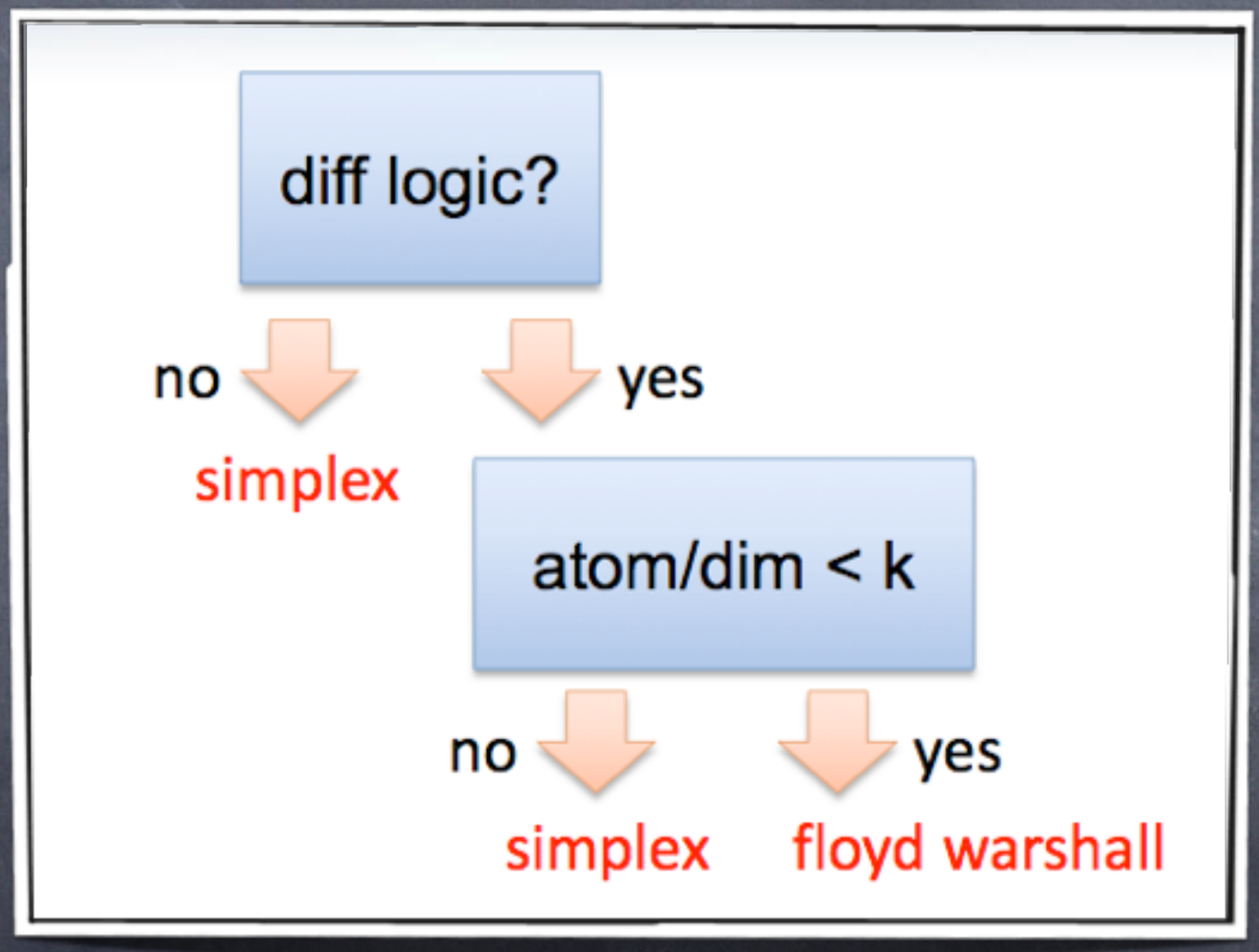
The Strategy Challenge in SMT Solving

Part II

Leo de Moura and Grant Passmore

Probing Formula Measures



Yices LRA Strategy

Probing Formula Measures

$$\text{orelse}(\text{then}(\text{failif}(\text{diff} \wedge \frac{\text{atom}}{\text{dim}} > \text{k}),\ \text{simplex}),\ \text{floydwarshall})$$

Fail if condition is not satisfied.
Otherwise, do nothing.

Yices LRA Strategy

Formula Measure Examples

bw: Sum total bit-width of all rational coefficients of polynomials in case.
diff: True if the formula is in the difference logic fragment.
linear: True if all polynomials are linear.
dim: Number of arithmetic constants.
atoms: Number of atoms.
degree: Maximal total multivariate degree of polynomials.
size: Total formula size.

Useful tactical syntactic sugar:

$$\mathsf{if}(c,\ t_1,\ t_2) = \mathsf{orelse}(\mathsf{then}(\mathsf{failif}(\neg c), t_1), t_2)$$

$$\mathsf{when}(c,\ t) = \mathsf{if}(c,\ t,\ \mathsf{skip})$$

Under/Over Approximations

Under-approximation

unsat answers cannot be trusted

Over-approximation

sat answers cannot be trusted

Under/Over Approximations

Under-approximation

Example: QF_NIA model finders

add bounds to unbounded variables (and blast)

Over-approximation

Example: Boolean abstraction

Combining under and over is bad!

sat and unsat answers cannot be trusted.

Under/Over Approximations

We want to write tactics that can check whether a goal is the result of an abstraction or not.

Solution

Associate an precision attribute to each goal.

Parameterised Decision Engines as Tacticals

- Abstract Partial Cylindrical Algebraic Decomposition (See my PhD thesis or CiE Turing Centenary paper with Paul Jackson)

AP-CAD (tactic) = tactic

then(then(simplify, gaussian), orelse(modelfinder, smt(apcad(icp))))

- Even full SMT lazy search loop!

then(preprocess, smt(finalcheck))

Apply “cheap” propagation/pruning steps;
and then apply complete “expensive” procedure

The Strategy Challenge

To build theoretical and practical tools allowing users to exert strategic control over core heuristic aspects of high-performance SMT solvers

Caveat Emptor

We shall not...

- give a new *theoretical framework* or *rule-based formalism* characterising the class of all SMT proof strategies (as in STRATEGO for rewriting)
- prove any theorems about the algebraic structure underlying the class of all SMT proof strategies (as in the PROOF MONAD of Kirchner-Munoz)
- propose a concrete syntax for SMT proof strategies, extending, e.g., the SMT-LIB standard

All worthy goals, but to attempt them would be premature.

Caveat Emptor

Our goals are more modest and practical...

- To bring awareness to the crucial role heuristics play in high-performance SMT
- To convince SMT solver developers that providing flexible, principled methods (i.e., a *strategy language*) for end-users to exert fine-grained control over heuristic aspects of their solvers is an important undertaking
- To show how the adaptation of some ideas of strategy drawn from the LCF and Argonne paradigms can go a long way towards these goals
- To stress how important being explicit about heuristic strategies is for *scientific reproducibility*

Strategies in Action

Z3 (QF_LIA)

RAHD

MetiTarski

Z3 LIA Strategies in Action

Z3 3.0 won QF_LIA at SMT-COMP'11 with this tactic:

```
then(preamble, orelse(mf, pb, bounded, smt)

where the preamble, mf, pb and bounded tactics are defined as

preamble = then(simplify, propagate-values, ctx-simplify,
                lift-if, gaussian, simplify)

mf       = then(failif(not is-ilp), propagate-bounds,
           orelse(tryfor(mip, 5000),
                  tryfor(smt-no-cut(100), 2000),
                  then(add-bounds(-16, 15), smt),
                  tryfor(smt-no-cut(200), 5000),
                  then(add-bounds(-32, 31), smt),
                  mip))

pb       = then(failif(not is-pb), pb2bv, bv2sat, sat)

bounded  = then(failif(unbounded),
           orelse(tryfor(smt-no-cut(200), 5000),
                  tryfor(smt-no-cut-no-relevancy(200), 5000),
                  tryfor(smt-no-cut(300), 15000)))
```

Z3 LIA Strategies in Action

To demonstrate the benefits of our approach we run all QF_LIA benchmarks using the following variations of the strategy above:

pre = then(preamble, smt)
pre+pb = then(preamble, orelse(pb, smt))
pre+bounded = then(preamble, orelse(bounded, smt))
pre+mf = then(preamble, orelse(mf, smt))
combined = then(preamble, orelse(mf, pb, bounded, smt)

Z3 LIA Strategies in Action

	smt		pre		pre+pb		pre+bounded		pre+mf		combined	
benchmark family	failed	time (s)	failed	time (s)	failed	time (s)	failed	time (s)	failed	time (s)	failed	time (s)
Averest (19)	0	4.0	0	5.9	0	6.0	0	5.9	0	5.9	0	5.9
bofill sched (652)	1	1530.7	1	1208.3	1	1191.5	1	1205.4	1	1206.0	1	1205.9
calypto (41)	1	2.0	1	7.7	1	8.0	1	7.8	1	7.9	1	7.8
CAV 2009 (600)	190	1315.3	190	1339.3	190	1329.7	190	1342.7	1	8309.5	1	8208.1
check (5)	0	0.1	0	0.1	0	0.1	0	0.1	0	0.1	0	0.1
CIRC (51)	17	188.4	17	239.8	17	238.2	17	336.1	17	221.5	8	158.66
convert (319)	206	1350.5	190	3060.6	190	3025.9	0	112.7	190	3030.2	0	112.5
cut lemmas (100)	48	2504.0	48	2532.4	48	2509.2	48	2543.4	27	3783.9	27	3709.0
dillig (251)	68	1212.0	68	1237.6	68	1226.9	68	1242.5	3	2677.8	3	2763.9
mathsat (121)	0	171.4	0	150.2	0	149.9	0	151.1	0	150.9	0	150.2
miplib2003 (16)	5	53.8	5	57.7	5	424.4	5	109.5	5	58.8	5	430.5
nec-smt (2780)	147	224149.0	8	59977.4	8	59968.3	8	59929.3	8	60042.1	8	60032.9
pb2010 (81)	43	90.3	43	96.2	25	2581.2	43	146.3	43	96.2	25	2583.1
pidgeons (19)	0	0.3	0	0.4	0	0.4	0	0.3	0	0.3	0	0.3
prime-cone (37)	13	9.6	13	9.5	13	9.5	13	9.7	0	11.0	0	11.0
rings (294)	48	4994.4	46	5973.7	46	6016.2	48	9690.0	46	6024.6	48	9548.2
rings pre (294)	57	441.5	54	1288.7	54	1261.9	54	1260.9	54	1274.7	54	1261.5
RTCL (2)	0	0.1	0	0.1	0	0.1	0	0.1	0	0.1	0	0.1
slacks (251)	135	1132.9	136	550.0	136	545.9	136	550.8	53	8969.3	53	8803.9
total (5938)	978	239153.0	819	77737.4	801	80495.2	631	78646.5	449	95872.4	234	98995.2

