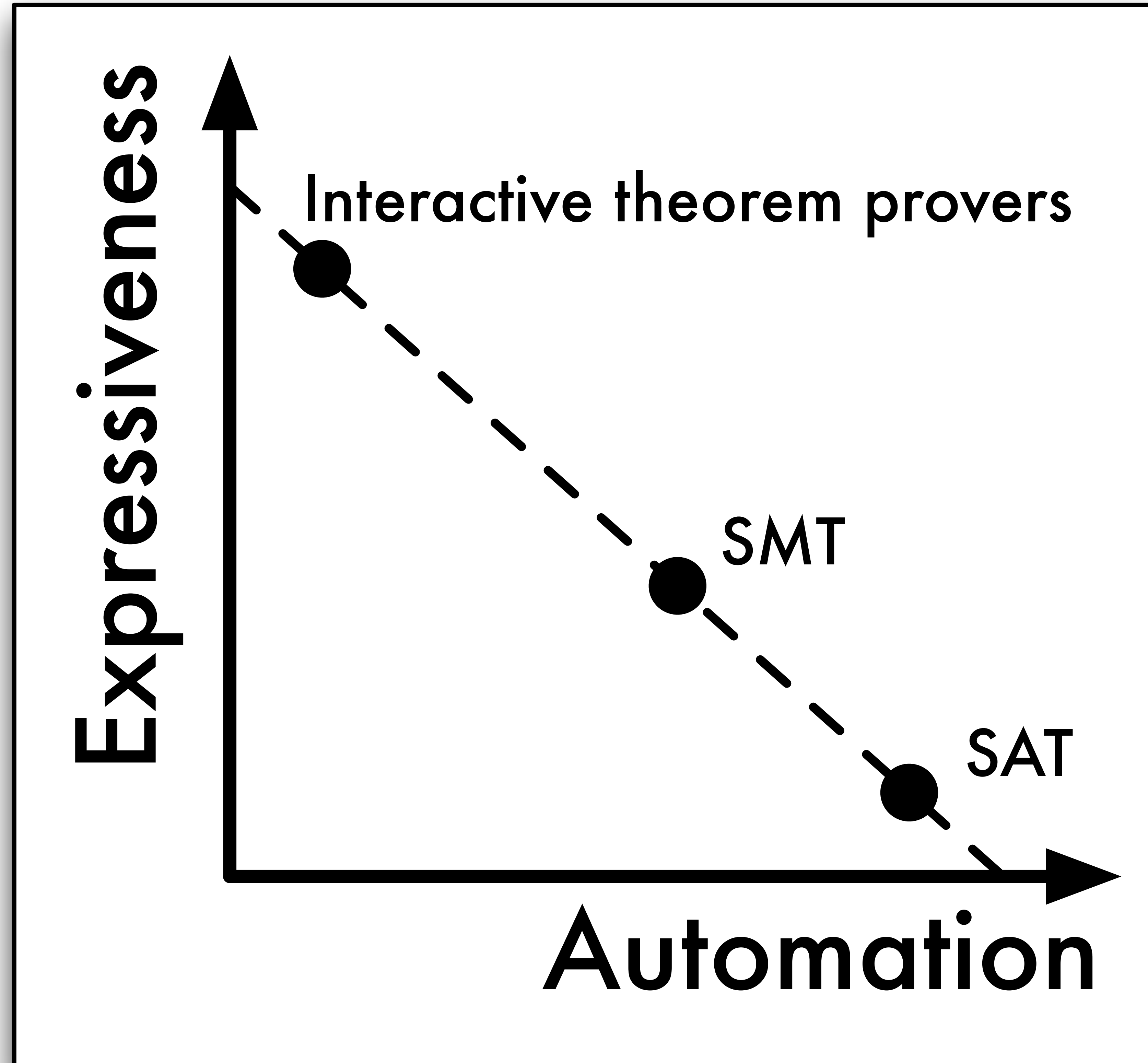
Combining solvers with interactive theorem provers

Leopoldo Teixeira
lmt@cin.ufpe.br

based on works by Tjark Weber, Sascha Böhme, Clément Hurlin et al.


