Guessing Game

I'm thinking of a research area where…

- Algorithms have recently improved by orders of magnitude
- Computers solve tasks better than humans
- Computers solve tasks without help from humans
- Big investments are being made by the government and companies like:
  Amazon, Apple, Facebook, Google, Intel, Microsoft
- It can be described using two letters; first one is A

```
(assert (forall ((lambda Real)) (let ((v_17 (+ x4 (* 60 lambda))) (v_11 (not bool.b19)) (v_10 (not bool.b18)) (v_6 (not bool.b17)) (v_8 (not bool.b21)) (v_3 (not bool.b23)) (v_
9 (not bool.b20)) (v_7 (not bool.b22))) (let ((v_4 (and v_9 v_7))) (let ((v_2 (not v_4)) (v_13 (and v_10 v_11))) (let ((v_12 (not v_13)) (v_14 (and v_9 v_13))) (let ((v_15 (and
 v_8 v_14))) (let ((v_16 (and v_7 v_15)) (v_40 (<= (* 1 v_17) 4820))) (let ((v_72 (not v_40)) (v_99 (not bool.b24)) (v_127 (+ x3 (* (/ (- 1) 20) lambda)))) (let ((v_1 (* 1 v_12
7))) (let ((v_0 (+ v_1 (* (/ 1 1200) v_17)))) (let ((v_96 (<= v_0 (/ 20 3))) (v_5 (<= v_1 0))) (let ((v_42 (not v_5)) (v_137 (<= v_1 40))) (let ((v_19 (not v_137)) (v_21 (* (-
1) v_17))) (let ((v_43 (<= v_21 (- 4100)))) (let ((v_38 (not v_43)) (v_20 (not (<= v_1 33)))) (let ((v_35 (and bool.b17 (not (and v_19 (and v_38 v_20)))))) (let ((v_18 (not v_3
5)) (v_45 (<= v_21 (- 4500)))) (let ((v_78 (not v_45)) (v_39 (<= v_21 (- 4910)))) (let ((v_94 (not v_39))) (let ((v_57 (not (and bool.b19 (not (and v_19 (and v_20 v_94)))))) (
let ((v_22 (not (and (and v_18 (not (and bool.b18 (not (and v_19 (and v_20 v_78)))))) v_57)))) (let ((v_32 (and bool.b23 v_22))) (let ((v_36 (not v_32)) (v_29 (and bool.b22 v_2
2))) (let ((v_33 (not v_29)) (v_26 (and bool.b21 v_22))) (let ((v_30 (not v_26)) (v_24 (and bool.b20 v_22))) (let ((v_27 (not v_24)) (v_23 (and bool.b18 v_22)) (v_47 (and bool.
b19 v_22))) (let ((v_28 (and (not v_23) (not v_47)))) (let ((v_25 (not v_28)) (v_31 (and v_27 v_28))) (let ((v_34 (and v_30 v_31))) (let ((v_37 (and v_33 v_34)) (v_123 (<= (+ v
_1 (* (/ 1 15) v_17)) (/ 964 3)))) (let ((v_121 (not v_123))) (let ((v_101 (and v_5 v_121))) (let ((v_67 (not v_101)) (v_49 (and v_38 v_35)) (v_48 (not (and bool.b19 v_39)))) (
let ((v_50 (and v_48 v_36)) (v_41 (and v_5 v_24))) (let ((v_59 (not (and v_40 v_41)))) (let ((v_52 (and v_33 v_59)) (v_58 (not (and v_72 v_41)))) (let ((v_54 (and v_30 v_58)) (
v_44 (not (and bool.b17 v_43))) (v_56 (and bool.b18 v_45))) (let ((v_46 (not v_56))) (let ((v_129 (and v_44 v_46))) (let ((v_61 (and v_44 (not (and v_129 v_23)))) (v_77 (and v_
46 (not (and v_47 (and v_46 v_48)))))) (let ((v_55 (and (not (and v_42 v_24)) (and v_61 v_77)))) (let ((v_53 (and v_54 v_55))) (let ((v_51 (and v_52 v_53))) (let ((v_68 (not (a
nd v_49 (not (and v_50 v_51))))) (v_69 (not v_49)) (v_62 (and bool.b24 (not (and v_18 v_57))))) (let ((v_60 (not (and v_56 (not v_62))))) (let ((v_65 (and v_60 (not (and v_24 (
and v_58 (and v_59 v_60)))))) (v_63 (not (and v_56 v_62)))) (let ((v_64 (and v_63 (not (and v_47 (and v_48 v_63)))))) (let ((v_66 (not (and v_61 v_64))) (v_79 (not v_61)) (v_70
 (not v_64)) (v_71 (not v_65))) (let ((v_83 (not (and v_39 v_70)))) (let ((v_90 (and v_50 v_83)) (v_73 (and v_5 v_71))) (let ((v_76 (not (and v_40 v_73)))) (let ((v_88 (and v_5
2 v_76)) (v_74 (not (and v_72 v_73)))) (let ((v_86 (and v_54 v_74)) (v_81 (and bool.b17 v_45))) (let ((v_75 (not (and v_99 v_81)))) (let ((v_84 (and v_75 (not (and v_71 (and v_
74 (and v_75 v_76)))))) (v_80 (and v_78 v_79))) (let ((v_108 (not (and (not v_77) v_80))) (v_93 (not v_80)) (v_82 (not (and bool.b24 v_81)))) (let ((v_95 (and v_82 (not (and v_
70 (and v_82 v_83)))))) (let ((v_87 (and v_93 v_95))) (let ((v_85 (not v_87)) (v_92 (not v_84)) (v_89 (and v_84 v_87))) (let ((v_91 (and v_86 v_89)) (v_105 (and v_42 v_92))) (l
et ((v_107 (not v_105)) (v_130 (and v_94 (not v_95)))) (let ((v_106 (not v_130))) (let ((v_120 (and v_22 (and v_107 (and (and v_69 v_93) v_106)))) (v_125 (+ v_1 (* (/ 1 20) v_1
7)))) (let ((v_126 (not (<= v_125 241))) (v_97 (not (and bool.b24 v_56)))) (let ((v_98 (not (and v_97 (not (and bool.b19 (and v_48 v_97)))))) (let ((v_114 (and (not (and v_39
(not (and v_82 (not (and v_98 (and (not (and v_39 v_98)) v_82))))))) v_90)) (v_100 (not (and v_99 v_56)))) (let ((v_102 (not (and v_100 (not (and bool.b20 (and (and v_100 (not
(and v_40 (and bool.b20 v_5)))) (not (and bool.b20 v_101)))))))) (let ((v_103 (and v_5 v_102))) (let ((v_104 (and v_5 (not (and v_75 (not (and v_102 (and (not (and v_72 v_103)
) (and (not (and v_40 v_103)) v_75)))))))) (let ((v_112 (and (not (and v_40 v_104)) v_88)) (v_110 (and (not (and v_72 v_104)) v_86)) (v_111 (and v_93 v_106))) (let ((v_109 (no
t v_111)) (v_118 (not v_110)) (v_113 (and v_107 v_111)) (v_117 (not v_112))) (let ((v_115 (and v_110 v_113)) (v_116 (not v_114))) (let ((v_133 (not (and v_5 (and v_40 (not (and
 v_68 (not (and v_69 (not (and (not (and v_114 (not (and (not (and v_112 (not (and (not (and v_110 (not (and (not (and v_105 v_109)) (not (and v_107 (not (and v_108 v_109))))))
)) (not (and v_118 (not v_113)))))))) (not (and v_117 (not v_115))))))) (not (and v_116 (not (and v_112 v_115)))))))))))))) (v_119 (not (and v_114 v_112))) (v_122 (<= v_1 20)))
(let ((v_135 (and v_122 (and v_121 v_105))) (v_124 (and v_122 (and v_123 v_105)))) (let ((v_143 (not v_124)) (v_128 (not (<= (* (- 1) v_127) (- 20)))) (v_138 (and bool.b17 v_38
)) (v_140 (and v_78 (not (and v_44 (not (and bool.b18 v_129))))))) (let ((v_134 (and (and (not (and v_128 v_138)) (not (and v_128 v_140))) (not (and v_128 v_130)))) (v_132 (and
 v_107 v_112))) (let ((v_131 (not v_132))) (let ((v_144 (and (not (and v_134 (not (and (not (and v_42 (not (and v_106 (and v_93 (and v_69 (and (not (and v_110 (not (and (not (a
nd v_114 (not (and v_131 (not (and v_105 v_117)))))) (not (and v_116 v_131)))))) (not (and v_118 (not (and v_114 v_132))))))))) v_133)))) (not (and v_67 (not v_134))))) (v_13
6 (+ v_1 (* (/ 3 20) v_17))) (v_141 (<= v_1 45))) (let ((v_139 (and v_141 v_20)) (v_142 (not v_141))) (or (or (exists ((lambdaprime Real)) (let ((v_145 (* 1 (+ x3 (* (/ (- 1) 2
0) lambdaprime))))) (let ((v_146 (not (<= v_145 40))) (v_148 (* (- 1) (+ x4 (* 60 lambdaprime)))) (v_147 (not (<= v_145 33)))) (and (and (<= 0 lambdaprime) (<= lambdaprime lamb
da)) (not (and (and (not (and bool.b17 (not (and v_146 (and (not (<= v_148 (- 4100))) v_147)))))) (not (and bool.b18 (not (and v_146 (and v_147 (not (<= v_148 (- 4500)))))))) (
not (and bool.b19 (not (and v_146 (and v_147 (not (<= v_148 (- 4910)))))))))))))) (< lambda 0)) (and (not (and v_96 (and (not (<= v_0 (/ 241 60))) (and (not (and v_42 (not (and
 v_11 (and v_10 (and v_6 (and (not (and v_8 (not (and (not (and v_3 (not (and v_2 (not (and bool.b20 bool.b22)))))) (not (and bool.b23 v_2)))))) (not (and bool.b21 (not (and v_
3 v_4)))))))))) (not (and v_5 (and (not (and (not (and v_6 (not (and (not (and v_3 (not (and (not (and v_7 (not (and (not (and v_8 (not (and (not (and v_9 (not (and v_12 (not
(and bool.b18 bool.b19)))))) (not (and bool.b20 v_12)))))) (not (and bool.b21 (not v_14)))))) (not (and bool.b22 (not v_15)))))) (not (and bool.b23 (not v_16)))))) (not (and
 bool.b17 (not (and v_3 v_16))))) v_40))))))) (not (and (and (not (and v_18 (not (and (not (and v_36 (not (and (not (and v_33 (not (and (not (and v_30 (not (and (not (and v_27
 (not (and v_25 (not (and bool.b19 v_23)))))) (not (and v_24 v_25)))))) (not (and v_26 (not v_31)))))) (not (and v_29 (not v_34)))))) (not (and v_32 (not v_37)))))) (not (an
d v_35 (not (and v_36 v_37))))) (not (and v_67 (not (and (and v_68 (not (and v_69 (not (and (not (and (not v_50) (not v_51))) (not (and v_50 (not (and (not (and (not v_52) (not
 v_53))) (not (and v_52 (not (and (not (and (not v_54) (not v_55))) (not (and v_54 (not (and (not (and v_65 (not (and v_66 (not (and v_79 v_70)))))) (not (and v_71 v_66)))))))
))))))))) (not (and v_67 (not (and (and v_68 (not (and v_69 (not (and (not (and v_90 (not (and (not (and v_88 (not (and (not (and v_86 (not (and (not (and v_84 (not (and v_10
8 v_85)))) (not (and v_92 v_85)))))) (not (and (not v_86) (not v_89))))))) (not (and (not v_88) (not v_91))))))) (not (and (not v_90) (not (and v_88 v_91))))))))) (not (and (no
t (and v_120 (and v_96 (and v_126 (and v_133 (not (and v_42 (not (and v_69 (and v_107 (and v_106 (and v_93 (and (not (and v_110 (not (and v_119 (not (and v_116 v_117)))))) (not
 (and v_118 v_119)))))))))))))) (not (and (not v_120) (and (not (and (not v_135) (not (and (not (and v_67 v_143)) (not (and v_124 (not (and (<= v_125 400) (and v_126 v_144)))))
```