RAHD Strategies in Action

```
RAHD: Real Algebra in High Dimensions v0.6a3 (May, 2011)
 designed and programmed by grant o. passmore {g.o.passmore@sms.ed.ac.uk}
  with intellectual contributions from p.b.jackson, b.boyer, g.collins,
  j.harrison, h.hong, f.kirchner, j moore, l.de moura, s.owre, n.shankar,
  a.tiwari, v.weispfenning and many others.  This version is using Maxima
  multivariate factorisation & SARAG subresultant PRS + Bernstein bases.

RAHD!> vars x y z w k1 k2 k3 k4 k5 k6
Current vars: (X Y Z W K1 K2 K3 K4 K5 K6).

RAHD!> assert x >= 12 /\ x > y /\ (x-y) = 1/2 + w
Formula asserted.

RAHD!> assert x > z /\ x^2 + w^3 < (x-y) + k2
Formula asserted.

RAHD!> assert x > k1 /\ k4 = k5^2 /\ k3 > k2 + 2*k4
Formula asserted.

RAHD!> assert k6 <= k4^2 + x*y + 3*k2
Formula asserted.

RAHD!> set print-model
Prover option print-model set.

RAHD!> check
 sat
 model: [X=12,
         Y=11,
         K5=0,
         K1=11,
         K2=1153/8,
         Z=11,
         K3=1161/8,
         K4=0,
         W=1/2,
         K6=4515/8].
```

```
RAHD: Real Algebra in High Dimensions v0.6a3 (May, 2011)
 designed and programmed by grant o. passmore {g.o.passmore@sms.ed.ac.uk}
  with intellectual contributions from p.b.jackson, b.boyer, g.collins,
  j.harrison, h.hong, f.kirchner, j moore, l.de moura, s.owre, n.shankar,
  a.tiwari, v.weispfenning and many others.  This version is using Maxima
  multivariate factorisation & SARAG subresultant PRS + Bernstein bases.

RAHD!> vars a b c d e
Current vars: (A B C D E).

RAHD!> assert a*d + c*b + b*d <= 0
Formula asserted.

RAHD!> assert b >= 0 /\ c >= 0 /\ d >= 0 /\ e >= 0
Formula asserted.

RAHD!> assert a^2 + a*b - b^2 >= 1
Formula asserted.

RAHD!> assert 2*a + b >= 1
Formula asserted.

RAHD!> assert c^2 + c*d - d^2 + 1 <= 0
Formula asserted.

RAHD!> check
 unsat
RAHD:0!u>
```

This class of experiments begins with the following ∀ **RCF** formula which was sent to us by John Harrison in 2008:

$$\begin{aligned}
&\forall a\ \forall b\ \forall c\ \forall d \\
&((0 \le a) \wedge (a \le 1) \wedge (0 \le b) \wedge (b \le 1) \wedge \\
&(0 \le c) \wedge (c \le 1) \wedge (0 \le d) \wedge (d \le 1)) \\
&\Rightarrow \\
&(((1 - a^2b^2)(1 - cd)(ad - bc)(ad - bc) + \\
&(2ab)(cd - ab)(1 - ab)(c - d)(c - d) + \\
&(a^2b^2 - c^2d^2)(1 - cd)(a - b)(a - b)) \ge 0)
\end{aligned}$$

Harrison couldn't prove it using REAL_SOS

We couldn't prove it using: QEPCAD-B, Redlog/Rlqe, Redlog/Rlcad, Realpaver, ...

Goal: Refute Psi over the Reals

Idea: Maybe /just/ out of reach of CAD

$$\psi = \left[\begin{array}{r} \exists a\ \exists b\ \exists c\ \exists d \\ ((0 \le a) \wedge (a \le 1) \wedge (0 \le b) \wedge (b \le 1) \wedge \\ (0 \le c) \wedge (c \le 1) \wedge (0 \le d) \wedge (d \le 1)) \\ \wedge \\ (((1-a^2b^2)(1-cd)(ad-bc)(ad-bc) + \\ (2ab)(cd-ab)(1-ab)(c-d)(c-d) + \\ (a^2b^2-c^2d^2)(1-cd)(a-b)(a-b)) < 0) \end{array}\right].$$

No obvious way to eliminate variables, but...

Goal: Refute Psi over the Reals

$(0 \le a) \wedge (a \le 1)$

$$\psi = \left[\begin{array}{r} \exists a\ \exists b\ \exists c\ \exists d \\ ((0 \le a) \wedge (a \le 1) \wedge (0 \le b) \wedge (b \le 1) \wedge \\ (0 \le c) \wedge (c \le 1) \wedge (0 \le d) \wedge (d \le 1)) \\ \wedge \\ (((1-a^2b^2)(1-cd)(ad-bc)(ad-bc) + \\ (2ab)(cd-ab)(1-ab)(c-d)(c-d) + \\ (a^2b^2-c^2d^2)(1-cd)(a-b)(a-b)) < 0) \end{array}\right].$$

We can split non-strict inequalities!

Goal: Refute Psi over the Reals

$$\varphi \iff \neg\psi \iff \neg(\psi_{a,=} \vee \psi_{a,<})$$

$$\psi_{a,=} = \left[\begin{array}{r} \exists a\ \exists b\ \exists c\ \exists d \\ ((0=a) \wedge (a \leq 1) \wedge (0 \leq b) \wedge (b \leq 1) \wedge \\ (0 \leq c) \wedge (c \leq 1) \wedge (0 \leq d) \wedge (d \leq 1)) \\ \wedge \\ (((1-a^2b^2)(1-cd)(ad-bc)(ad-bc) + \\ (2ab)(cd-ab)(1-ab)(c-d)(c-d) + \\ (a^2b^2-c^2d^2)(1-cd)(a-b)(a-b)) < 0) \end{array}\right]$$

$$\psi_{a,<} = \left[\begin{array}{r} \exists a\ \exists b\ \exists c\ \exists d \\ ((0<a) \wedge (a \leq 1) \wedge (0 \leq b) \wedge (b \leq 1) \wedge \\ (0 \leq c) \wedge (c \leq 1) \wedge (0 \leq d) \wedge (d \leq 1)) \\ \wedge \\ (((1-a^2b^2)(1-cd)(ad-bc)(ad-bc) + \\ (2ab)(cd-ab)(1-ab)(c-d)(c-d) + \\ (a^2b^2-c^2d^2)(1-cd)(a-b)(a-b)) < 0) \end{array}\right].$$

Goal: Refute Psi over the Reals

$$\Psi^*_{a,=} = \left[\begin{array}{r} \exists b\ \exists c\ \exists d \\ ((0 \le b) \wedge (b \le 1) \wedge \\ (0 \le c) \wedge (c \le 1) \wedge (0 \le d) \wedge (d \le 1)) \\ \wedge \\ (((1-cd)(bc)^2 + \\ (0-c^2d^2)(1-cd)b^2) < 0) \end{array}\right].$$

Now refutable by CAD.

Goal: Refute Psi over the Reals

$$\Psi^*_{a,=} = \left[\begin{array}{r} \exists b\ \exists c\ \exists d \\ ((0 \le b) \wedge (b \le 1) \wedge \\ (0 \le c) \wedge (c \le 1) \wedge (0 \le d) \wedge (d \le 1)) \\ \wedge \\ (((1-cd)(bc)^2 + \\ (0-c^2d^2)(1-cd)b^2) < 0) \end{array}\right].$$

Now refutable by CAD.

What about the other (strict inequality) branch?

$$\Psi_{a,<} = \left[\begin{array}{r} \exists a\ \exists b\ \exists c\ \exists d \\ ((0 < a) \wedge (a \le 1) \wedge (0 \le b) \wedge (b \le 1) \wedge \\ (0 \le c) \wedge (c \le 1) \wedge (0 \le d) \wedge (d \le 1)) \\ \wedge \\ (((1-a^2b^2)(1-cd)(ad-bc)(ad-bc) + \\ (2ab)(cd-ab)(1-ab)(c-d)(c-d) + \\ (a^2b^2-c^2d^2)(1-cd)(a-b)(a-b)) < 0) \end{array}\right].$$

Now it's closer to a use of McCallum's Theorem!

Goal: Refute Psi over the Reals

$$\Psi^*_{a,=} = \left[\begin{array}{r} \exists b\ \exists c\ \exists d \\ ((0 \le b) \wedge (b \le 1) \wedge \\ (0 \le c) \wedge (c \le 1) \wedge (0 \le d) \wedge (d \le 1)) \\ \wedge \\ (((1-cd)(bc)^2 + \\ (0-c^2d^2)(1-cd)b^2) \ < \end{array}\right.$$

Now refutable by CAD.

Let's try this!

What about inequality) branch?

$$\Psi_{a,<} = \left[\begin{array}{r} \exists a\ \exists b\ \exists c\ \exists a \\ ((0 < a) \wedge (a \le 1) \wedge (0 \le b) \wedge (b \le 1) \wedge \\ (0 \le c) \wedge (c \le 1) \wedge (0 \le d) \wedge (d \le 1)) \\ \wedge \\ (((1-a^2b^2)(1-cd)(ad-bc)(ad-bc) + \\ (2ab)(cd-ab)(1-ab)(c-d)(c-d) + \\ (a^2b^2-c^2d^2)(1-cd)(a-b)(a-b)) \ < \ 0) \end{array}\right]$$

Now it's closer to a use of McCallum's Theorem!