Expressiveness
Interactive theorem provers
SMT
SAT
Automation

Higher-order logic theorem proving

- Supports a variety of specification styles: axiomatic, declarative, functional, executable, etc.
- Analyzing systems/programs thoroughly
- Finding design and specification errors early
- High assurance (mathematical, machine checked proof)

Applications

- Good where cost of failure is highest
  - safety-critical systems (NASA)
  - security-critical systems (Protocols)
  - hardware products (Intel)
- however, very **high cost**

pvs

New Open Recent Save Print Undo Redo Cut Copy Paste Search Preferences Help

scratch 1 | PVS Welcome 2 | SPLrefinement.pvs 3

```
CK: TYPE
[||] : [CK->[AM->[Conf->finite_sets[Asset].finite_set]]]

% Axiom over ck evaluation
amRef: AXIOM
  FORALL(am1,am2): |>(am1,am2) =>
    FORALL(K:CK,c:Conf):
       wfProduct(([| K |](am1))(c))
    => wfProduct(([| K |](am2))(c)) AND
          (([| K |](am1))(c) |- ([| K |](am2))(c) )

ck,ck1,ck2,ck3: VAR CK

% Definition <CK equivalence>
equivalentCKs(ck1,ck2): bool =
  [|ck1|]=[|ck2|]

% Theorem <CK equivalence properties>
eqCK: THEOREM relations[CK].equivalence?( equivalentCKs )

weakerEqCK(fm,ck1,ck2): bool =
  FORALL am:
    FORALL c: {| fm |}(c) => ([| ck1 |](am))(c) = ([| ck2 |](am))(
c)
```

-:--- SPLrefinement.pvs 30% (109,49) (PVS :ready)

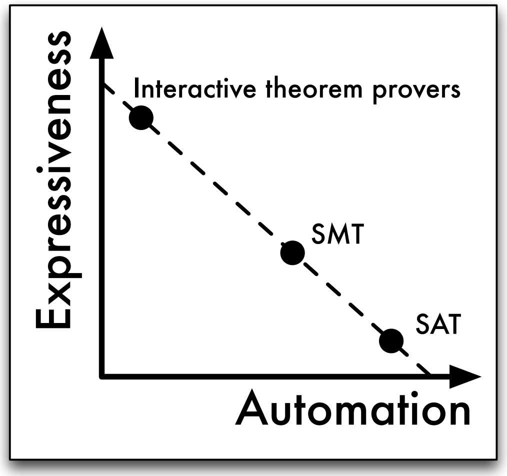
```
Rerunning step: (instantiate 2 c1)
Instantiating the top quantifier in 2 with the terms:
 c1,
this simplifies to:
plRefEq.2.2 :

[-1]  {||}(F(pl1))(c1)
  |-------
[1]   EXISTS c2:
        {||}(F(pl2))(c2) AND
         (([||](K(pl1))(A(pl1))(c1)) |- ([||](K(pl2))(A(pl2))(c2)))
{2}   ({||}(F(pl1))(c1))

which is trivially true.

This completes the proof of plRefEq.2.2.


This completes the proof of plRefEq.2.

Q.E.D.



Run time  = 0.10 secs.
Real time = 1.48 secs.
```

U:**- *pvs* 97% (488,0) (ILISP :ready)

Manual guidance

THEOREM

empty?(S) IFF card(S) = 0




```
(""
 (skolem!)
 (prop)
 (("1"
   (rewrite "emptyset_is_empty?[T]")
   (replace -1)
   (use "card_emptyset"))
  ("2"
   (rewrite "card_def")
   (rewrite "Card_bijection")
   (skolem!)
   (delete -)
   (grind)
   (typepred "f!1(x!1)")
   (propax))))
```