Automated Reasoning and the future of Formal Methods

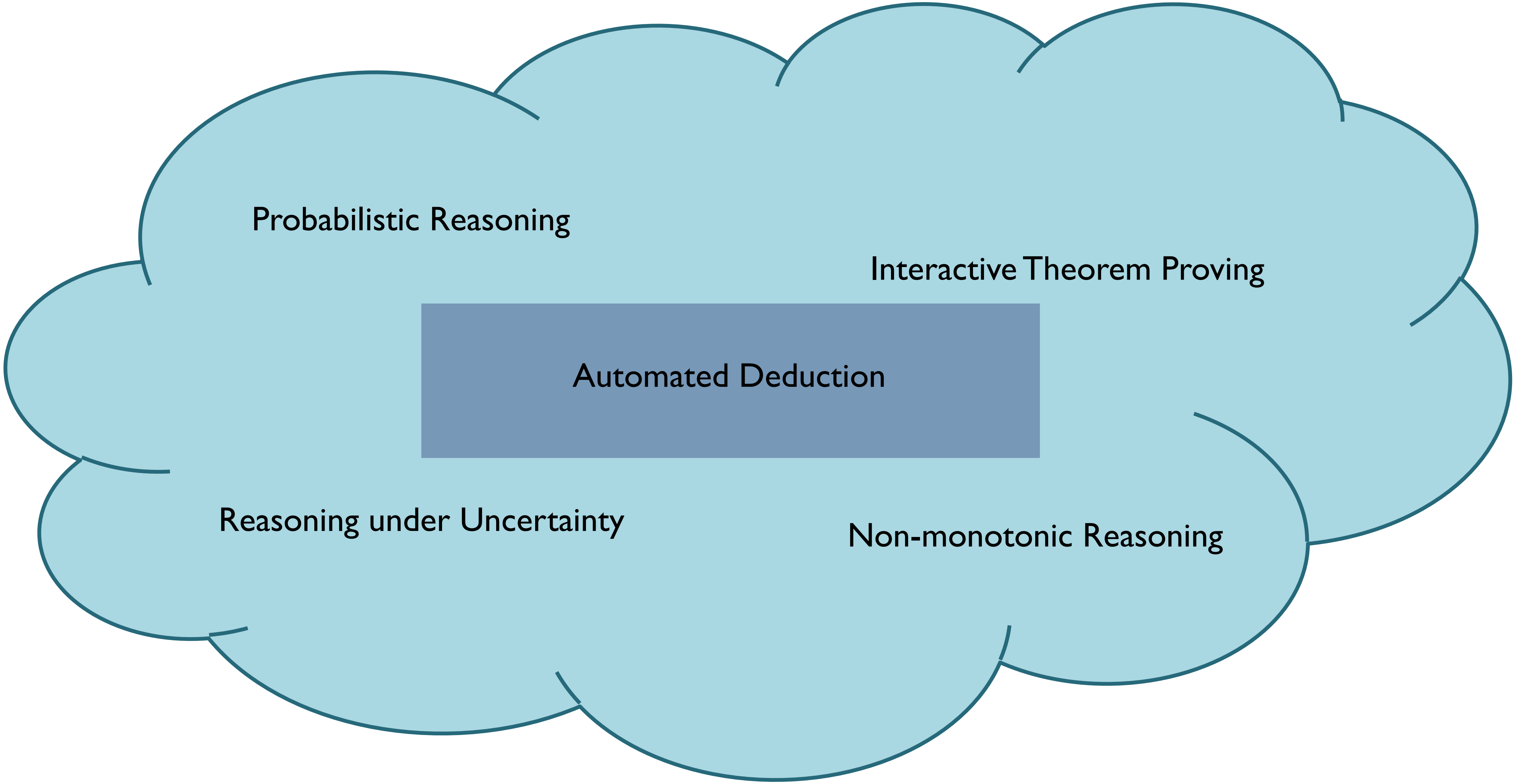
Clark Barrett
Stanford University

FM@Scale
October 9, 2019
SRI

Outline

- Improving Core AR Engines
- The Many Uses of Proofs
- Improving Usability

What is Automated Reasoning?



Automated Reasoning Engines

- Find p and q:
  - $(p \vee \neg q) \wedge (\neg p \vee q)$
  - *Boolean Satisfiability (SAT)*
  - The original NP-complete problem
- Prove or disprove:
  - $\exists x. (P(x) \rightarrow \forall y. P(y))$
  - *Automated Theorem Proving (ATP)*
  - Pure first-order logic
  - Semi-decidable
- Find a solution:
  - $a = b + 2 \wedge A = write(B, a + 1,4) \wedge (A[b + 3] = 2 \vee f(a - 1) \neq f(b + 1))$
  - *Satisfiability Modulo Theories (SMT)*
  - Language includes Boolean logic, first-order logic, and certain built-in theories
  - Theory examples: arithmetic, arrays, functions, bitvectors, strings, sets, etc.
  - From NP-complete to undecidable

What is Automated Reasoning Good For?

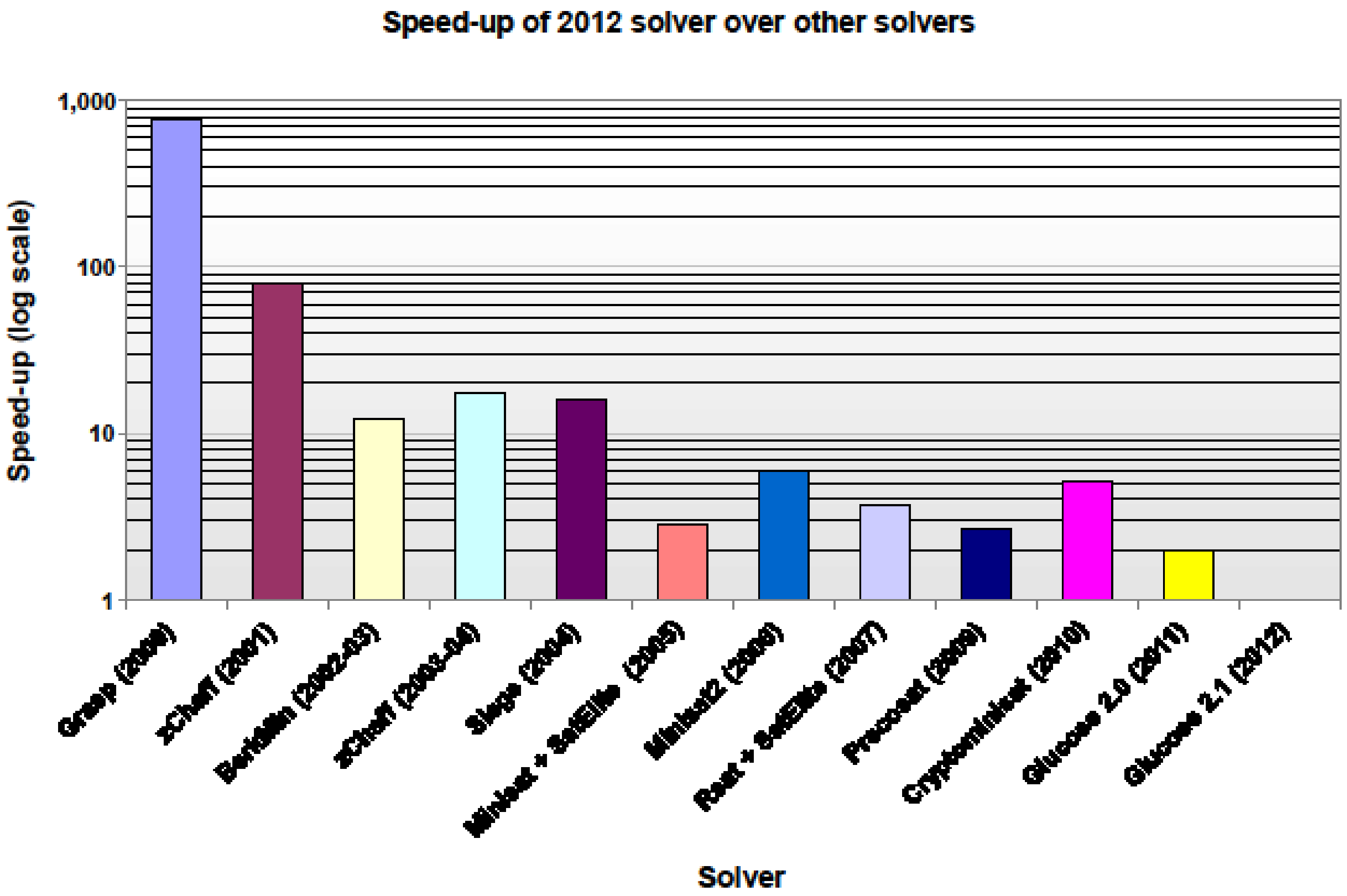
- Used for lots of things, but one big success is when coupled with *formal methods* to check whether a system conforms to some desired property