Can explore strategy interactively...

```
RAHD:0.0!> e1 [when (cid = 0) [demod-lin; run stable-simp]]
RAHD:0.0!> pc 0

 Printing case 0 for goal 0.0.

 -------     ------------------------------------------------------------------------
 case-id     case
 -------     ------------------------------------------------------------------------

       0     ((<= 0 B) (<= B 1) (<= 0 C) (<= C 1) (<= 0 D) (<= D 1)
              (<
               (+ (* (- 1 (* C D)) (* (* B C) (* B C)))
                  (* (* (- 0 (* (* C C) (* D D))) (- 1 (* C D))) (* B B)))
               0))    (UNKNOWN SIMP-ARITH SIMP-GLS DEMOD-LIN)


     2 cases in goalset (goal 0.0) awaiting refutation.

RAHD:0.0!> e1 [when (cid = 0) [qepcad]]
RAHD:0.0!> opens

 Printing all of the open 1 cases for goal 0.0.

 -------     ------------------------------------------------------------------------
 case-id     case
 -------     ------------------------------------------------------------------------

       1     ((<= A 1) (<= 0 B) (<= B 1) (<= 0 C) (<= C 1) (<= 0 D) (<= D 1)
              (<
               (+
                (+
                 (* (* (- 1 (* (* A A) (* B B))) (- 1 (* C D)))
                    (* (- (* A D) (* B C)) (- (* A D) (* B C))))
                 (* (* (* (* (* 2 A) B) (- (* C D) (* A B))) (- 1 (* A B)))
                    (* (- C D) (- C D))))
                (*
                 (* (- (* (* A A) (* B B)) (* (* C C) (* D D))) (- 1 (* C D)))
                 (* (- A B) (- A B))))
               0)
              (< 0 A))    (UNKNOWN)


     1 case in goalset (goal 0.0) awaiting refutation.

RAHD:0.0!>
```

```
RAHD:0.0!> e1 [split-ineqs(atom := 0)]
RAHD:0.0!> cguc
RAHD:0.0.1!> e1 [when (cid = 0) [demod-lin; run stable-simp; qepcad]]
RAHD:0.0.1!> e1 [split-ineqs(atom := 0)]
RAHD:0.0.1!> cguc
RAHD:0.0.1.1!> e1 [when (cid = 0) [demod-lin; run stable-simp; qepcad]]
RAHD:0.0.1.1!> e1 [split-ineqs(atom := 0)]
RAHD:0.0.1.1!> cguc
RAHD:0.0.1.1.1!> e1 [when (cid = 0) [demod-lin; run stable-simp; qepcad]]
RAHD:0.0.1.1.1!> e1 [split-ineqs(atom := 0)]
RAHD:0.0.1.1.1!> cguc
RAHD:0.0.1.1.1.1!> e1 [when (cid = 0) [demod-lin; run stable-simp; qepcad]]
RAHD:0.0.1.1.1.1!> e1 [split-ineqs(atom := 0)]
RAHD:0.0.1.1.1.1!> cguc
RAHD:0.0.1.1.1.1.1!> e1 [when (cid = 0) [demod-lin; run stable-simp; qepcad]]
RAHD:0.0.1.1.1.1.1!> e1 [split-ineqs(atom := 0)]
RAHD:0.0.1.1.1.1.1!> cguc
RAHD:0.0.1.1.1.1.1.1!> e1 [when (cid = 0) [demod-lin; run stable-simp; qepcad(open? := 1)]]
RAHD:0.0.1.1.1.1.1.1!> e1 [split-ineqs(atom := 0)]
RAHD:0.0.1.1.1.1.1.1!> cguc
RAHD:0.0.1.1.1.1.1.1.1!> e1 [when (cid = 0) [demod-lin; run stable-simp; qepcad(open? := 1)]]
RAHD:0.0.1.1.1.1.1.1.1!> e1 [qepcad(open? := 1)]
RAHD:0.0.1.1.1.1.1.1.1!u> up
Trickled refutation up to Case 1 of Goal 0.0.1.1.1.1.1.

RAHD:0.0.1.1.1.1.1.1!u> up
Trickled refutation up to Case 1 of Goal 0.0.1.1.1.1.1.

RAHD:0.0.1.1.1.1.1!u> up
Trickled refutation up to Case 1 of Goal 0.0.1.1.1.1.

RAHD:0.0.1.1.1.1!u> up
Trickled refutation up to Case 1 of Goal 0.0.1.1.1.

RAHD:0.0.1.1.1!u> up
Trickled refutation up to Case 1 of Goal 0.0.1.1.

RAHD:0.0.1.1!u> up
Trickled refutation up to Case 1 of Goal 0.0.1.1.

RAHD:0.0.1!u> up
Trickled refutation up to Case 1 of Goal 0.0.1.

RAHD:0.0.1!u> up
Trickled refutation up to Case 1 of Goal 0.0.

RAHD:0.0!u> up
Trickled refutation up to Case 0 of Goal 0.

RAHD:0!u> status

 Goalkey: 0,
 Unknown cases: 0 of 1.

 Decision: unsat.
```

Can explore strategy interactively...

```
RAHD:0.0!> e1 [when (cid = 0) [demod-lin; run stable-simp]]
RAHD:0.0!> pc 0

 Printing case 0 for goal 0.0.

 -------     ------------------------------------------------------------------
 case-id     case
 -------     ------------------------------------------------------------------

       0     ((<= 0 B) (<= B 1) (<= 0 C) (<= C 1) (<= 0 D) (<= D 1)
              (<
               (+ (* (- 1 (* C D)) (* (* B C) (* B C)))
                  (* (* (- 0 (* (* C C) (* D D))) (- 1 (* C D))) (*
               0))    (UNKNOWN SIMP-ARITH SIMP-GLS DEMOD-LIN)


     2 cases in goalset (goal 0.0) awaiting refutation.

RAHD:0.0!> e1 [when (cid = 0) [qepcad]]
RAHD:0.0!> opens

 Printing all of the open 1 cases for goal 0.0.

 -------     ------------------------------------------------------------------
 case-id     case
 -------     ------------------------------------------------------------------

       1     ((<= A 1) (<= 0 B) (<= B 1) (<= 0 C) (<= C 1) (<= 0 D) (<= D 1)
              (<
               (+
                (+
                 (* (* (- 1 (* (* A A) (* B B))) (- 1 (* C D)))
                    (* (- (* A D) (* B C)) (- (* A D) (* B C))))
                 (* (* (* (* (* 2 A) B) (- (* C D) (* A B))) (- 1 (* A B)))
                    (* (- C D) (- C D))))
                (*
                 (* (- (* (* A A) (* B B)) (* (* C C) (* D D))) (- 1 (* C D)))
                 (* (- A B) (- A B))))
               0)
              (< 0 A))    (UNKNOWN)


     1 case in goalset (goal 0.0) awaiting refutation.

RAHD:0.0!>
```

Let's automate this!

```
RAHD:0.0!> e1 [split-ineqs(atom := 0)]
RAHD:0.0!> cguc
RAHD:0.0.1!> e1 [when (cid = 0) [demod-lin; run stable-simp; qepcad]]
RAHD:0.0.1!> e1 [split-ineqs(atom := 0)]
RAHD:0.0.1!> cguc
RAHD:0.0.1.1!> e1 [when (cid = 0) [demod-lin; run stable-simp; qepcad]]
RAHD:0.0.1.1!> e1 [split-ineqs(atom := 0)]
RAHD:0.0.1.1!> cguc
RAHD:0.0.1.1.1!> e1 [when (cid = 0) [demod-lin; run stable-simp; qepcad]]
RAHD:0.0.1.1.1!> e1 [split-ineqs(atom := 0)]
RAHD:0.0.1.1.1!> cguc
RAHD:0.0.1.1.1.1!> e1 [when (cid = 0) [demod-lin; run stable-simp; qepcad]]
RAHD:0.0.1.1.1.1!> e1 [split-ineqs(atom := 0)]
RAHD:0.0.1.1.1.1!> cguc
RAHD:0.0.1.1.1.1.1!> e1 [when (cid = 0) [demod-lin; run stable-simp; qepcad(open? := 1)]]
RAHD:0.0.1.1.1.1.1!> e1 [qepcad(open? := 1)]
RAHD:0.0.1.1.1.1.1!u> up
Trickled refutation up to Case 1 of Goal 0.0.1.1.1.1.1.1.

RAHD:0.0.1.1.1.1.1!u> up
Trickled refutation up to Case 1 of Goal 0.0.1.1.1.1.1.

RAHD:0.0.1.1.1.1.1!u> up
Trickled refutation up to Case 1 of Goal 0.0.1.1.1.1.

RAHD:0.0.1.1.1.1.1!u> up
Trickled refutation up to Case 1 of Goal 0.0.1.1.1.

RAHD:0.0.1.1.1.1!u> up
Trickled refutation up to Case 1 of Goal 0.0.1.1.

RAHD:0.0.1.1.1!u> up
Trickled refutation up to Case 1 of Goal 0.0.1.

RAHD:0.0.1.1!u> up
Trickled refutation up to Case 1 of Goal 0.0.

RAHD:0.0!u> up
Trickled refutation up to Case 0 of Goal 0.

RAHD:0!u> status

 Goalkey: 0,
 Unknown cases: 0 of 1.

 Decision: unsat.
```

Can explore strategy interactively...

```
RAHD!> opens

 Printing all of the open 1 cases for goal 0.

 -------     ------------------------------------------------------------------
 case-id     case
 -------     ------------------------------------------------------------------

       0     ((<= 0 A) (<= A 1) (<= 0 B) (<= B 1) (<= 0 C) (<= C 1) (<= 0 D)
              (<= D 1)
              (<
               (+
                (+
                 (* (* (- 1 (* (* A A) (* B B))) (- 1 (* C D)))
                    (* (- (* A D) (* B C)) (- (* A D) (* B C))))
                 (* (* (* (* (* 2 A) B) (- (* C D) (* A B))) (- 1 (* A B)))
                    (* (- C D) (- C D))))
                (*
                 (* (- (* (* A A) (* B B)) (* (* C C) (* D D))) (- 1 (* C D)))
                 (* (- A B) (- A B))))
               0))    (UNKNOWN)


      1 case in goalset (goal 0) awaiting refutation.

RAHD!> e [demod-lin; run stable-simp;
           [if (dim <= 3) [qepcad]
              [split-ineqs(atom := 0); qepcad(open? := 1)]]]
RAHD:0!u> status

 Goalkey: 0,
 Unknown cases: 0 of 1.

 Decision: unsat.
```