Proof scripts can become big

```
(""
 (skolem 1 (am1 am2))
 (lemma asRefCompositional)
 (flatten)
 (expand "|>")
 (flatten)
 (use "measure_induction[finite_sets[AssetName].finite_set,nat,
finite_sets[AssetName].Card,<]")
 (("1"
   (bddsimp)
   (skolem 1 ans)
   (flatten)
   (case "EXISTS(an:AssetName): ans(an) and dom(am1)(an)")
   (("1"
     (skolem -1 an)
     (flatten)
     (lemma sets_lemmas[AssetName].nonempty_member)
     (expand member)
     (instantiate -1 ans)
     (expand nonempty?)
     (bddsimp)
     (("1" (instantiate 1 an) (propax))
      ("2"
       (lemma set_aux_lemmas[AssetName].setMember)
       (expand member)
       (instantiate -1 (ans an))
       (assert)
       (skolem -1 ans2)
       (flatten)
       (instantiate -5 ans2)
       (lemma set_aux_lemmas[AssetName].cardUnion)
       (expand member)
       (instantiate -1 (an ans2))
       (assert)
       (skolem 3 a)
       (replace -2)
       (flatten)
       (instantiate -10 an)
       (assert)
       (skolem -10 (a1 a2))
       (flatten)
       (instantiate -6 "union(a,a1)")
       (lemma "maps[AssetName,Asset].mapAM")
       (instantiate -1 (am1 an ans2))
       (assert)
       (skolem -1 ax)
       (typepred am1)
       (flatten)
       (expand unique)
       (instantiate -2 (an ax a1))
       (assert)
       (replace -2)
       (hide (-1 -2 -3))
       (replace -1)
       (lemma "maps[AssetName,Asset].mapAM")
       (instantiate -1 (am2 an ans2))
       (assert)
       (skolem -1 ay)
       (flatten)
       (typepred am2)
       (expand unique)
       (instantiate -2 (an ay a2))
       (assert)
       (replace -2)
       (hide (-1 -2 -3))
       (lemma asRefCompositional)
       (assert)
       (case "union(a, union(singleton[Asset](a1), map(am1,
ans2))) = union(union(a, singleton[Asset](a1)), map(am1, ans2)))")
       (("1"
         (assert)
         (flatten)
         (instantiate -2 (a1 a2 "union(a,map(am2,ans2))"))
         (assert)
         (case "union(union(a, singleton[Asset](a1)), map(am2,
ans2))) = union(union(a, singleton[Asset](a1)), map(am2, ans2))")
         (("1"
           (assert)
           (flatten)
           (case "union(a, map(am2, union(singleton[AssetName](an),
ans2))) = union(singleton[Asset](a2), union(a, map(am2,
           (("1"
             (assert)
             (case "union(singleton[Asset](a2), union(a, map(am2,
ans2))) = union(a, map(am2, union(singleton[AssetName](an),
             (("1"
               (use assetRefinement)
               (expand* preorder? transitive?)
               (flatten)
               (instantiate -2
                ("union(a, union(singleton[Asset](a1), map(am1, ans2)))"
                 "union(union(a, singleton[Asset](a1)), map(am2, ans2))"
ans2)))"))    "union(singleton[Asset](a2), union(a, map(am2,
               (assert))
              ("2" (propax))))
            ("2"
             (decompose-equality 1)
             (replace -5)
             (expand* union singleton member)
             (bddsimp))))
          ("2"
           (decompose-equality 1)
           (expand* union singleton member)
           (bddsimp))))
        ("2"
         (decompose-equality 1)
         (expand* union singleton member))))))
    ("2"
     (lemma "maps[AssetName,Asset].notExists")
     (copy -1)
     (instantiate -1 (am1 ans))
     (bddsimp)
     (replace -5)
     (instantiate -2 (am2 ans))
     (bddsimp)
     (replace -1)
     (replace -2)
     (skolem 2 a)
     (flatten)
     (use assetRefinement)
     (expand* preorder? reflexive?)
     (flatten)
     (instantiate -1 "union(a,emptyset)")
     (assert))))
  ("2" (lemma wf_nat) (grind))))
```

ITP vs. Solvers

- rich specification logic [undecidable]
- infinite state space
- lots of manual effort
- little to no feedback for ‘unprovable’ theorems

- decidable theories
- state explosion problem
- high degree of automation
- better feedback: counterexamples

Why not combine the best of both worlds:

ITP + Solvers (ATP)?

