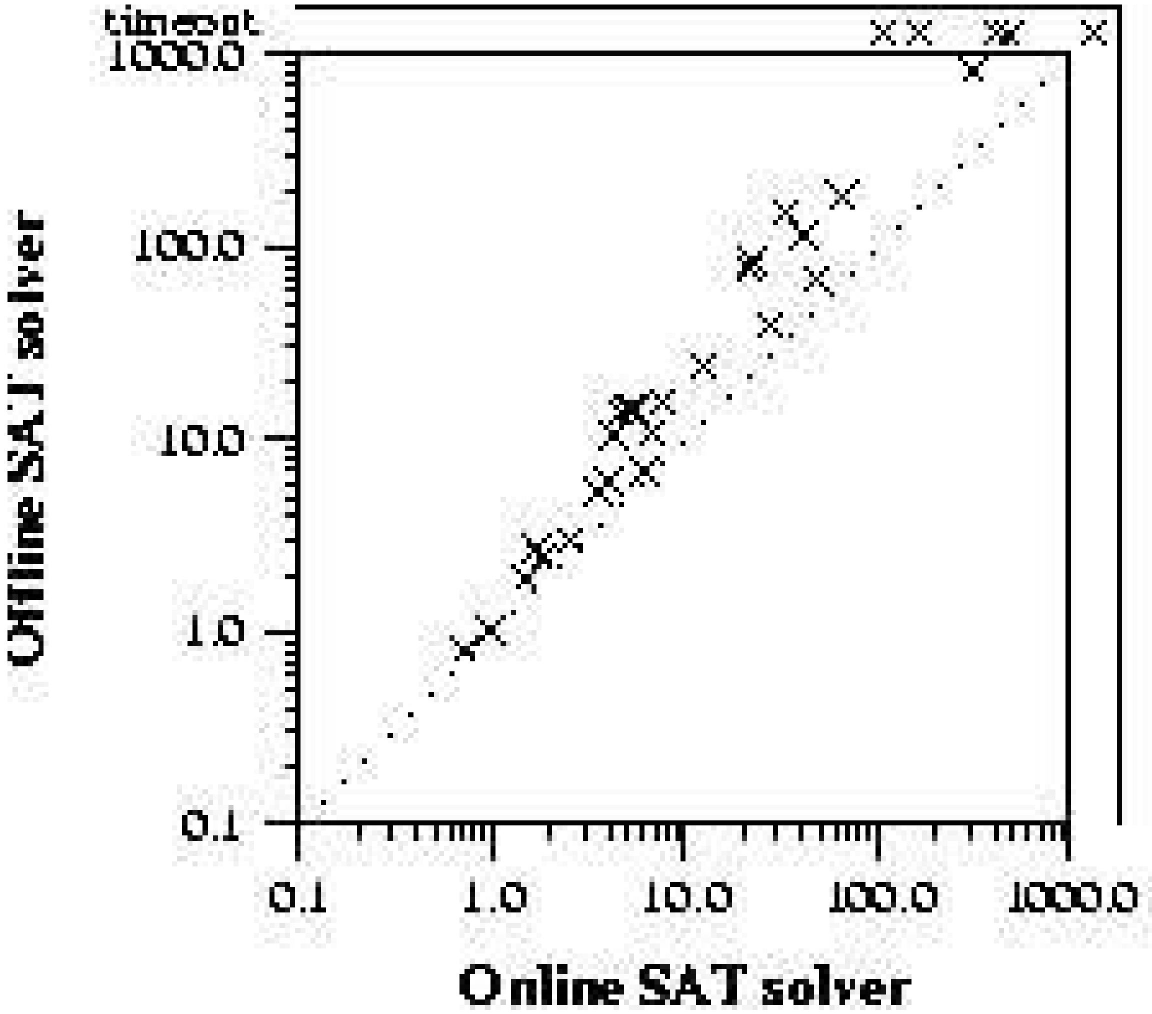
Detecting multiple inconsistencies

- Useful when used with lazy theory invocation
- Given an assignment from the SAT solver, detect as many inconsistencies as possible
- Can reduce number of round-trips to the SAT solver
- Best done with backtrackable theories
- Verifun asserts all the equalities first, then checks each disequality in turn for inconsistency

Incremental SAT solving

- The sequence of CNF formulae given to the SAT solver forms a strengthening chain
- Any assignment that does not satisfy the current problem can safely be rejected in the future
- Verifun used a simple naïve hack to zChaff;
  now zChaff supports incremental solving