Can explore strategy interactively...

```
RAHD!> opens

 Printing all of the open 1 cases for goal 0.

 -------     ------------------------------------------------------------
 case-id     case
 -------     ------------------------------------------------------------

       0     ((<= 0 A) (<= A 1) (<= 0 B) (<= B 1) (<= 0 C) (<= C 1) (<= 0 D)
```

Moral of the Story:

Interactive proof exploration and heuristic strategies are not just for undecidable domains; we need them in expensive decision procedures, too!

```
RAHD:0!u> status

 Goalkey: 0,
 Unknown cases: 0 of 1.

 Decision: unsat.
```

RAHD Strategies in Action

```
defstrat calculemus-0
  [split-ineqs(max-splits := 12); simp-zrhs; run stable-simp; demod-lin;
   run stable-simp; simp-real-null; fert-tsos; univ-sturm-ineqs;
   satur-lin; triv-ideals; run stable-simp; rcr-ineqs; run stable-simp;
   fert-tsos; run stable-simp; simp-zrhs; int-dom-zpb; rcr-ineqs;
   qepcad(open? := 1); qepcad].
```

```
defstrat calculemus-1
  [[when (gd = 0) [split-ineqs(max-splits := 12)]];
   interval-cp(max-contractions := 10); simp-zrhs; run stable-simp;
   demod-lin; run stable-simp; simp-real-null; fert-tsos; univ-sturm-ineqs;
   satur-lin; interval-cp; triv-ideals; run stable-simp; interval-cp;
   rcr-ineqs; run stable-simp; fert-tsos; run stable-simp; interval-cp;
   simp-zrhs; interval-cp; int-dom-zpb; rcr-ineqs;
   when (dim <= 7 /\ deg <= 30) [qepcad(open? := 1); qepcad]].
```

```
defstrat calculemus-2
  [interval-cp(max-contractions := 10);
   [when (dim <= 3 /\ deg <= 3) [qepcad]];
   [when (gd = 0 /\ dim >= 2) [split-ineqs(max-splits := 12)]];
   interval-cp(max-contractions := 20); simp-zrhs; run stable-simp;
   demod-lin; run stable-simp; simp-real-null; fert-tsos; univ-sturm-ineqs;
   satur-lin; interval-cp; triv-ideals; run stable-simp; interval-cp;
   rcr-ineqs; run stable-simp; fert-tsos; run stable-simp; interval-cp;
   simp-zrhs; interval-cp; int-dom-zpb; rcr-ineqs;
   when (dim <= 7 /\ deg <= 30) [qepcad(open? := 1); qepcad]].
```

			calc-0	calc-1	calc-2	qepcad-b	redlog/rlqe	redlog/rlcad
benchmark	dimension	degree	time (s)	time (s)	time (s)	time (s)	time (s)	time (s)
P0	5	4	0.9	1.6	1.7	416.4	40.4	>600.0
P1	6	4	1.7	3.1	3.4	>600.0	>600.0	>600.0
P2	5	4	1.3	2.4	2.6	>600.0	>600.0	>600.0
P3	5	4	1.5	2.5	2.7	>600.0	>600.0	>600.0
P4	5	4	1.1	2.0	2.7	>600.0	>600.0	>600.0
P5	14	2	0.3	0.3	0.3	>600.0	97.4	>600.0
P6	11	5	147.4	<0.1	<0.1	>600.0	<0.1	<0.1
P7	8	2	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1
P8	7	32	4.5	0.1	<0.1	8.4	<0.1	>600.0
P9	7	16	4.5	0.2	<0.1	0.3	<0.1	6.7
P10	7	12	100.7	20.8	8.9	>600.0	>600.0	>600.0
P11	6	2	1.6	0.5	0.5	<0.1	<0.1	<0.1
P12	5	3	0.8	0.3	0.4	<0.1	<0.1	<0.1
P13	4	10	3.8	3.9	4.0	>600.0	>600.0	>600.0
P14	2	2	4.5	1.7	<0.1	<0.1	>600.0	>600.0
P15	4	3	0.2	0.2	0.1	<0.1	<0.1	<0.1
P16	4	2	10.0	2.2	2.1	<0.1	<0.1	<0.1
P17	4	2	0.6	0.6	0.7	0.3	<0.1	0.6
P18	4	2	1.3	1.3	1.3	<0.1	<0.1	<0.1
P19	3	6	3.3	1.7	2.1	<0.1	<0.1	0.7
P20	3	4	1.2	0.7	0.7	<0.1	<0.1	0.3
P21	3	2	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1
P22	2	4	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1
P23	2	2	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1

P10	calc-0	calc-1	calc-2
#(Proof-tree)	32768	8192	8192
TRIV-IDEALS	278 (45.847)	96 (0.936)	21 (0.101)
SATUR-LIN	485 (1.465)	344 (0.756)	25 (0.275)
FERT-TSOS	70 (0.355)	48 (0.200)	8 (0.041)
DEMOD-LIN	1101 (0.238)	444 (0.048)	33 (0.003)
SIMP-GLS	34351 (1.290)	3384 (0.185)	33 (0.000)
SIMP-ZRHS	32919 (0.370)	3133 (0.031)	33 (0.000)
SPLIT-INEQS (MAX-SPLITS:=12)	8194 (0.468)	2 (0.109)	2 (0.131)
QEPCAD (OPEN?:=1)	207 (21.081)	70 (10.881)	-
RCR-INEQS	180 (7.601)	70 (0.225)	-
INT-DOM-ZPB	0 (0.012)	0 (0.006)	-
INTERVAL-CP (MAX-CONTRACTIONS:=10)	-	5120 (6.052)	5120 (6.092)
INTERVAL-CP (MAX-CONTRACTIONS:=20)	-	-	3039 (2.126)
DEMOD-NUM	32766 (3.318)	3070 (0.170)	33 (0.003)
SIMP-ARITH	1954 (0.430)	482 (0.143)	33 (0.006)
SIMP-REAL-NULL	142 (0.031)	0 (0.032)	0 (0.002)
UNIV-STURM-INEQS	4 (0.035)	0 (0.016)	0 (0.001)
INTERVAL-CP	-	178 (0.834)	4 (0.024)
	100.850	21.049	9.343

MetiTarski: an automatic theorem prover coupled with RCF decision procedures

- *Objective*: to prove first-order statements involving real-valued functions such as exp, ln, sin, cos, $\tan^{-1}$, …
- *Method*: **resolution** theorem proving augmented with
  - **axioms** bounding these functions by rational functions
  - **heuristics** to isolate function occurrences and create RCF problems
  - … to be solved using QE tools: QEPCAD, Mathematica, Z3, etc.

the basic idea

Our approach involves replacing functions by *rational function upper or lower bounds*.

First-order formulae involving +, −, × and ≤ (on reals) are decidable.

We end up with polynomial inequalities in a decidable theory: *real closed fields (RCF)*.

Real QE and *resolution theorem proving* are the core technologies.

Some MetiTarski Theorems

$$0 < t \wedge 0 < v_f \Longrightarrow ((1.565 + .313v_f)\cos(1.16t) + (.01340 + .00268v_f)\sin(1.16t))e^{-1.34t} - (6.55 + 1.31v_f)e^{-.318t} + v_f + 10 \geq 0$$

$$0 \leq x \wedge x \leq 1.46 \times 10^{-6} \Longrightarrow (64.42\sin(1.71 \times 10^6 x) - 21.08\cos(1.71 \times 10^6 x))e^{9.05\times10^5 x} + 24.24e^{-1.86\times10^6 x} > 0$$

$$0 \leq x \wedge 0 \leq y \Longrightarrow y\tanh(x) \leq \sinh(yx)$$

Each is proved in a few seconds!

some bounds for ln

- based on the continued fraction for ln(x+1)
- *much* more accurate than the Taylor expansion
- Simplicity can be exchanged for accuracy.
- With these, the maximum degree we use is 8.

$$\frac{x-1}{x} \leq \ln x \leq x-1$$

$$\frac{(1+5x)(x-1)}{2x(2+x)} \leq \ln x \leq \frac{(x+5)(x-1)}{2(2x+1)}$$

bounds for other functions

- a mix of *continued fraction* approximants and truncated *Taylor series*, etc, modified to suit various argument ranges and accuracies