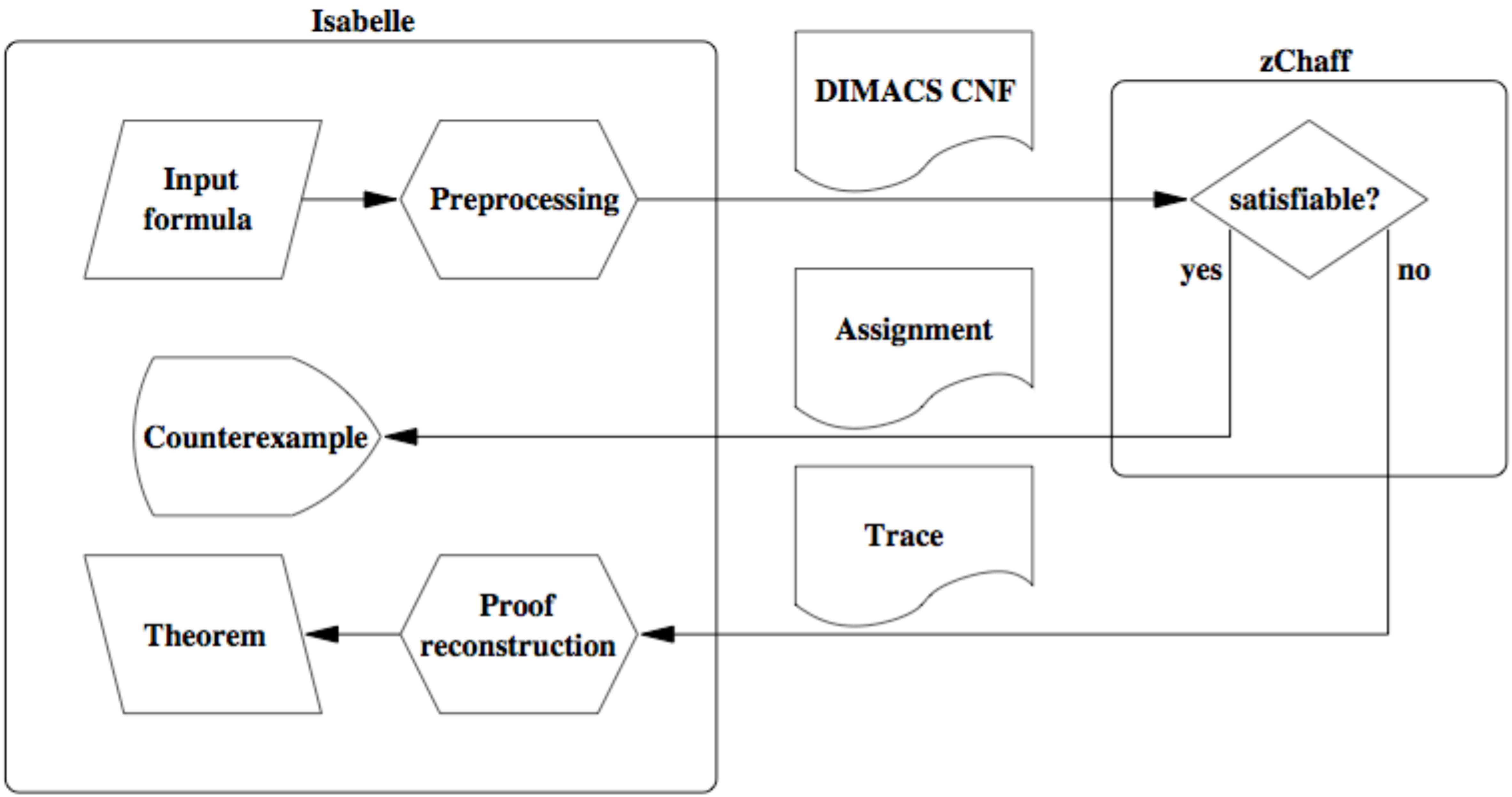
Integrate **zChaff** and **Isabelle/HOL** to prove theorems of propositional logic