Results with naïve incremental SAT



Backtrackable Theories

- With incremental SAT, consecutive assignments returned by the SAT solver would differ only in the assignment to a small suffix of literals
- So it would be advantageous to design theories that do not have to infer the consequences of the common prefix all over again
- For instance: assert literals to theories in increasing order of “decision depth” assigned by the SAT solver

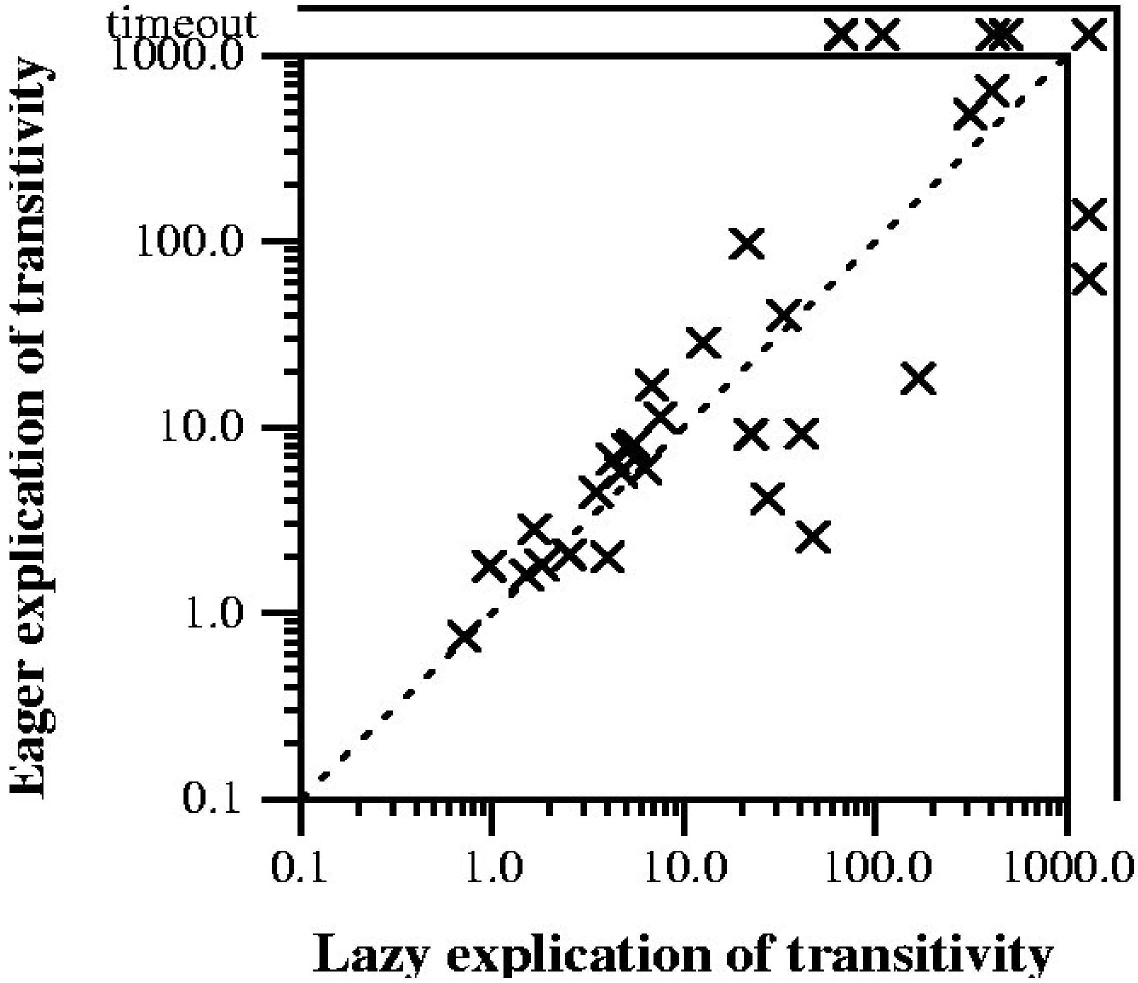
Eager Proof Introduction

- Inspired by the work of Bryant, German and Velev [TOCL 2000]
- Idea: Augment initial SAT problem with additional clauses that encode appropriate inference rules from the theories
- In the extreme case, one can encode enough rules so that only one invocation of the SAT solver is required – the “purely eager” approach

Eager Proof Introduction

- Reduces the number of round-trips to the SAT solver
- But, it is non-trivial to design a procedure that generates a sufficient set of clauses without producing too many clauses
- It seems unlikely that one could deal with arbitrary quantifiers using a purely eager approach