- Safety
  - Critical systems don't fail catastrophically
- Security
  - Systems are free from vulnerabilities
- Verification
  - Systems behave as intended

New Capabilities of Automated Reasoning

- *Faster*
  - Off-the shelf performance of tools has increased by orders of magnitude
- *Stronger*
  - More deductive power in modern tools (e.g. increasing number of supported theories in SMT solvers)
- *Better*
  - Flexible, adaptable, extensible – modern AR platforms can be modified for new challenging problems

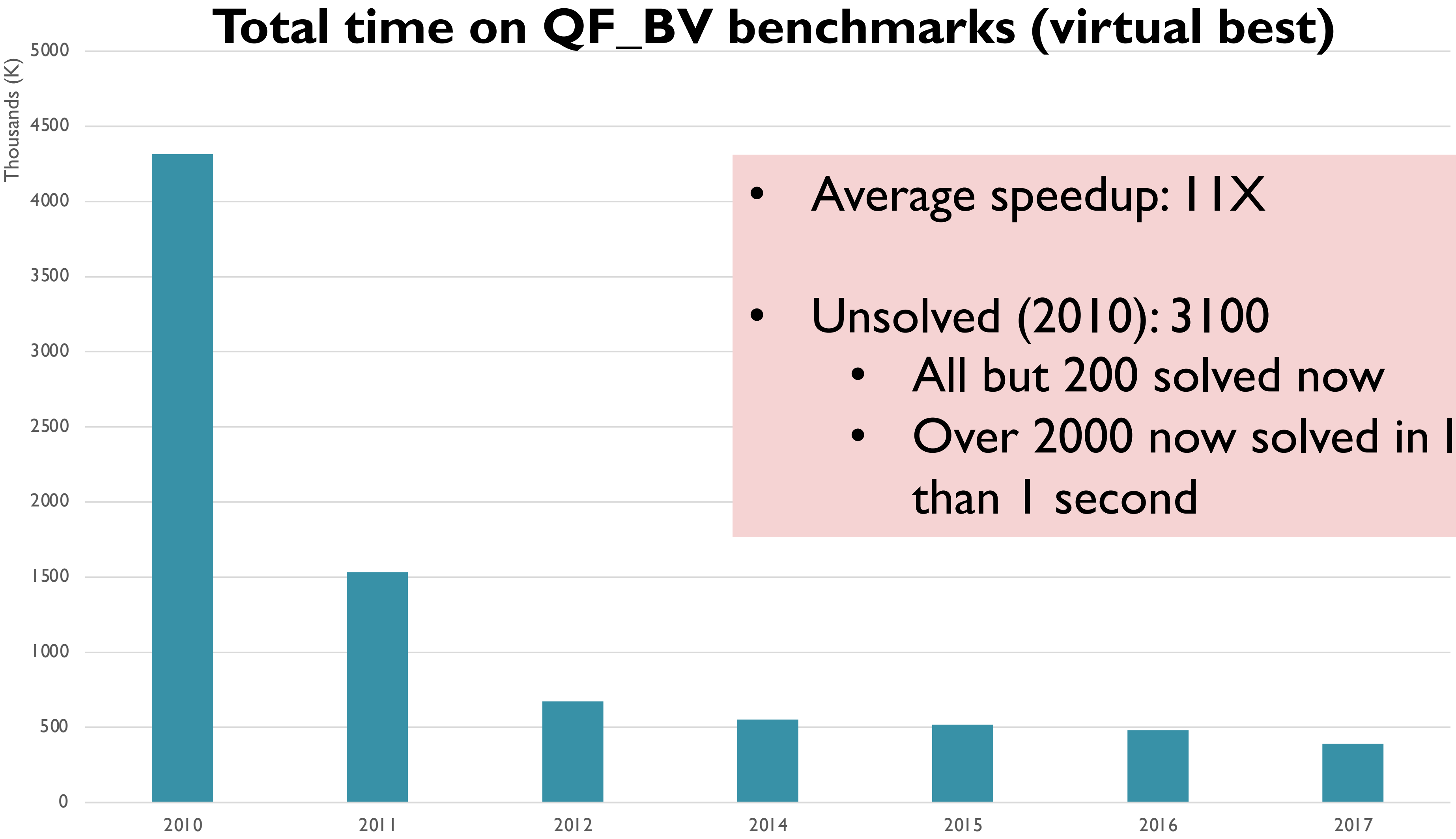
Some Experience with SAT Solving



6

Moshe Y. Vardi, "Machine Learning and Logic: Fast and Slow Thinking"
Summit on Machine Learning Meets Formal Methods, July 2018

Evolution of SMT Solving



- Average speedup: 11X
- Unsolved (2010): 3100
  - All but 200 solved now
  - Over 2000 now solved in less than 1 second

Automated Reasoning: Opportunities

- Core engines can still improve *dramatically*
- Need more people *developing* AR engines
  - Fund system-building proposals in AR!
  - Competition for new tools?
- Co-evolve engines with applications
  - Example: verification of neural networks
- Pursue parallelization
  - Some promising directions in SAT (cube and conquer)
- Use machine learning to tune configurations and strategies

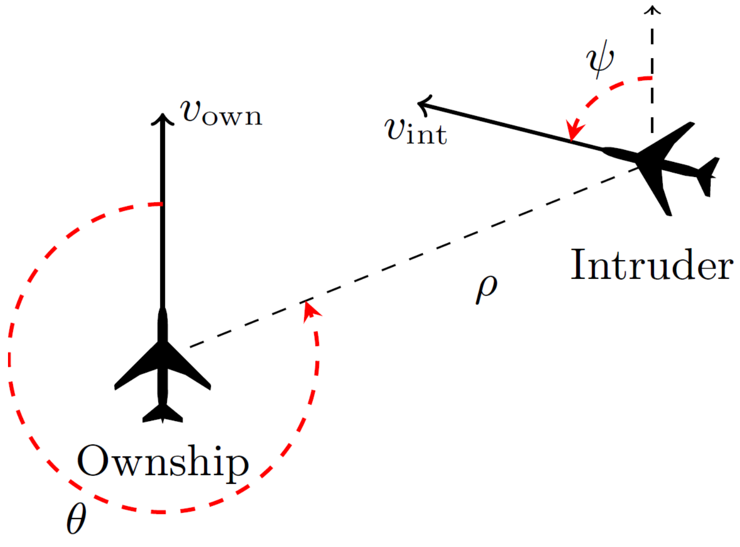
Automated Reasoning: Opportunities

- Core engines can still improve *dramatically*
- Need more people *developing* AR engines
  - Fund system-building proposals in AR!
  - Competition for new tools?
- Co-evolve engines with applications
  - Example: verification of neural networks
- Pursue parallelization
  - Some promising directions in SAT (cube and conquer)
- Use machine learning to tune configurations and strategies