Integrating a SAT Solver with an LCF-style Theorem Prover
Tjark Weber. In Proceedings of the Third Workshop on Pragmatics of Decision Procedures in Automated Reasoning (PDPAR 2005), volume 144(2) of Electronic Notes in Theoretical Computer Science, pages 67-78. Elsevier, January 2006.

zChaff

- A leading SAT solver (winner of the SAT 2002 and SAT 2004 competitions in several categories)
- Returns a satisfying assignment, or . . .
- . . . a proof of unsatisfiability (since 2003)

Solution overview



Interesting point: zChaff provides feedback to Isabelle, so we can also verify its soundness

Running example

$$\phi \equiv (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee x_3).$$

Preprocessing

$$\phi \equiv (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee x_3).$$

$$\downarrow$$

$$\vdash \phi = \phi.$$

$$\downarrow$$

−1	2	0
−2	−3	0
1	2	0
−2	3	0

CNF conversion
Normalization
Removal of duplicate literals
Removal of tautological clauses
Output:
theorem in the form $\varphi = \varphi_*$

Proof of unsatisfiability

```
CL: 4 <= 2 0
VAR: 2 L: 0 V: 1 A: 4 Lits: 4
VAR: 3 L: 1 V: 0 A: 1 Lits: 5 7
CONF: 3 == 5 6
```

zChaff proof trace

Derived clauses

$$\begin{array}{cccc} \textbf{0} & \textbf{1} & \textbf{2} & \textbf{3} \end{array}$$

$$\boxed{\phi \equiv (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee x_3).}$$