Verifun experiment with eager transitivity



Granularity of Proof Explication

- Suppose the equality decision theory is given

  $a{=}b \;\wedge\; b{=}c \;\wedge\; f(a) \neq f(c)$

- The theory of equality could generate the proof

  $(\, a{=}b \;\wedge\; b{=}c \,) \;\Rightarrow\; f(a) = f(c)$

- Alternatively, it could generate two proofs

  $(\, a{=}b \;\wedge\; b{=}c \,) \;\Rightarrow\; a = c$ (transitivity)

  $a{=}c \;\Rightarrow\; f(a) = f(c)$ (congruence)

Granularity of Proof Explication

- Smaller proofs could reduce the number of rounds
- For instance, the proof

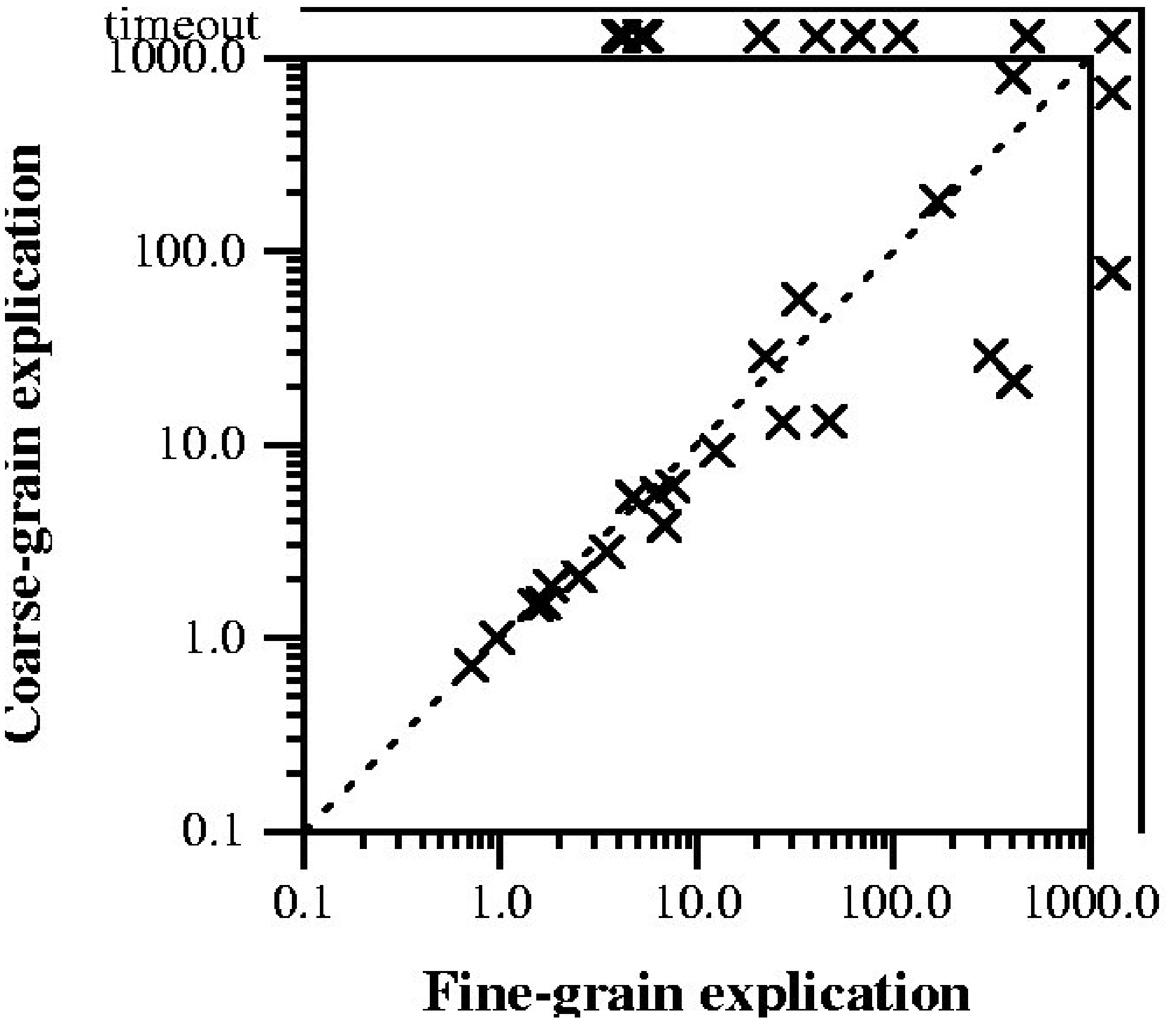
  $$a=c \quad \Rightarrow \quad f(a) = f(c)$$

  might be useful when $a=c$ holds for a different reason (say we had $a=k \land k=c$)
- One complication is that finer-grained explication introduces new atomic formulae