Motivation: ACAS Xu

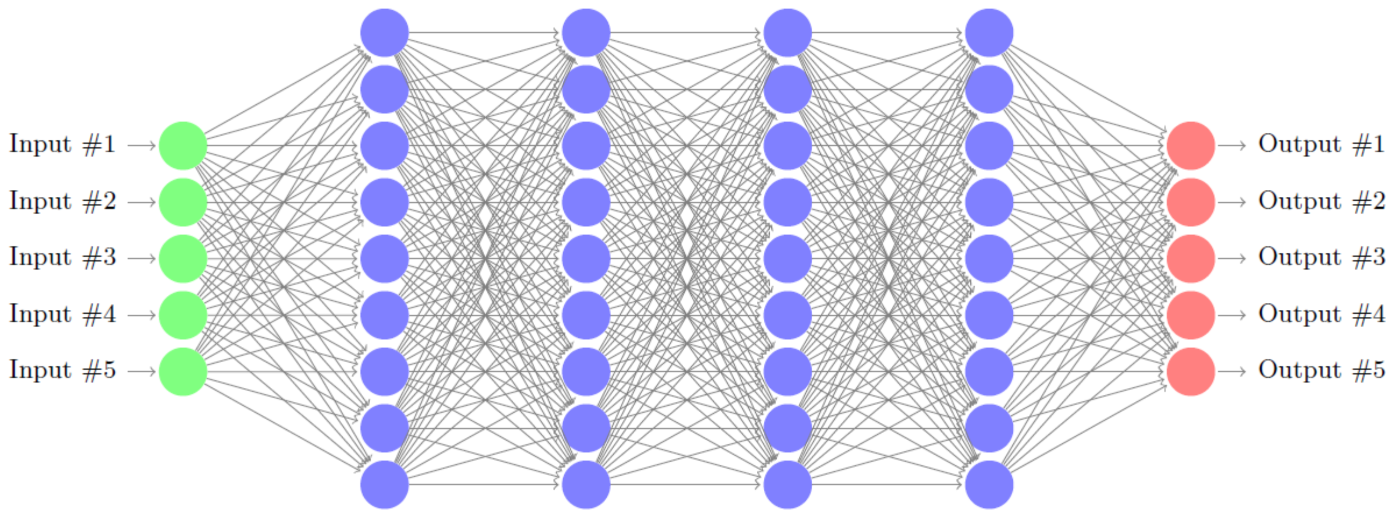
- Airborne Collision-Avoidance System for drones
- A new standard being developed by the FAA

- Produce advisories:
  1. Strong left (SL)
  2. Weak left (L)
  3. Strong right (SR)
  4. Weak right (R)
  5. Clear of conflict (COC)



- Best-performing implementation uses 45 deep neural networks
  - How do we verify them?

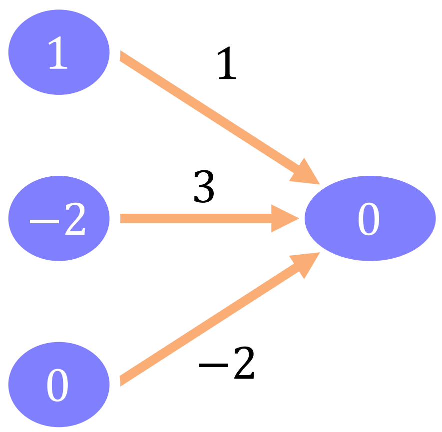
Deep Neural Nets (DNNs)



- ACAS Xu networks: 8 layers, 310 nodes (x 45)
- Naïve translation to SMT scales to networks with ~20 nodes
- NP-Complete problem!

The Culprits: Rectified Linear Units (ReLUs)

- ReLU(x) = $\max(0, x)$
  - $x \geq 0$: active case, return x
  - $x < 0$: inactive case, return 0
  - Example:



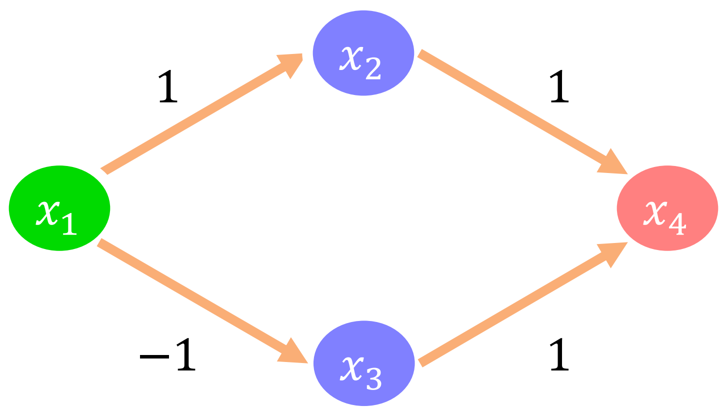
$$1 \cdot 1 + (-2) \cdot 3 + 0 \cdot (-2) = -5$$

ReLU(-5) = 0

Reluplex: SMT Solver for Neural Networks

- A technique for solving linear programs with ReLUs
  - Can encode neural networks as input
- Extends the simplex method
- Does *not* require case splitting in advance
  - ReLU constraints satisfied incrementally
  - Split only if we must
- Scales to the ACAS Xu networks
  - An order of magnitude larger networks than previously possible

A Simple Example



- Property being checked:
  Is it possible that $x_1 \in [0,1]$ and $x_4 \in [0.5,1]$?

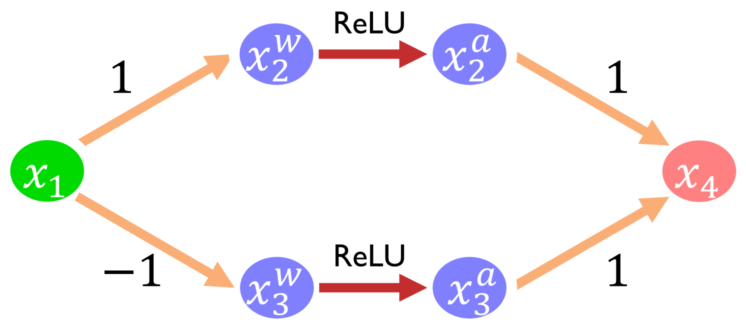
Encoding Networks

- Introduce equalities:

$$x_2^w - x_1 = 0$$
$$x_3^w + x_1 = 0$$
$$x_4 - x_3^a - x_2^a = 0$$

- Set bounds:
  $x_1 \in [0,1]$
  $x_4 \in [0.5,1]$
  $x_2^w, x_3^w \in (-\infty, \infty)$
  $x_2^a, x_3^a \in [0, \infty)$

- Special ReLU constraints:
  $x_2^a = ReLU(x_2^w)$
  $x_3^a = ReLU(x_3^w)$



Reluplex: Example

$$x_5 = x_2^w - x_1$$

$$x_6 = x_2^w + x_3^w - x_5$$

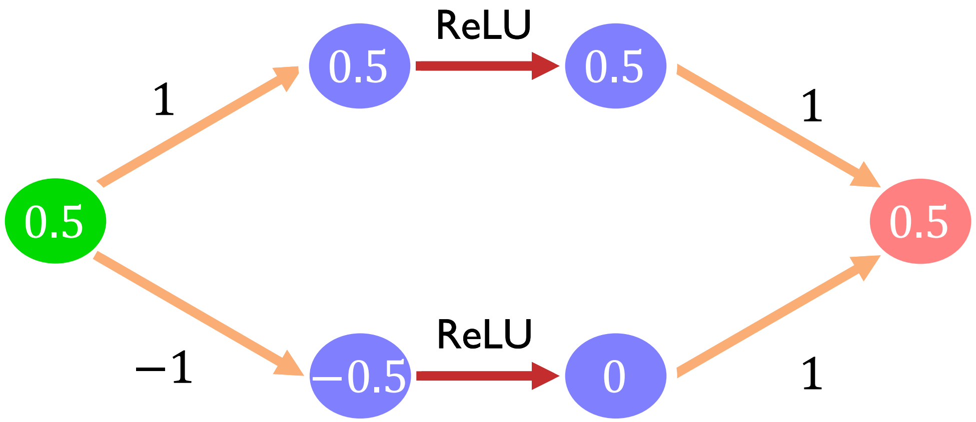
$$x_2^a = x_4 - x_3^a - x_7$$

<u>Operation:</u>

Success $x_2^w := x_2^w + 0.5$

Lower Bound	Variable	Assignment	Upper Bound
0	x_1	0.5	1
	x_2^w	0.5	
0	x_2^a	0.5	
	x_3^w	-0.5	
0	x_3^a	0	
0.5	x_4	0.5	1
0	x_5	0.5	0
0	x_6	0.5	0
0	x_7	0.5	0

The Assignment is a Solution

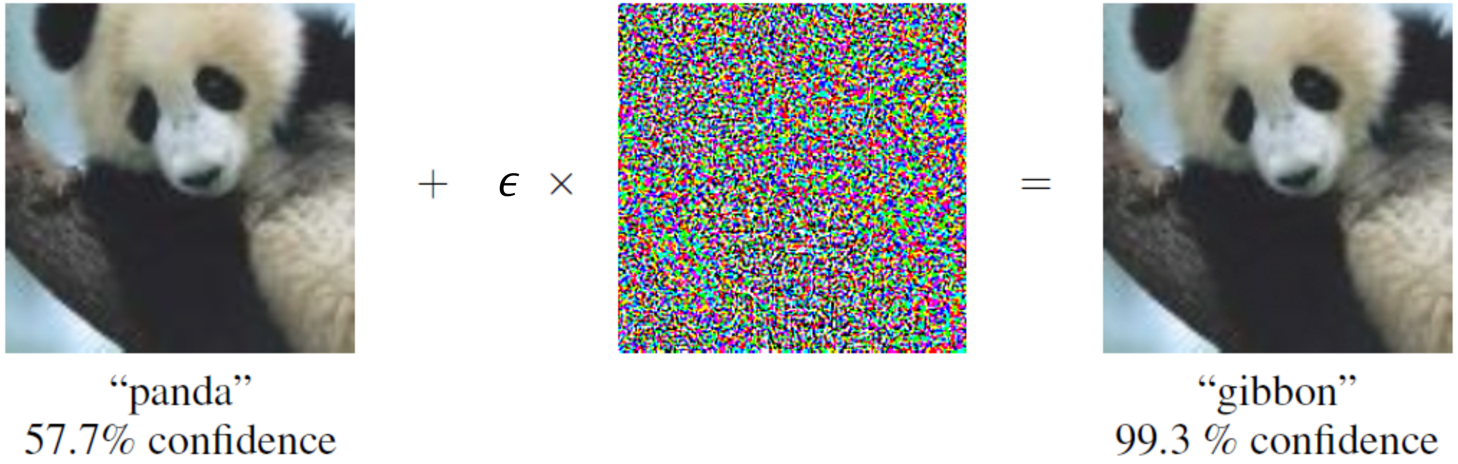


- Property being checked:
  Is it possible that $x_1 \in [0,1]$ and $x_4 \in [0.5,1]$?