clause id resolvents

`CL: 4 <= 2 0` $\longrightarrow x_2$

Variable Assignments

0 1 2 3

$$\phi \equiv (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee x_3).$$

clause id resolvents

CL: 4 <= 2 0 $\longrightarrow x_2$

variable id level value antecedent

VAR: 2 L: 0 V: 1 A: 4 Lits: 4 $\longrightarrow x_2 = true$

VAR: 3 L: 1 V: 0 A: 1 Lits: 5 7 $\longrightarrow x_3 = false$

Conflict Clause

0 1 2 3

$$\phi \equiv (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee x_3).$$

clause id resolvents

CL: 4 <= 2 0 $\longrightarrow x_2$

variable id level value antecedent

VAR: 2 L: 0 V: 1 A: 4 Lits: 4 $\longrightarrow x_2 = true$

VAR: 3 L: 1 V: 0 A: 1 Lits: 5 7 $\longrightarrow x_3 = false$

conflict clause id

CONF: 3 == 5 6 $\longrightarrow$ **conflict clause**

Proof Reconstruction

```
resolution: list Thm -> Thm
```

$$[\phi \Rightarrow X \vee Y \vee Z, \phi \Rightarrow W \vee \neg Y \vee Z]$$

$$\downarrow$$

$$\phi \Rightarrow X \vee W \vee Z$$

Proof reconstruction

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

prove_literal var_id =
  let
    th_ante = prove_clause (antecedent_of var_id)
    var_ids = filter (fn i => i <> var_id)
                     (var_ids_in_clause th_ante)
  in resolution (th_ante :: map prove_literal var_ids)
```

1. Call **`prove_clause`** for conflict clause
2. Call **`prove_literal`** for every literal in the conflict clause, to show that literal must be false
3. Finally resolving the conflict clause with these negated literals yields the theorem $\varphi_* \rightarrow$ *False*.

$$[\phi \Rightarrow X, \phi \Rightarrow \neg X] \longrightarrow \phi \Rightarrow false$$

Resolution Proof tree

$$\boxed{\phi \equiv \overset{0}{(\neg x_1 \vee x_2)} \wedge \overset{1}{(\neg x_2 \vee \neg x_3)} \wedge \overset{2}{(x_1 \vee x_2)} \wedge \overset{3}{(\neg x_2 \vee x_3)}.}$$

$$\dfrac{\dfrac{{}^{3}\,\neg x_2 \vee x_3 \qquad \dfrac{{}^{2}\,x_1 \vee x_2 \qquad {}^{0}\,\neg x_1 \vee x_2}{{}^{4}\,x_2}}{x_3} \qquad \dfrac{{}^{1}\,\neg x_2 \vee \neg x_3 \qquad \dfrac{{}^{2}\,x_1 \vee x_2 \qquad {}^{0}\,\neg x_1 \vee x_2}{{}^{4}\,x_2}}{\neg x_3}}{\perp}$$

A set of clauses is unsatisfiable iff the empty clause is derivable via resolution.

Improving proof reconstruction

- 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.

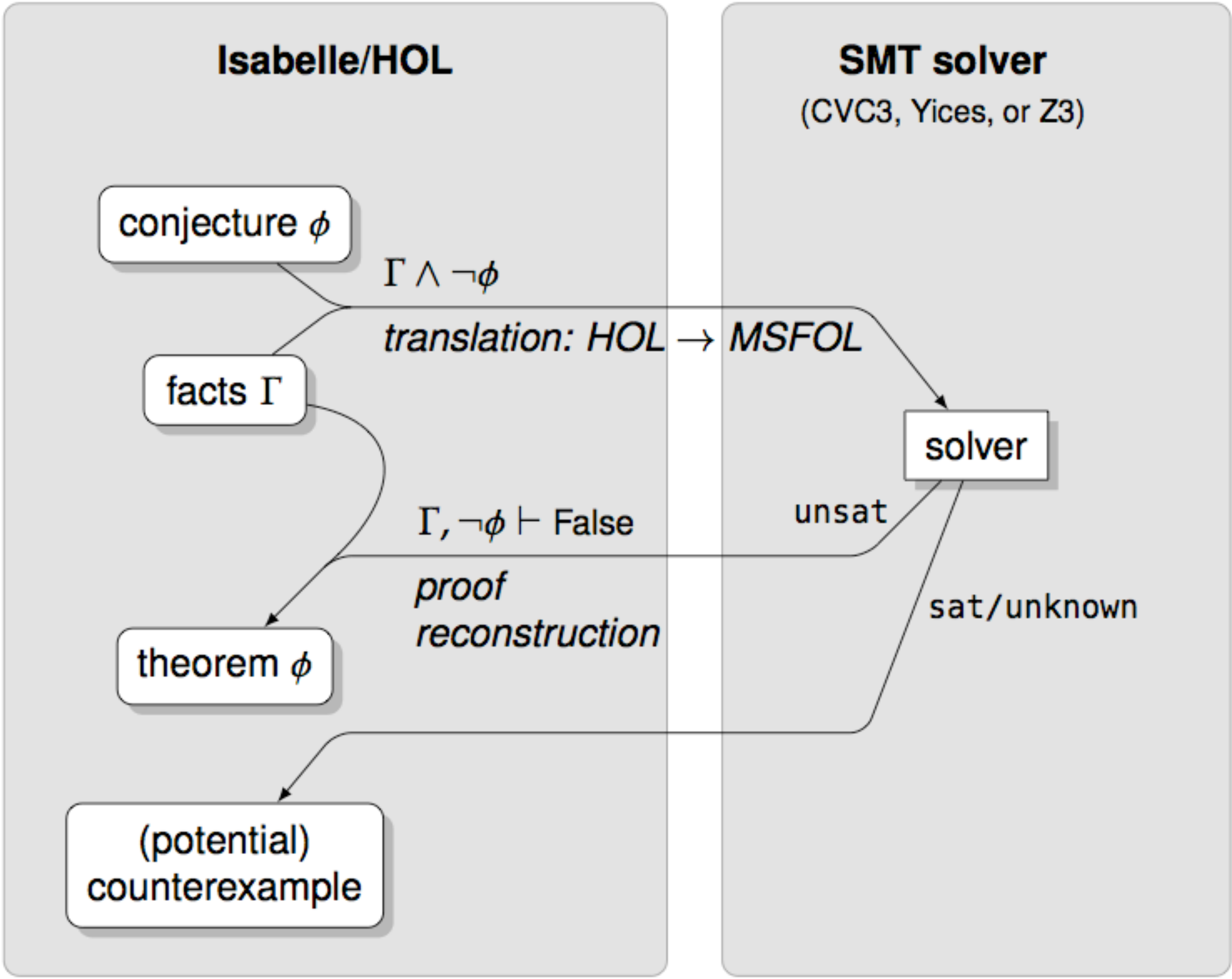
Evaluation

- Compare integration with built-in procedures (*auto*, *blast*, *fast*)
- 42 propositional problems in TPTP v2.6.0
- Problems were negated, so that unsatisfiable problems became provable
- 19 are solved in less than 1 sec