- a tiny bit of **built-in knowledge** about signs, for example, $\exp(x) > 0$

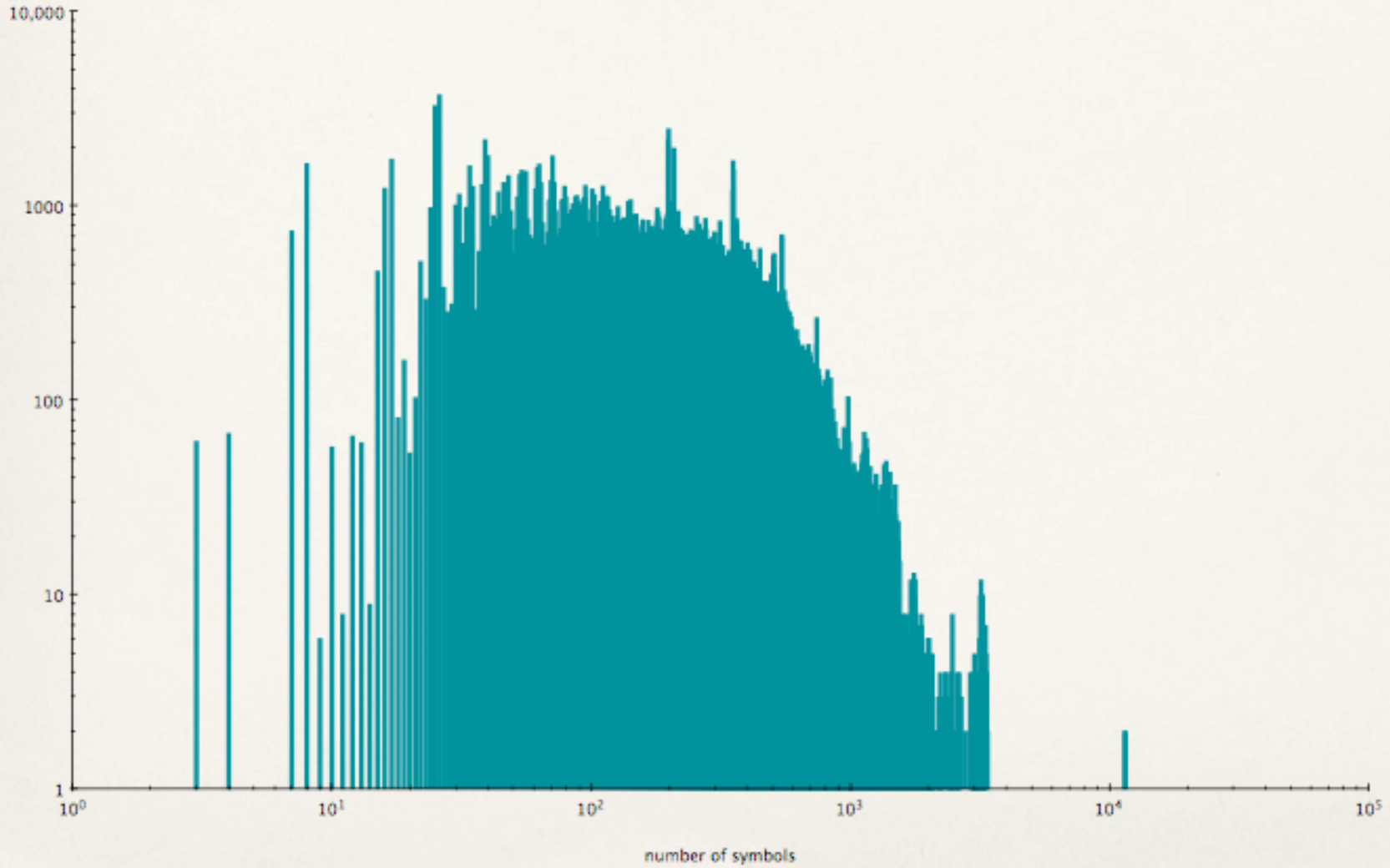
- NO fundamental mathematical knowledge, for example, the geometric interpretation of trigonometric functions

- MetiTarski can reason about any function that has well-behaved *upper and lower bounds* as rational functions.

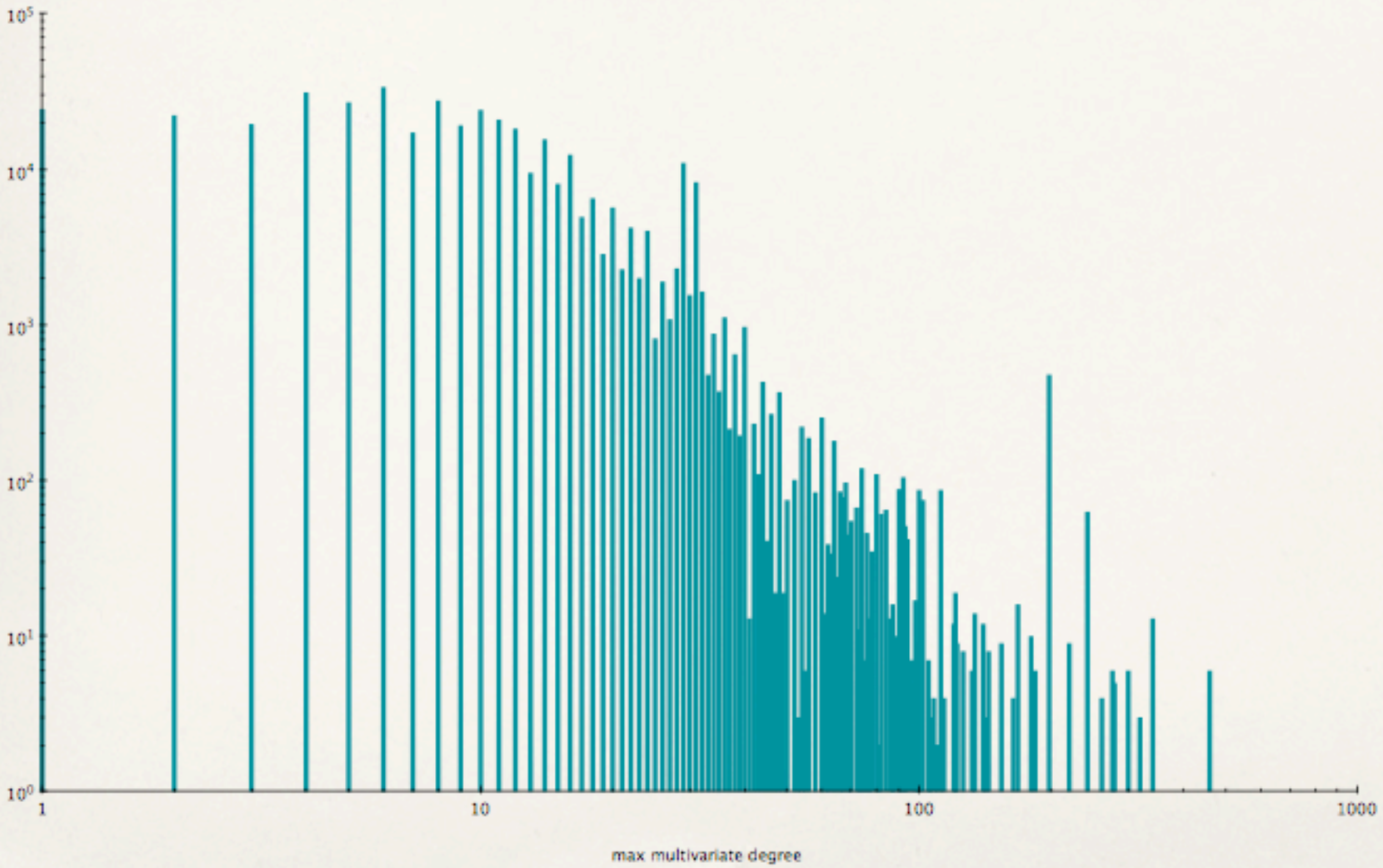
statistics about the RCF problems

- 400,000 RCF problems generated from 859 MetiTarski problems.
- Number of *symbols*: in some cases, 11,000 or more!
- Maximum *degree*: up to 460!
- But… number of *variables*? Typically just 1. No more than 8.

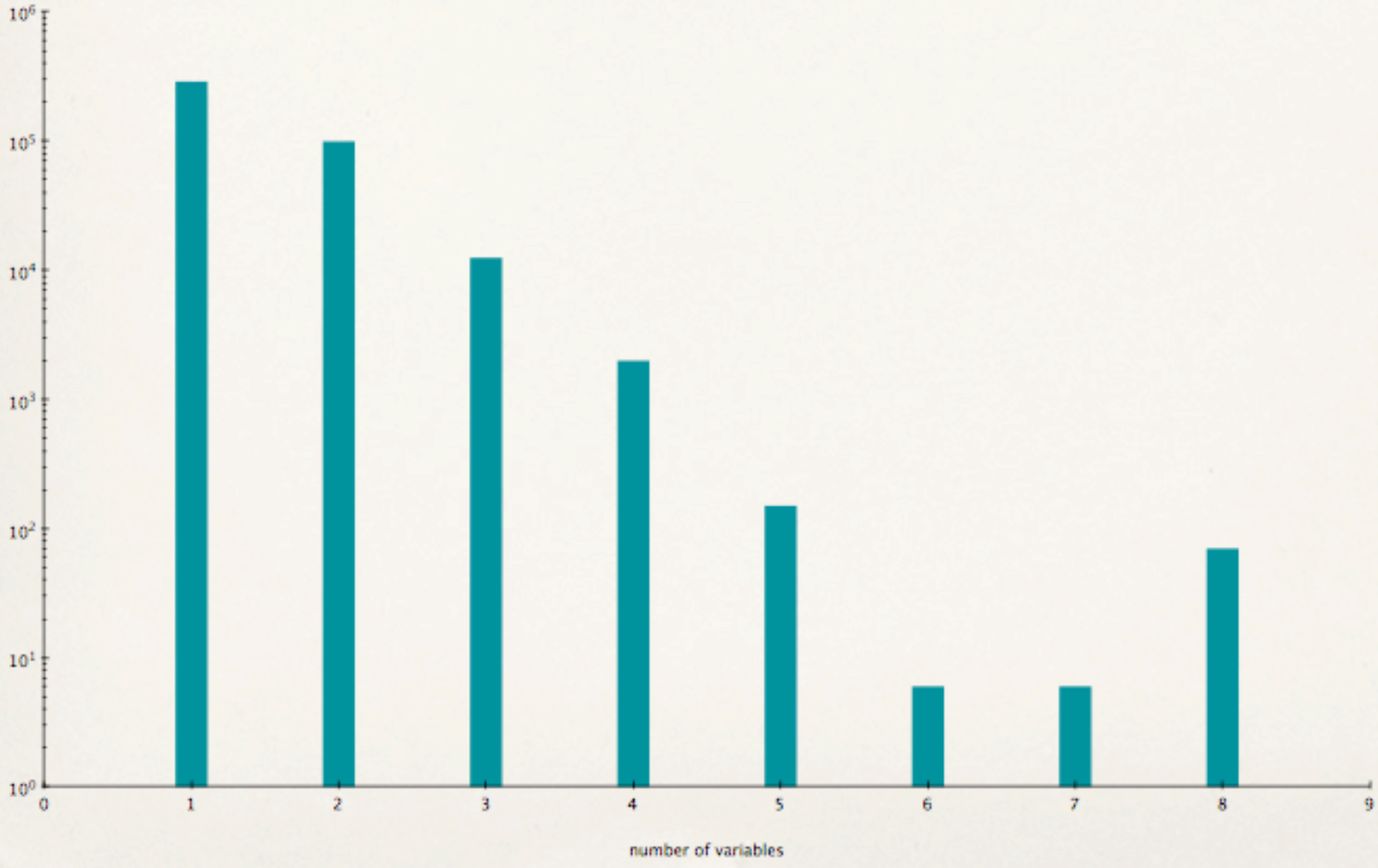
distribution of problem sizes (in symbols)



distribution of polynomial degrees (multivariate)



distribution of problem dimensions



a heuristic: *model sharing*

- MetiTarski applies QE only to existential formulas, $\exists x\ \exists y\ \ldots$
- Many of these turn out to be satisfiable,…
- and many satisfiable formulas have the *same model*.
- By maintaining a list of "successful" models, we can show many RCF formulas to be satisfiable **without performing QE**.