Robustness to Adversarial Inputs

- Slight input perturbations cause misclassification

Goodfellow et al., 2015



- We can *prove* that these cannot occur (for given input and amount of noise)

Outline

- Improving Core AR Engines
- The Many Uses of Proofs
- Improving Usability

Who Watches The WATCHMEN?

The Need for Proofs

- If an AR engine returns a model/counter-example, it can be checked
- But if it returns unsatisfiable, the result has to be trusted
- …unless the tool can produce an independently-checkable proof
- There is already a standard for SAT solvers

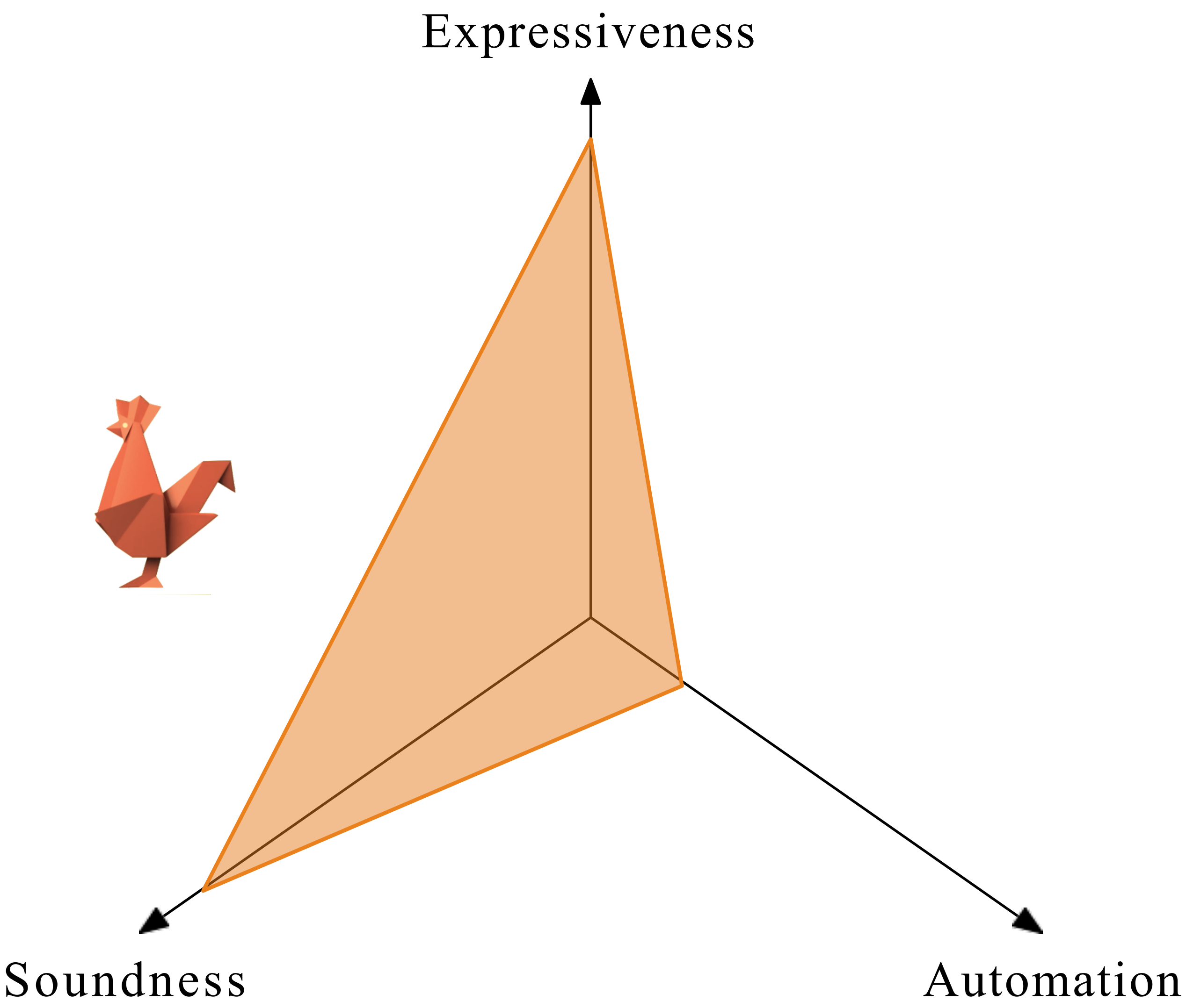
What are Proofs good for?

- Reducing the trusted code base
- Improving code quality of AR engines
- *Trusted* interoperability with other tools
- Can be mined for additional information (e.g. interpolants)
- Auditable trail for building assurance cases

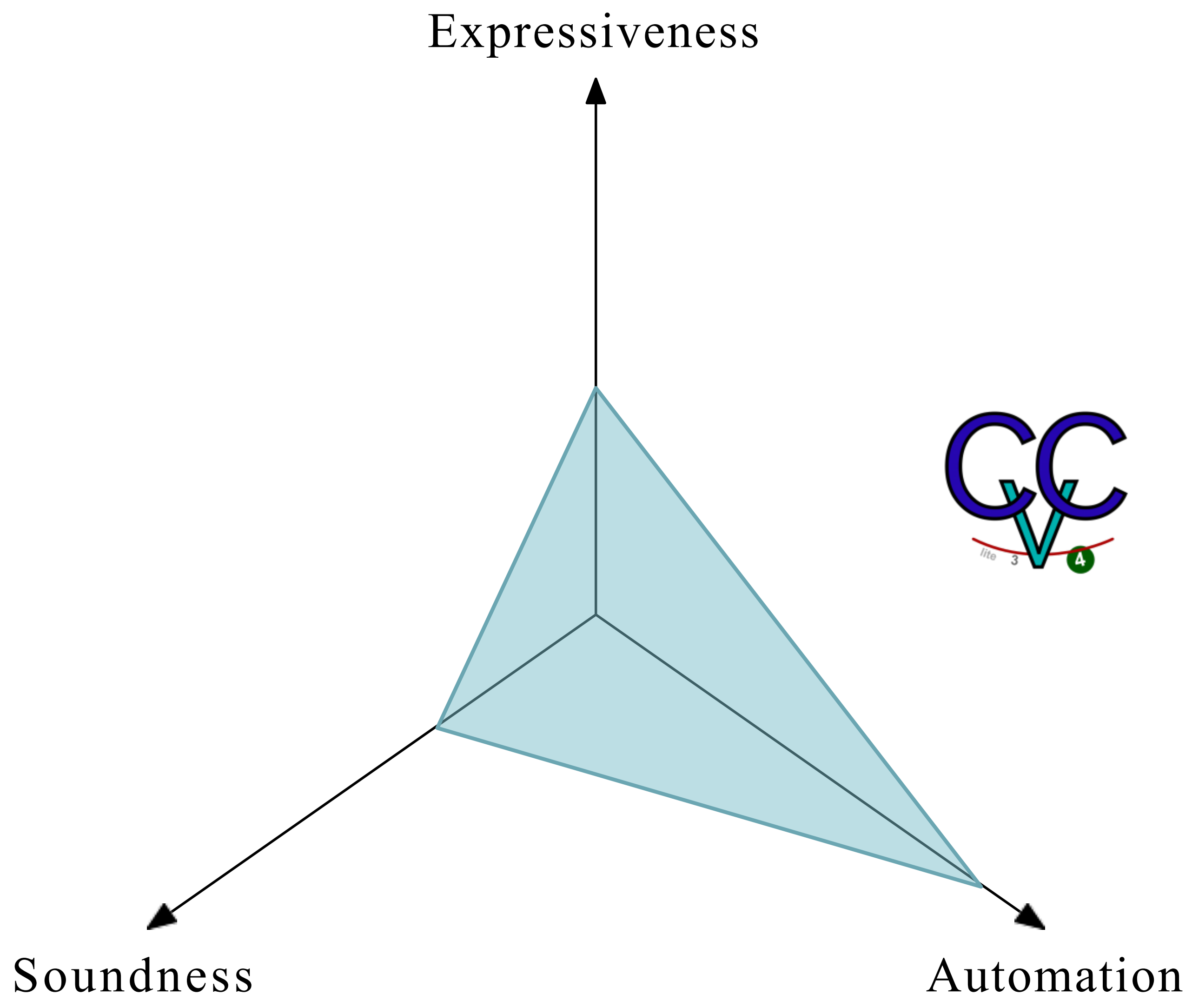
What are Proofs good for?