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

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

We can apply the **same** concept to do more interesting things



Proving Theorems of Higher-Order Logic with SMT Solvers.
Sascha Böhme. Dissertation, Technische Universität München, 2012.

Theorem example (i)

$$x_3 = |x_2| - x_1 \wedge x_4 = |x_3| - x_2 \wedge x_5 = |x_4| - x_3 \wedge x_6 = |x_5| - x_4 \wedge$$
$$x_7 = |x_6| - x_5 \wedge x_8 = |x_7| - x_6 \wedge x_9 = |x_8| - x_7 \wedge x_{10} = |x_9| - x_8 \wedge$$
$$x_{11} = |x_{10}| - x_9 \longrightarrow x_1 = x_{10} \wedge x_2 = x_{11}$$

SMT solver almost instantaneously finds a proof, and proof reconstruction for Z3 takes only a few seconds. In contrast, Isabelle's arithmetic decision procedure requires several minutes to prove the same result.

Theorem example (ii)

$$\forall f.\ \mathsf{map}\ f\ \mathsf{Nil} = \mathsf{Nil}$$
$$\forall f, x, xs.\ \mathsf{map}\ f\ (\mathsf{Cons}\ x\ xs) = \mathsf{Cons}\ (f\ x)\ (\mathsf{map}\ f\ xs)$$

$$\downarrow$$

$$\mathsf{map}\ (\lambda x^{\mathsf{int}}.\ x + 2)\ (\mathsf{Cons}\ y\ (\mathsf{Cons}\ 3\ \mathsf{Nil})) = \mathsf{Cons}\ (y + 2)\ (\mathsf{Cons}\ 5\ \mathsf{Nil})$$

SMT solver is able to prove this
higher-order theorem instantaneously

Evaluation

	Arrow	*FFT*	*FTA*	*Hoare*	*Jinja*	*NS*	*QE*	*S2S*	*SN*	All	Uniq.
CVC3	36%	18%	53%	51%	37%	29%	21%	57%	55%	41.8%	1.3%
Yices	29%	18%	51%	51%	37%	31%	22%	**59%**	59%	41.7%	.9%
Z3	**48%**	18%	**62%**	**54%**	**47%**	**42%**	**25%**	58%	**62%**	**48.5%**	**5.8%**
SMT	**50%**	**23%**	66%	**65%**	**48%**	42%	27%	**66%**	63%	**52.4%**	8.8%
ATPs	40%	21%	**68%**	55%	37%	**46%**	**31%**	55%	**70%**	49.9%	6.3%
All	55%	28%	73%	66%	48%	50%	41%	73%	72%	58.7%	–

9 Isabelle theories with altogether 1591 proof goals

SMT solvers prove > 50% of all goals

SMT solvers find 8.8% unique proofs,
i.e., proofs for goals on which all ATPs fail.

Hence, SMT solvers increase proof automation.

Non-trivial goals

	Arrow	*FFT*	*FTA*	*Hoare*	*Jinja*	*NS*	*QE*	*S2S*	*SN*	All	Uniq.
CVC3	23%	14%	28%	36%	31%	18%	7%	25%	27%	24.3%	2.1%
Yices	11%	14%	30%	40%	33%	20%	7%	26%	44%	25.4%	1.5%
Z3	**36%**	13%	**41%**	**46%**	**46%**	**34%**	7%	**28%**	**46%**	**33.0%**	**9.7%**
SMT	**37%**	18%	**43%**	**54%**	**46%**	34%	8%	**33%**	48%	**35.8%**	8.9%
ATPs	32%	18%	42%	42%	33%	**38%**	**19%**	26%	**59%**	33.8%	6.9%
All	41%	23%	50%	57%	46%	44%	23%	42%	61%	42.7%	–

non-trivial goals require additional arguments

users tend to query existing proof automation before attempting to find the proof themselves

SMT solvers adds ~8.9% unique proofs again, i.e., proofs for goals on which all ATPs fail.