… because most of our RCF problems are satisfiable…

Problem	All RCF		SAT RCF		% SAT	
	#	secs	#	secs	#	secs
CONVOI2-sincos	268	3.28	194	2.58	72%	79%
exp-problem-9	1213	6.25	731	4.11	60%	66%
log-fun-ineq-e-weak	496	31.50	323	20.60	65%	65%
max-sin-2	2776	253.33	2,221	185.28	80%	73%
sin-3425b	118	39.28	72	14.71	61%	37%
sqrt-problem-13-sqrt3	2031	22.90	1403	17.09	69%	75%
tan-1-1var-weak	817	19.5	458	7.60	56%	39%
trig-squared3	742	32.92	549	20.66	74%	63%
trig-squared4	847	45.29	637	20.78	75%	46%
trigpoly-3514-2	1070	17.66	934	14.85	87%	84%

In one example, 2172 of 2221 satisfiable RCF problems can be settled using model sharing, with only 37 separate models.

Also, most polynomials are irreducible!

Problem	# Factor	# Irreducible	% Runtime
asin-8-sqrt2	7791	5975 (76.7%)	22.4%
atan-problem-2-sqrt-weakest21	65304	63522 (97.3%)	55.4%
atan-problem-2-weakest21	9882	8552 (86.5%)	2.2%
cbrt-problem-5a	88986	61068 (68.6%)	38.6%
cbrt-problem-5b-weak	138861	25107 (18.0%)	53.1%
cos-3411-a-weak	150354	138592 (92.1%)	53.9%
ellipse-check-2-weak2	5236	3740 (71.4%)	88.7%
ellipse-check-3-ln	1724	1284 (74.4%)	86.7%
ellipse-check-3-weak	12722	9464 (74.3%)	77.9%

In one example, 2172 of 2221 satisfiable RCF problems can be settled using model sharing, with only 37 separate models.