- Reducing the trusted code base
- Improving code quality of AR engines
- *Trusted* interoperability with other tools
- Can be mined for additional information (e.g. interpolants)
- Auditable trail for building assurance cases

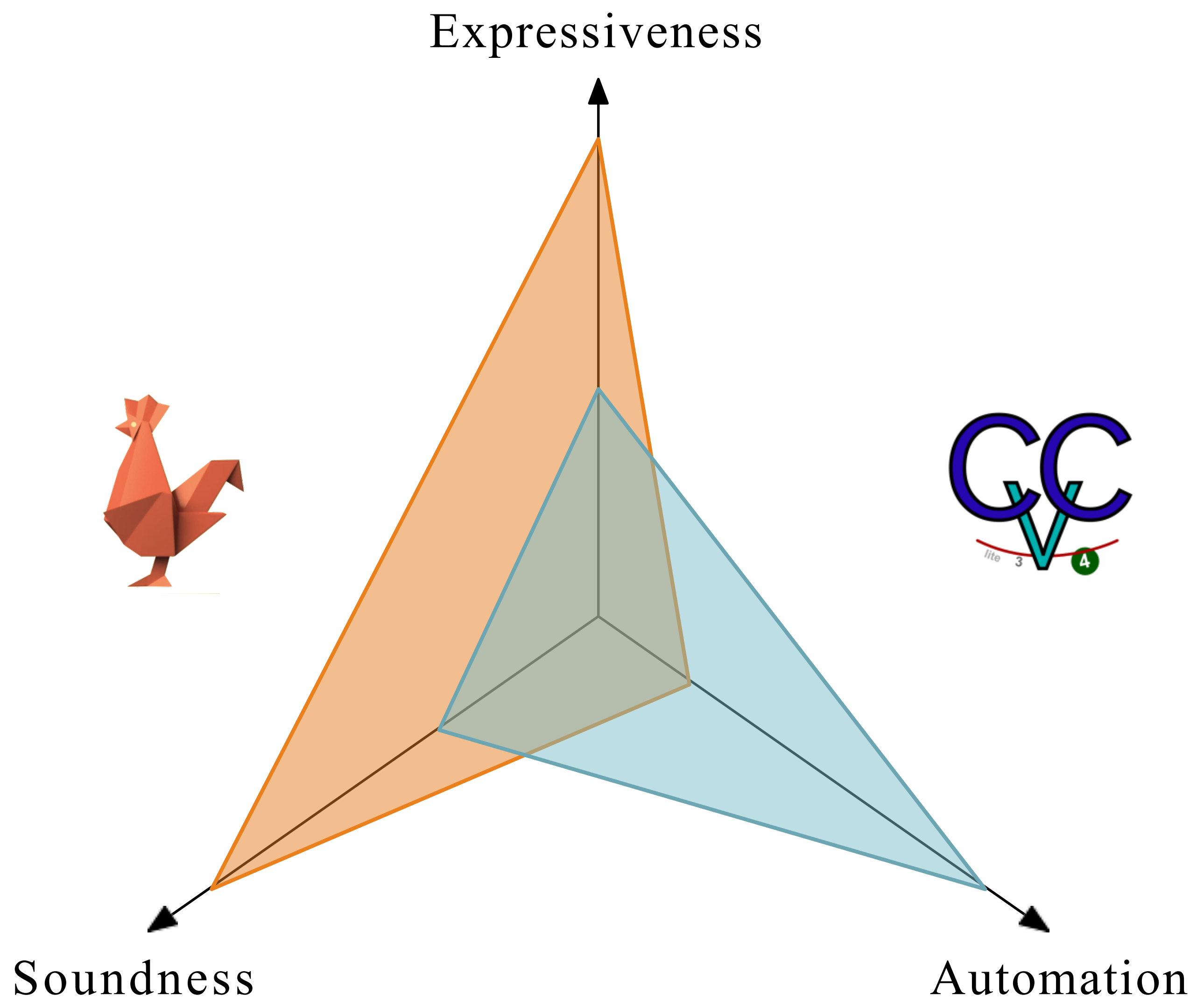
Proof assistants vs SMT solvers