Z3 alone contributes to 9.7% unique proofs

Benefits of decision procedures

- SMT solvers perform really well over theories containing lambda abstractions
- Arithmetic support of SMT solvers is in general not essential, but certain goals can benefit.
- Direct support for datatypes, records and type definition by Z3 gives no clear benefits.
- Extra-logical information that is automatically inferred by our current simple algorithm slightly deteriorates the success rates of SMT solvers.

$\ll$ and $\gg$ shifting by a natural number.
ucast x extend x by padding zeros to its front.
mask n is defined as $(1 \ll n) - 1$ for natural numbers n.
x_n select bit n from a bitvector x and interpret it as Boolean
&, | and ! bitwise conjunction, disjunction and negation.
u is a bitvector of size 12
v and w are bitvectors of size 32
x and y are bitvectors that have arbitrary but equal size.

$$u \neq 0 \longrightarrow (1 \ll 20) - 1 < (\text{ucast}\, u \ll 20)^{32\ \text{word}} \tag{2.1}$$

$$(\text{if } w = 0 \text{ then } v \leq 0 \text{ else } v \leq w - 1) \longrightarrow v \neq -1 \tag{2.2}$$

$$n < 32 \longrightarrow 1^{32\ \text{word}} \leq (1 \ll n) \tag{2.3}$$

$$v \mathbin{\&} \text{mask}\ 14 = 0 \longrightarrow v + (w \gg 20 \ll 2) \mathbin{\&} (!\ (\text{mask}\ 14)) = v \tag{2.4}$$

$$v \mathbin{\&} \text{mask}\ n = 0 \wedge (\forall n'.\, n \leq n' \wedge n' < 32 \longrightarrow \neg w_{n'}) \longrightarrow v + w \mathbin{\&} (!\ (\text{mask}\ n)) = v \tag{2.5}$$

$$0 < y \wedge y \leq x \longrightarrow (x - y) \text{ div } y = (x \text{ div } y) - 1 \tag{2.6}$$

$$(x \mathbin{\&} y) + (x \mid y) = x + y \tag{2.7}$$

$$x \mathbin{\&} y = 0 \longrightarrow x + y = x \mid y \tag{2.8}$$

Proving problems of fixed-size bitvectors works well unless functions that are unsupported by SMT solvers are used.

Yet another example

$$(X \cap Y = \emptyset \wedge X \setminus Z = X) \Rightarrow X \cap (Y \cup Z) = \emptyset.$$

I have proved this using PVS+Yices :-)
The PVS strategy **`grind`** (automatic procedure) fails

Practical Proof Reconstruction for First-Order Logic and Set-Theoretical Constructions. Clément Hurlin, Amine Chaieb, Pascal Fontaine, Stephan Merz, and Tjark Weber. In Proceedings of Isabelle Workshop (ISABELLE-WS), pages 2-13, Bremen, Germany, July 2007.

Conclusions

- Combine the best of both worlds
  - rich specification logics
  - counterexamples for unprovable formulae
  - performance improvements
  - automation
  - plus: solver verification
  - still room for improvements, there is work to be done!