introducing Strategy 1

model sharing

+

omitting the standard test for *irreducibility*

= Strategy 1

introducing Strategy 1

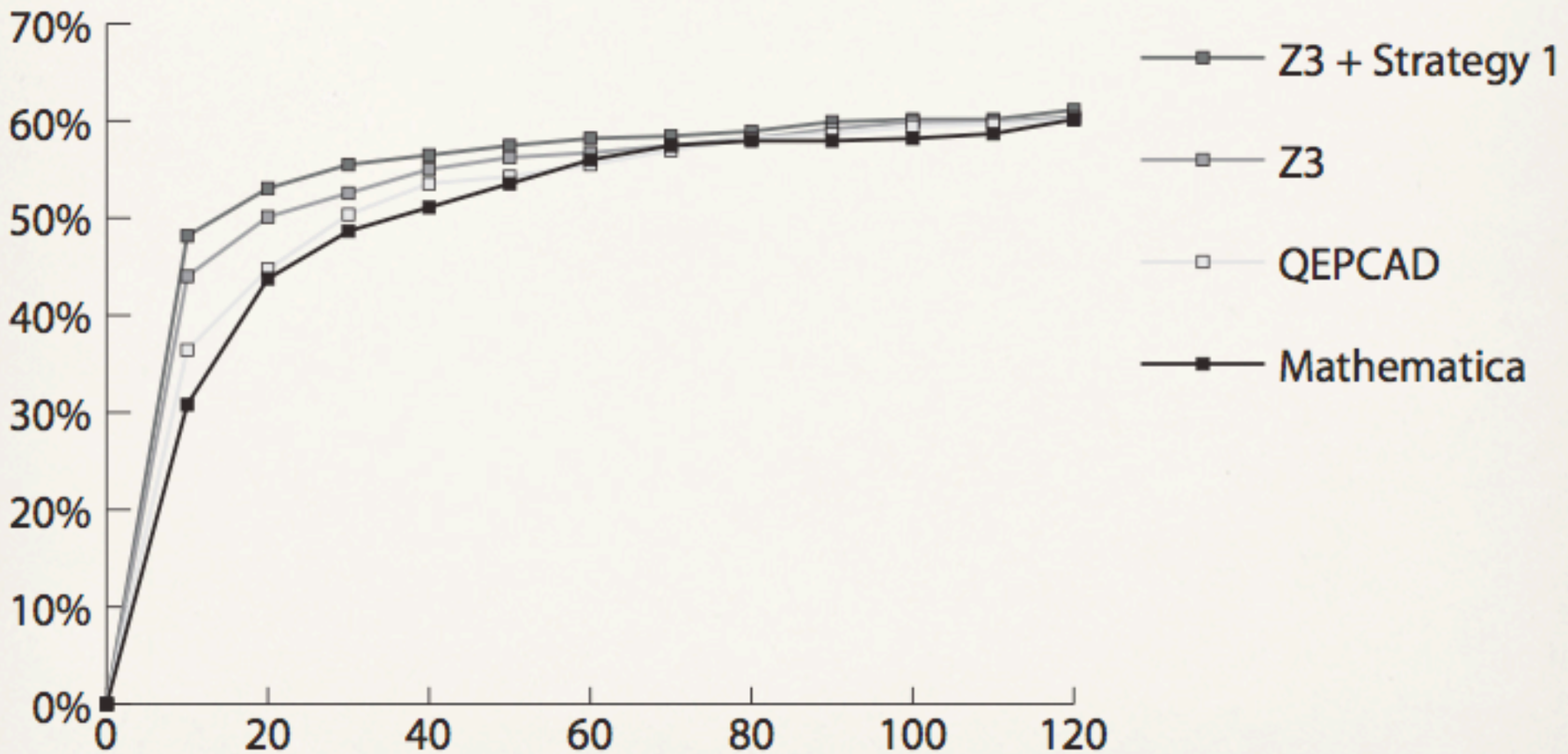
model sharing

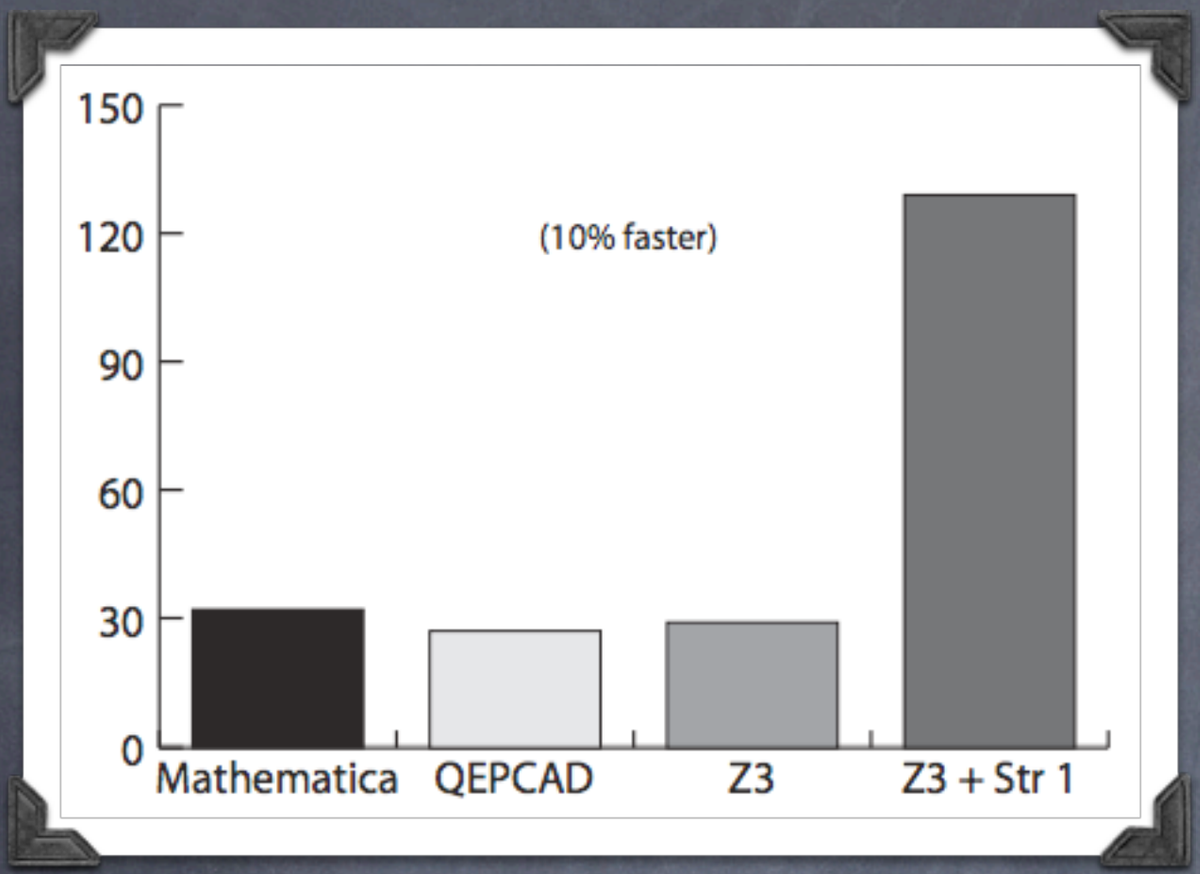
+

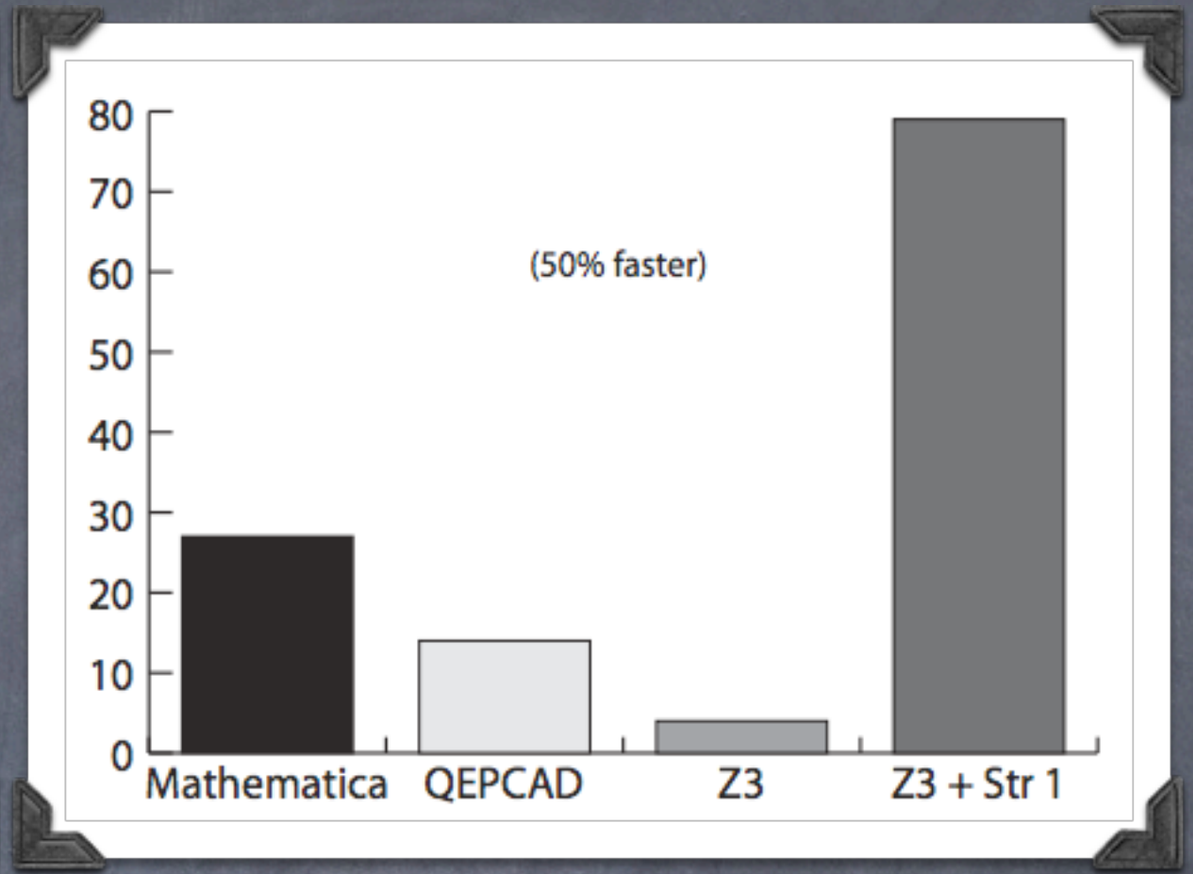
```
(and-then simplify purify-arith
          propagate-values elim-term-ite
          solve-eqs tseitin-cnf simplify
          (using-params nlsat
                        :factor false
                        :algebraic-min-mag 256))
```

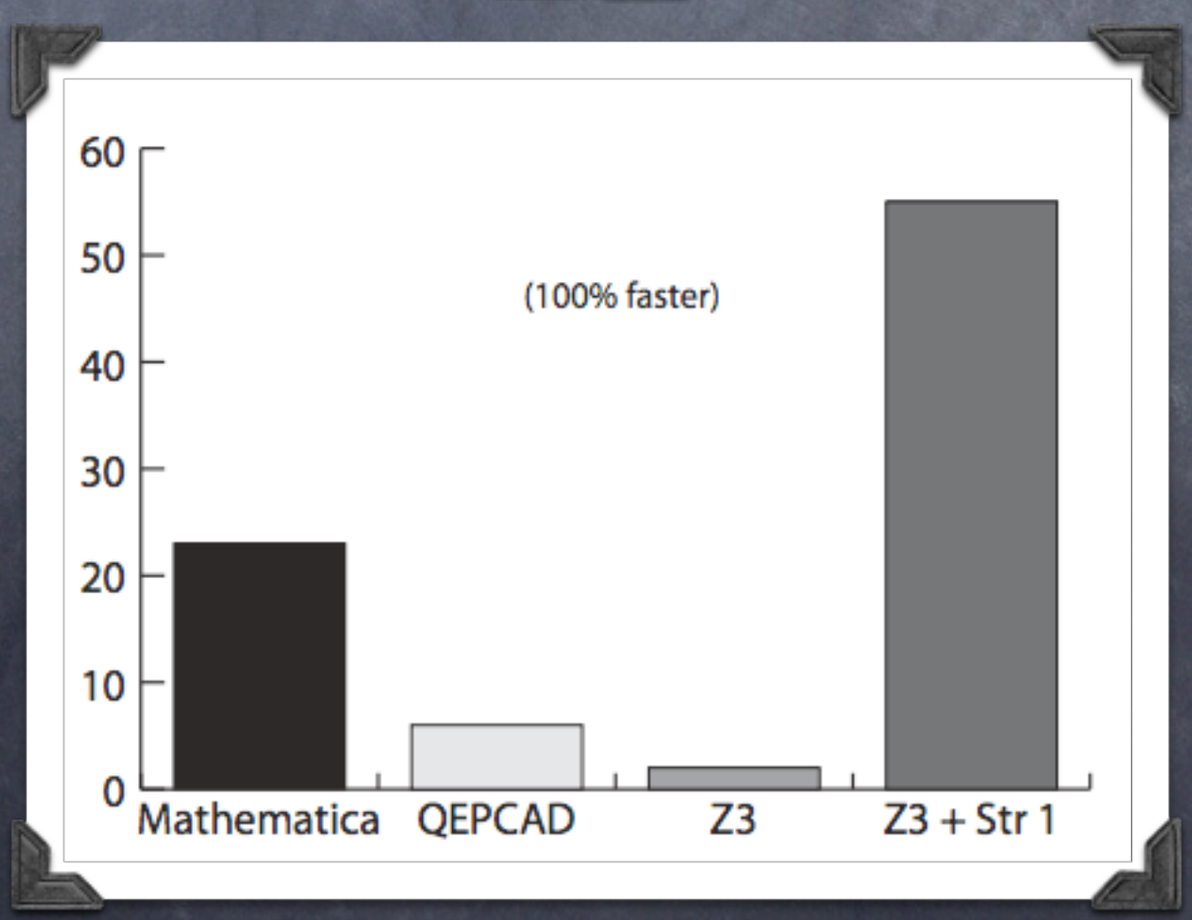
= Strategy 1

comparative results
(% proved in up to 120 secs)




150
120
90
60
30
0
(10% faster)
Mathematica
QEPCAD
Z3
Z3 + Str 1


80
70
60
50
40
30
20
10
0
(50% faster)
Mathematica
QEPCAD
Z3
Z3 + Str 1


60
50
40
30
20
10
0
(100% faster)
Mathematica
QEPCAD
Z3
Z3 + Str 1

Z3 Strategy Interfaces: C++ API, Z3Py, ...

load in editor

```
x, y, z = Reals('x y z')
g = Goal()
g.add(Or(x == 0, x == 1),
      Or(y == 0, y == 1),
      Or(z == 0, z == 1),
      x + y + z > 2)

# Split all clauses"
split_all = Repeat(OrElse(Tactic('split-clause'),
                          Tactic('skip')))
print split_all(g)

split_at_most_2 = Repeat(OrElse(Tactic('split-clause'),
                          Tactic('skip')),
                         1)
print split_at_most_2(g)

# Split all clauses and solve equations
split_solve = Then(Repeat(OrElse(Tactic('split-clause'),
                                 Tactic('skip'))),
                   Tactic('solve-eqs'))

print split_solve(g)
```

In closing

We have illustrated that not only is a strategy-language based approach practical in the context of high-performance solvers, it is also desirable.

In difficult (i.e., infeasible or undecidable) theorem proving domains, the situation with heuristic proof strategies is rarely ``one size fits all."

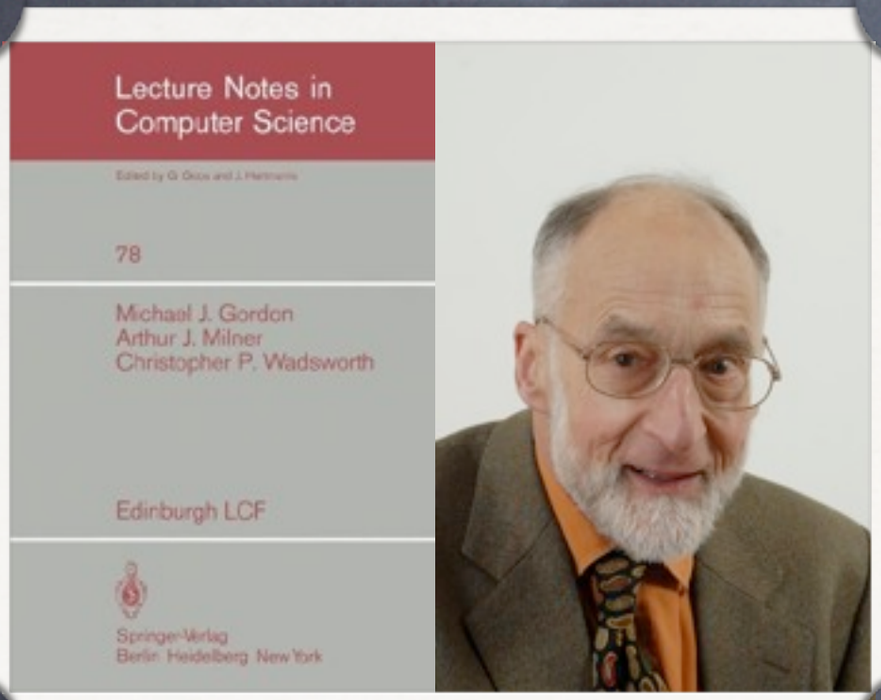
Instead, given a class of problems to solve, it is often the case that one heuristic combination of reasoning engines is far more suited to the task than another.