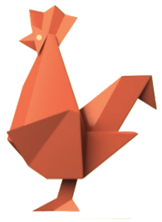
Proof assistants vs SMT solvers



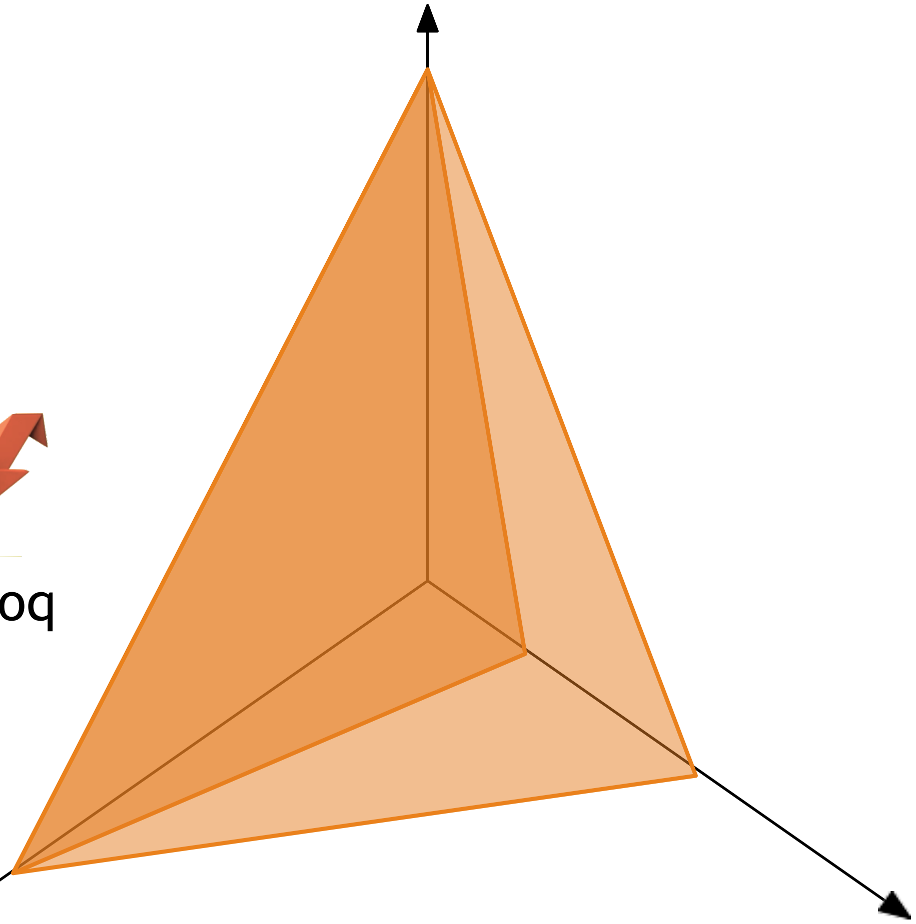
Proof assistants vs SMT solvers



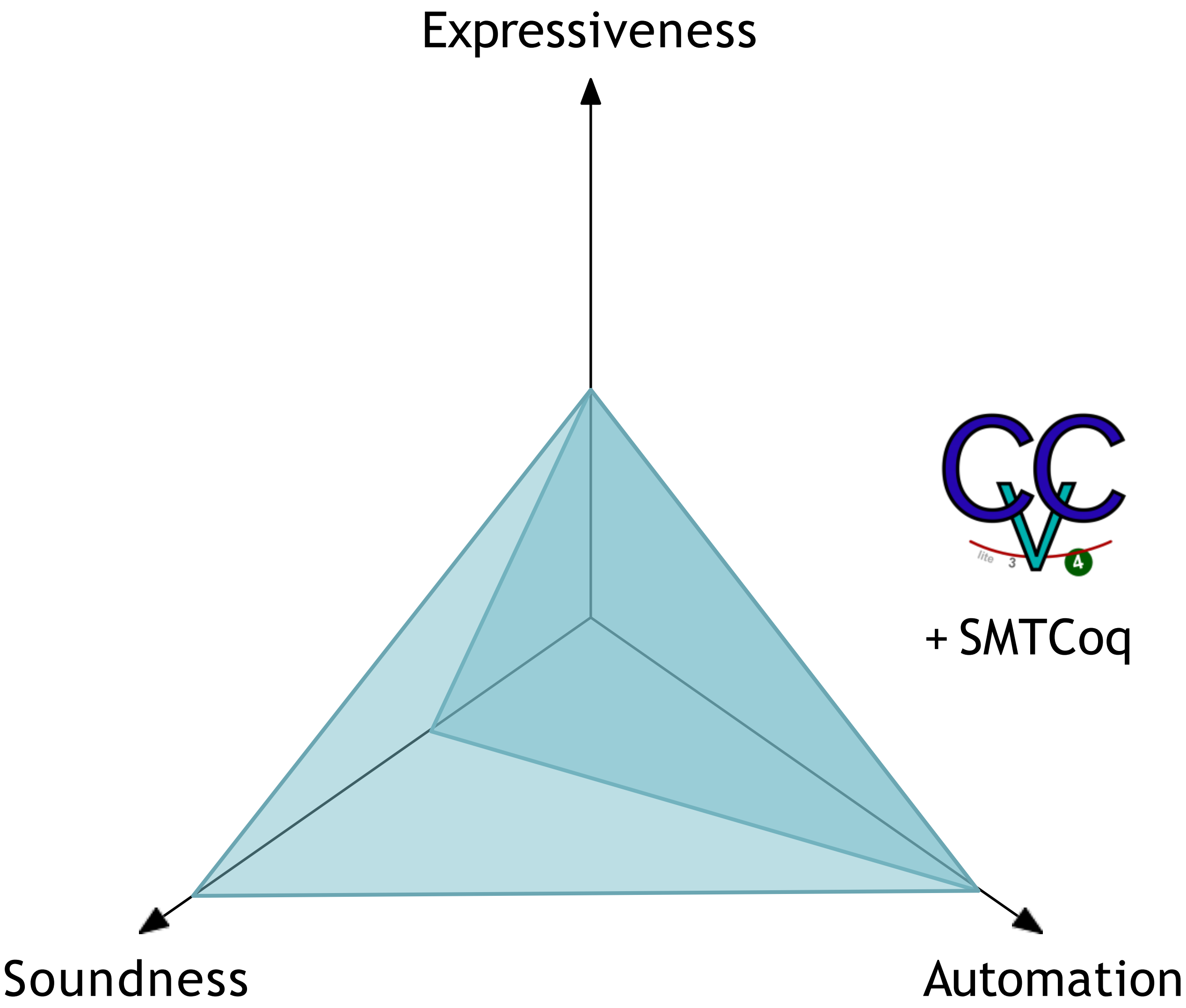
Effects of SMTCoq



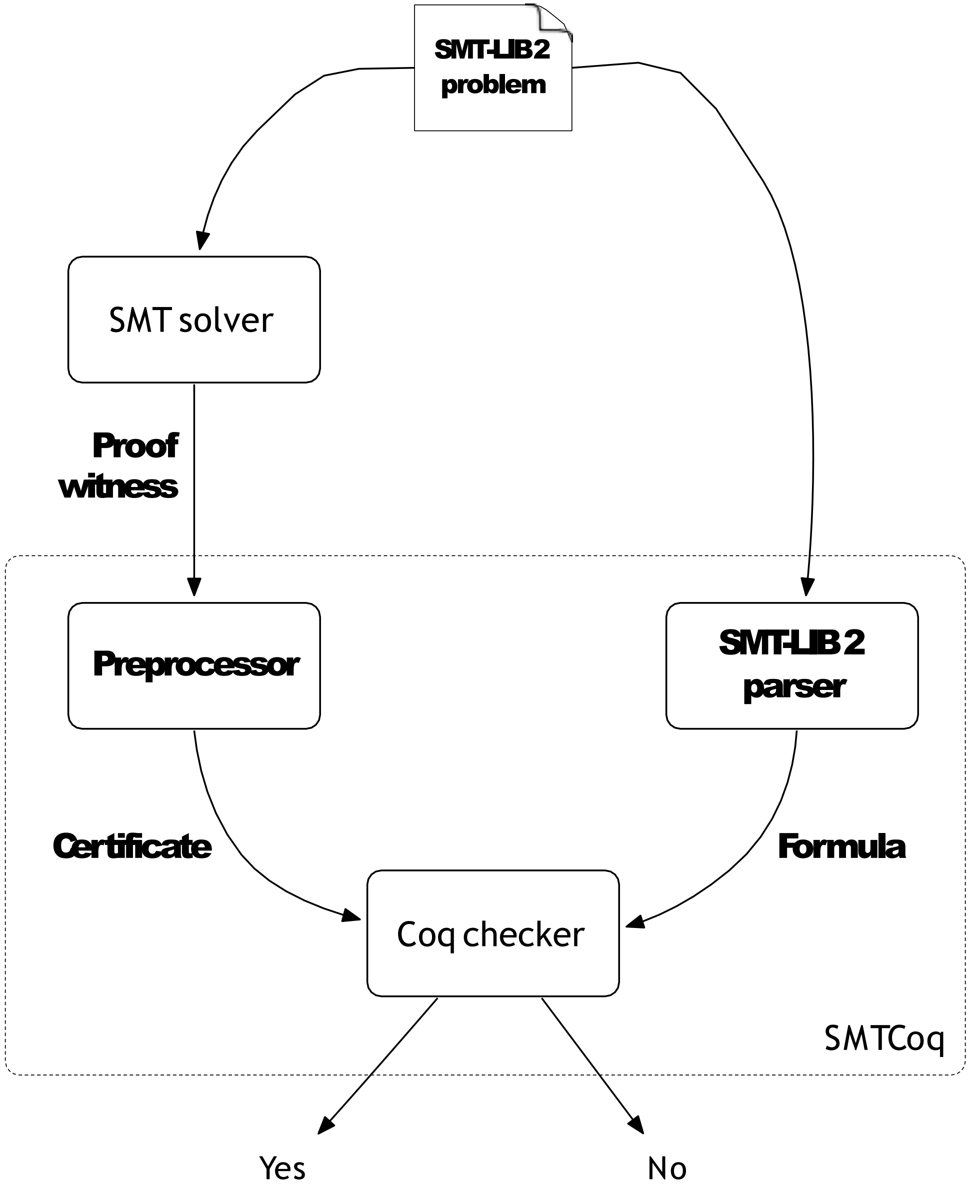
+ SMTCoq



Effects of SMTCoq as a proof checker