Verifun's proof explication

- Somewhat fine-grained proof explication
  - Given $(a{=}b \wedge b{=}c \wedge c{=}d \wedge f(a) \neq f(d))$,
    Verifun produces $(a{=}b \wedge b{=}c \wedge c{=}d \;\Rightarrow\; a{=}d)$
    and $(a{=}d \;\Rightarrow\; f(a){=}f(d))$
    instead of
    $(a{=}b \wedge b{=}c \;\Rightarrow\; a{=}c)$ $(a{=}c \wedge c{=}d \;\Rightarrow\; a{=}d)$
    and $(a{=}d \;\Rightarrow\; f(a){=}f(d))$

Coarse- vs fine-grained proofs



Aside: Checking Verifun's proofs

- The “proofs” explicated by Verifun's theories are universally valid (in the context of the theories)
- Checking each such proof is easy, since the steps are quite small
- We have used Simplify to check Verifun's proofs, in order to find bugs

Related Work

- CVC [Dill, Stump, Barrett], CVC-Lite [Barrett, Berezin]
- ICS [de Moura, Ruess, Shankar,]
- Math-SAT [Audemard, Bertoli, Cimatti, Kornilowicz, Sebastiani]
- DPLL(T) [Ganzinger, Hagen, Nieuwenhius,Oliveras,Tinelli]
- UCLID [Bryant, Velev, Strichman, Seshia, Lahiri]
- Zapato [Ball,Cook,Lahiri,Zhang]
- TSAT++ [Armando,Castellini,Giunchiglia,Idini,Maratea]

Further Information

- *Theorem Proving Using Lazy Proof Explication*
  Flanagan, Joshi, Ou, Saxe
  CAV 2003

- *An Explicating Theorem Prover for Quantified Formulas*
  Flanagan, Joshi, Saxe
  HP Tech Report (in preparation)

Additional Material

Quantifier Instantiation using matching

- Associate with each quantified formula a pattern, e.g, $\forall x.(\ f(x) = \underline{f(f(x))}\ )$
- Produce quantifier instances for terms that match the pattern (match upto equivalence)
- Example

  $$a{=}b \ \wedge\ f(a){=}b \ \wedge\ f(b) \neq f(a)$$
  $$\wedge\ \forall x.(\ f(x) = \underline{f(f(x))}\ )$$

- Matcher produces instantiation $x := a$

Procedure *checkMonome(..)*

```
checkMonome(m) {  /* returns SAT or an explicated proof */
    R := satisfyTheories(m)
    if  (R $\neq$ SAT) {  return R }
    if  m contains  $\exists x.F(x)$
        such that $m \wedge \neg F(x \leftarrow V_F)$  is propositionally satisfiable
            { return  $(\exists x.F(x)) \Rightarrow F(V_F)$ }
    if  m  contains  $\forall x.F(x)$  for some matching substitution $\sigma$
        such that $m \wedge \neg\sigma(F)$  is propositionally satisfiable
            { return  $(\forall x.F(x)) \Rightarrow \sigma(F)$ }
    return SAT
}
```

requires calls to satisfyProp(..)

Procedure *checkMonome(..)*

```
- checkMonome(m) {  /* returns SAT or an explicated proof */
    R := satisfyTheories(m)
    if  (R $\neq$ SAT) {  return R }
    if  m contains  $\exists x.F(x)$
        such that $m \wedge \neg F(x \leftarrow V_F)$  is propositionally satisfiable
            { return  $(\exists x.F(x)) \Rightarrow F(V_F)$ }
    if  m  contains  $\forall x.F(x)$  for some matching substitution $\sigma$
        such that $m \wedge \neg\sigma(F)$  is propositionally satisfiable
            { return  $(\forall x.F(x)) \Rightarrow \sigma(F)$ }
    return SAT
  }
```

Note that these guards
can be weakened

A simpler *checkMonome(..)*

```
checkMonome(m) {  /* returns SAT or an explicated proof */
    R := satisfyTheories(m)
    if  (R $\neq$ SAT) {  return R }
    if  m contains  $\exists x.F(x)$
        such that $\exists x.F(x)$ is not in E
            { add $\exists x.F(x)$ to E  ; return  $(\exists x.F(x)) \Rightarrow F(V_F)$ }
    if  m  contains  $\forall x.F(x)$  for some matching substitution $\sigma$
        such that  $(\sigma, \forall x.F(x))$ is not in A
            { add $(\sigma, \forall x.F(x))$ to A ; return  $(\forall x.F(x)) \Rightarrow \sigma(F)$ }
    return SAT
}
```

where E,A record the instantiated quantified formulae