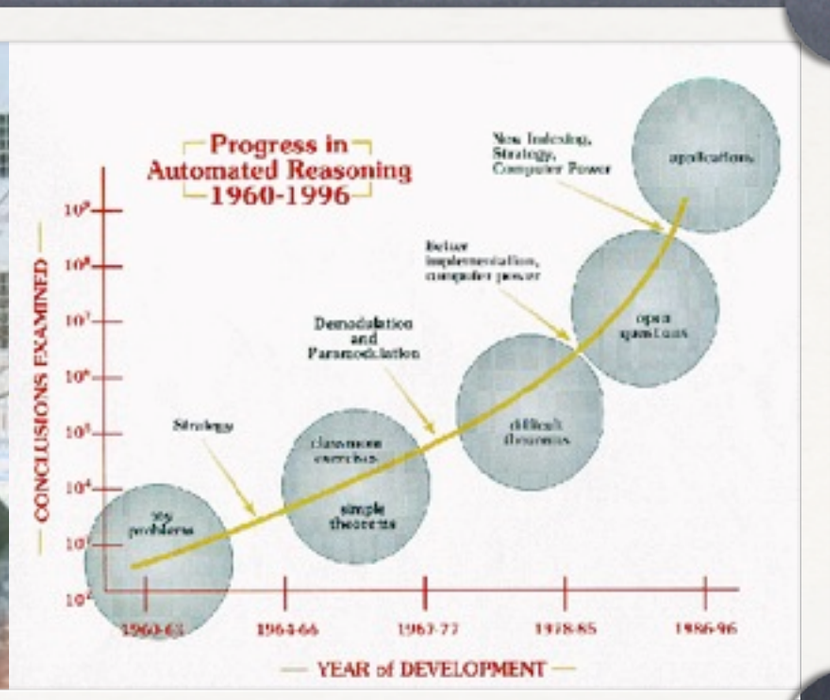
SMT solver developers cannot anticipate all classes of problems end-users will wish to analyze.

By virtue of this, heuristic components of high-performance solvers will never be sufficient in general when they are beyond end-users' control.

Without providing end-users mechanisms to control and modify the heuristic components of their solvers, solver developers are inhibiting their chances of success.

Context and a Story





Prologue: Bill McCune

1953-2011

Solution of the Robbins Problem

WILLIAM McCUNE*
Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439.

Submitted to *J. Automated Reasoning*, Jan. 22, 1997.

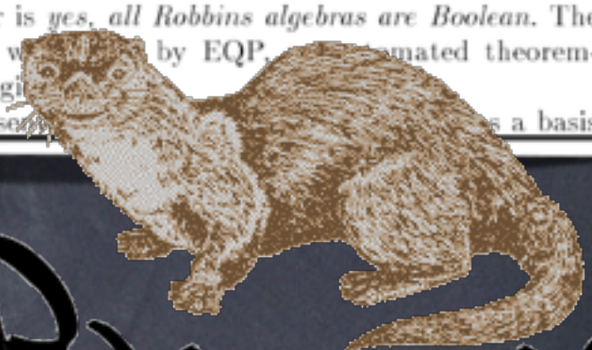
Abstract. In this article we show that the three equations known as commutativity, associativity, and the Robbins equation are a basis for the variety of Boolean algebras. The problem was posed by Herbert Robbins in the 1930s. The proof was found automatically by EQP, a theorem-proving program for equational logic. We present the proof and the search strategies that enabled the program to find the proof.

Key words: Associative-commutative unification, Boolean algebra, EQP, equational logic, paramodulation, Robbins algebra, Robbins problem.

1. Introduction

This article contains the answer to the Robbins question of whether all Robbins algebras are Boolean. The answer is *yes, all Robbins algebras are Boolean.* The proof that answers the question w[illegible] by EQP, [illegible]omated theorem-proving program for equational logi[illegible]

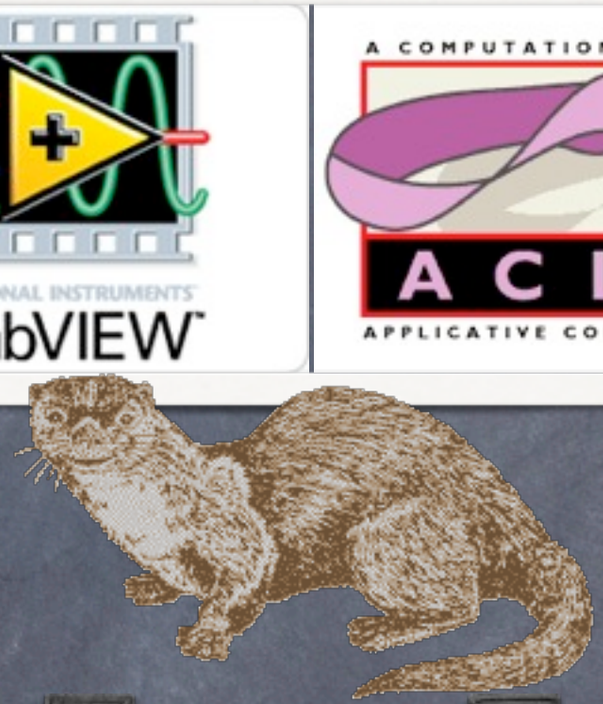
In 1933, E. V. Huntington prese[illegible]s a basis

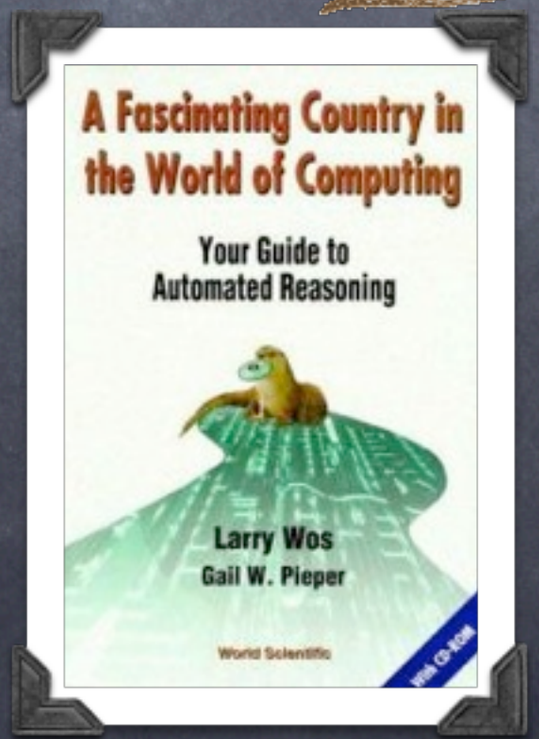


EQP Prover9

Prologue: Bill McCune

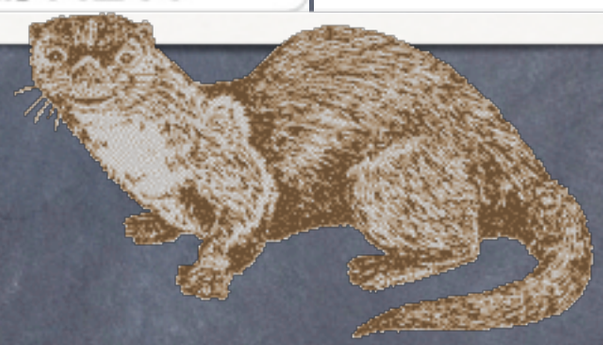
2004



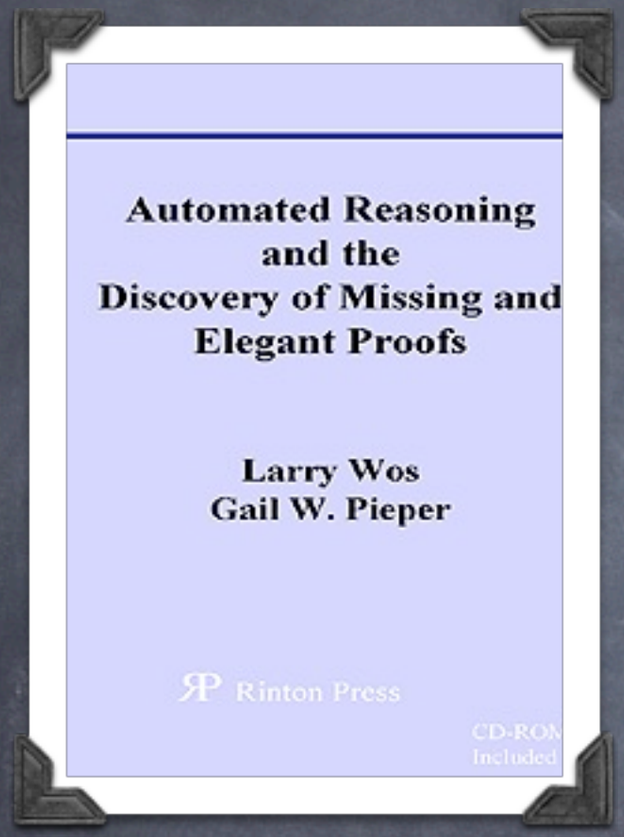


Prologue: Bill McCune

2004



Prologue: Bill McCune



Harrison
Rudnicki Boyer Waldinger Wachter
Uribe
Dahn Lusk Jech McCune
Overbeek
Staples Walsh Jackson Kunen Kapur Hagiya
Holmes
Trybulec Beeson Murthy Thayer Hart Pase