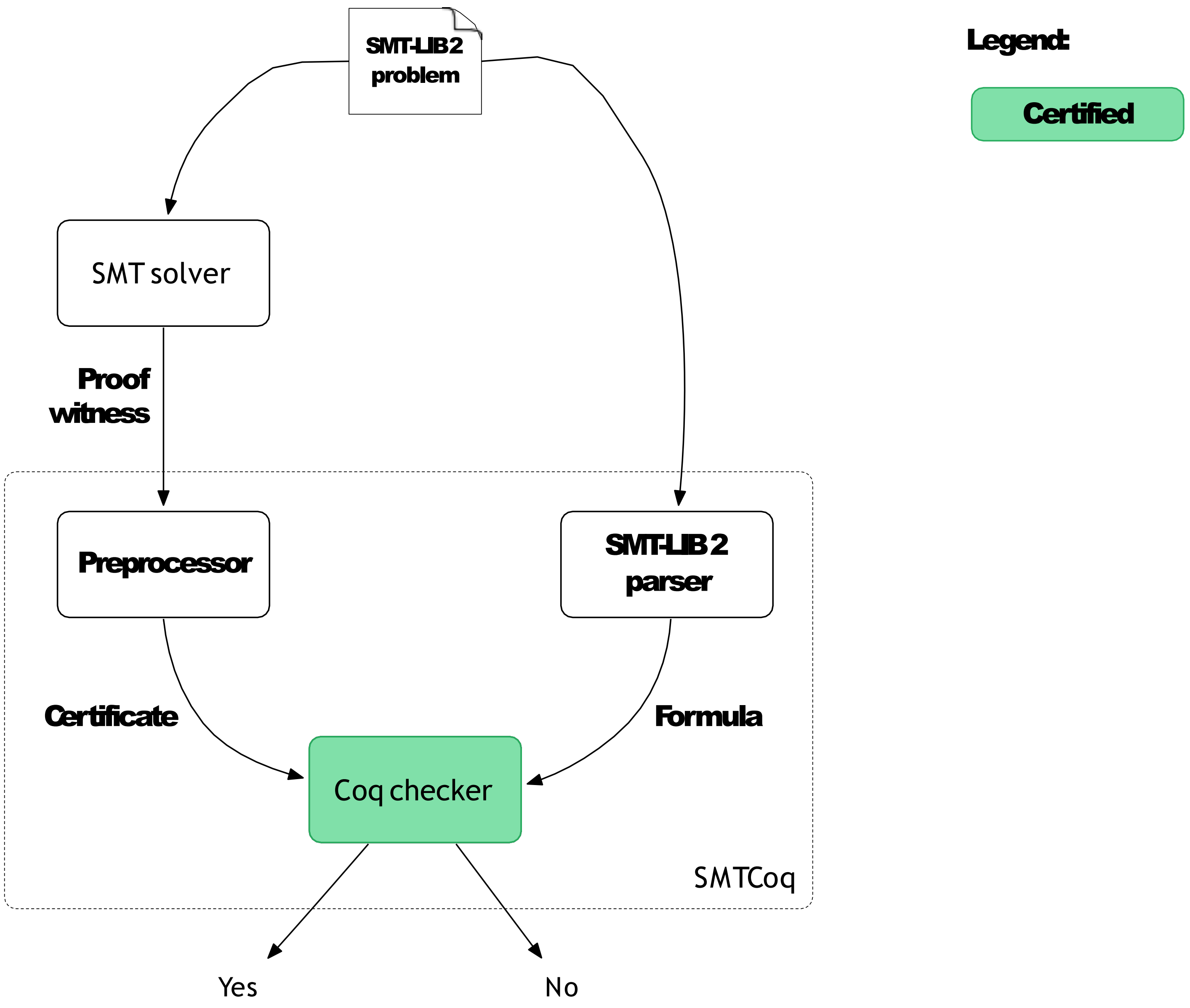


SMTCoq as a stand-alone checker

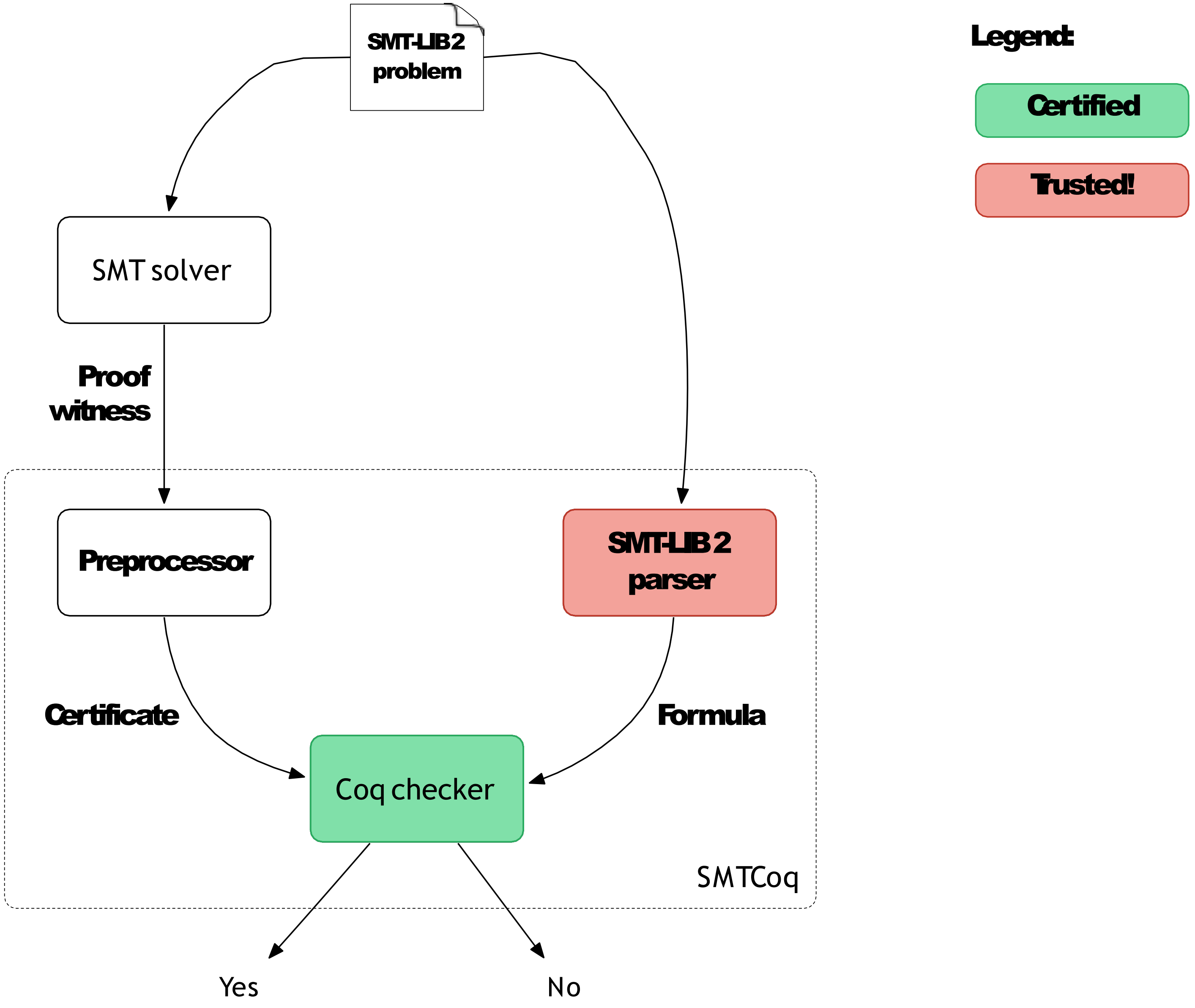


Legend:

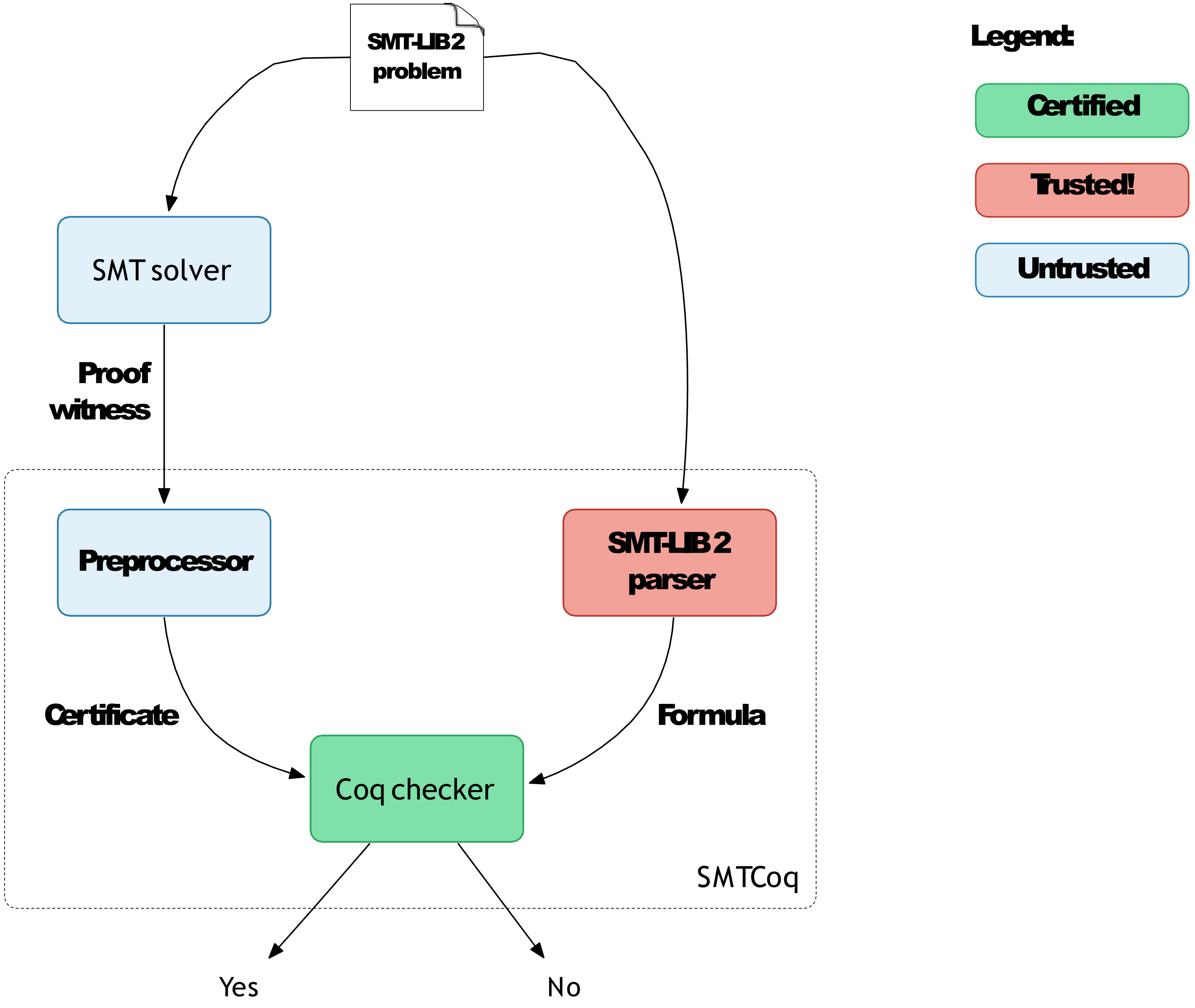
SMTCoq as a stand-alone checker



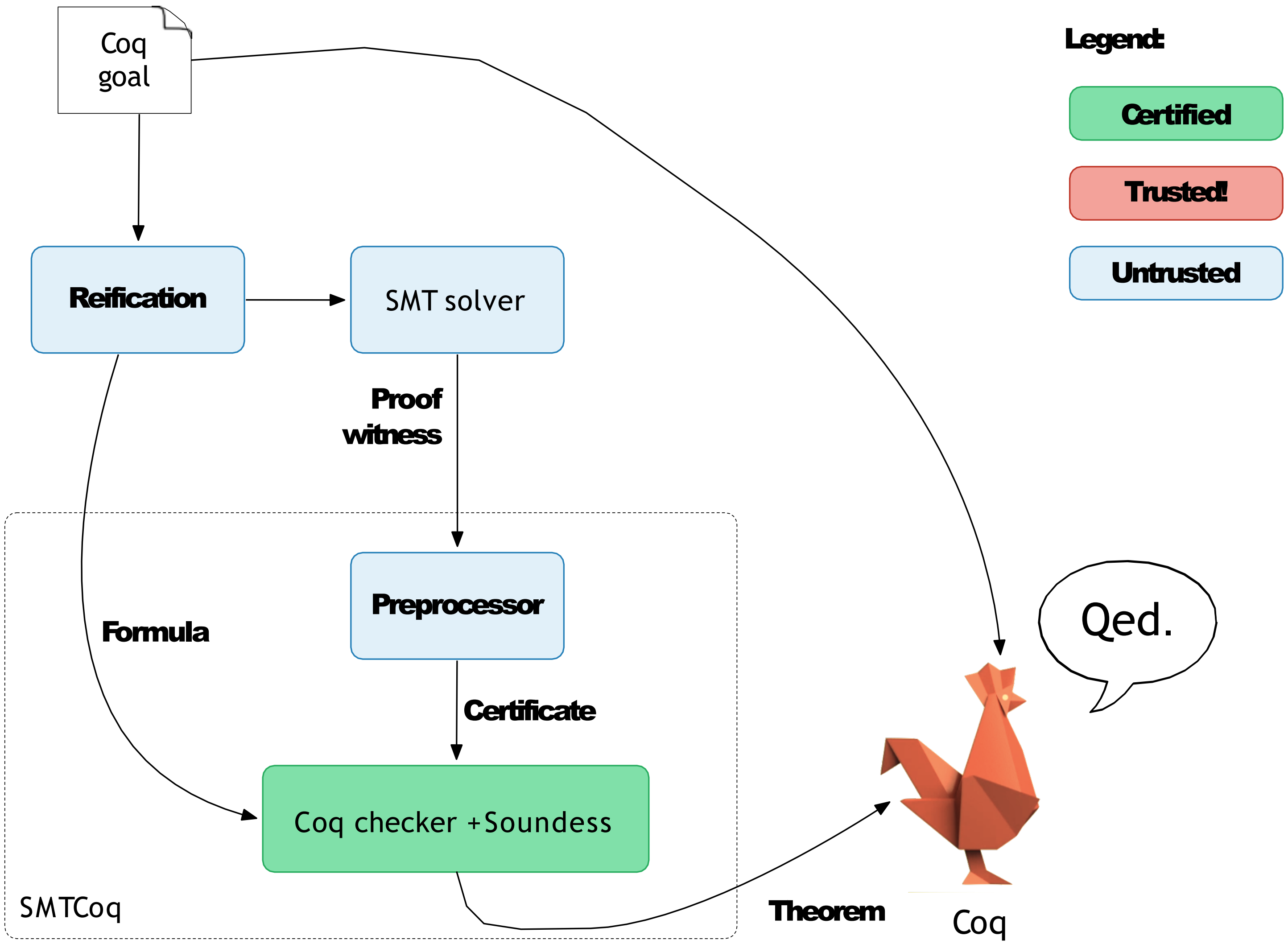
SMTCoq as a stand-alone checker



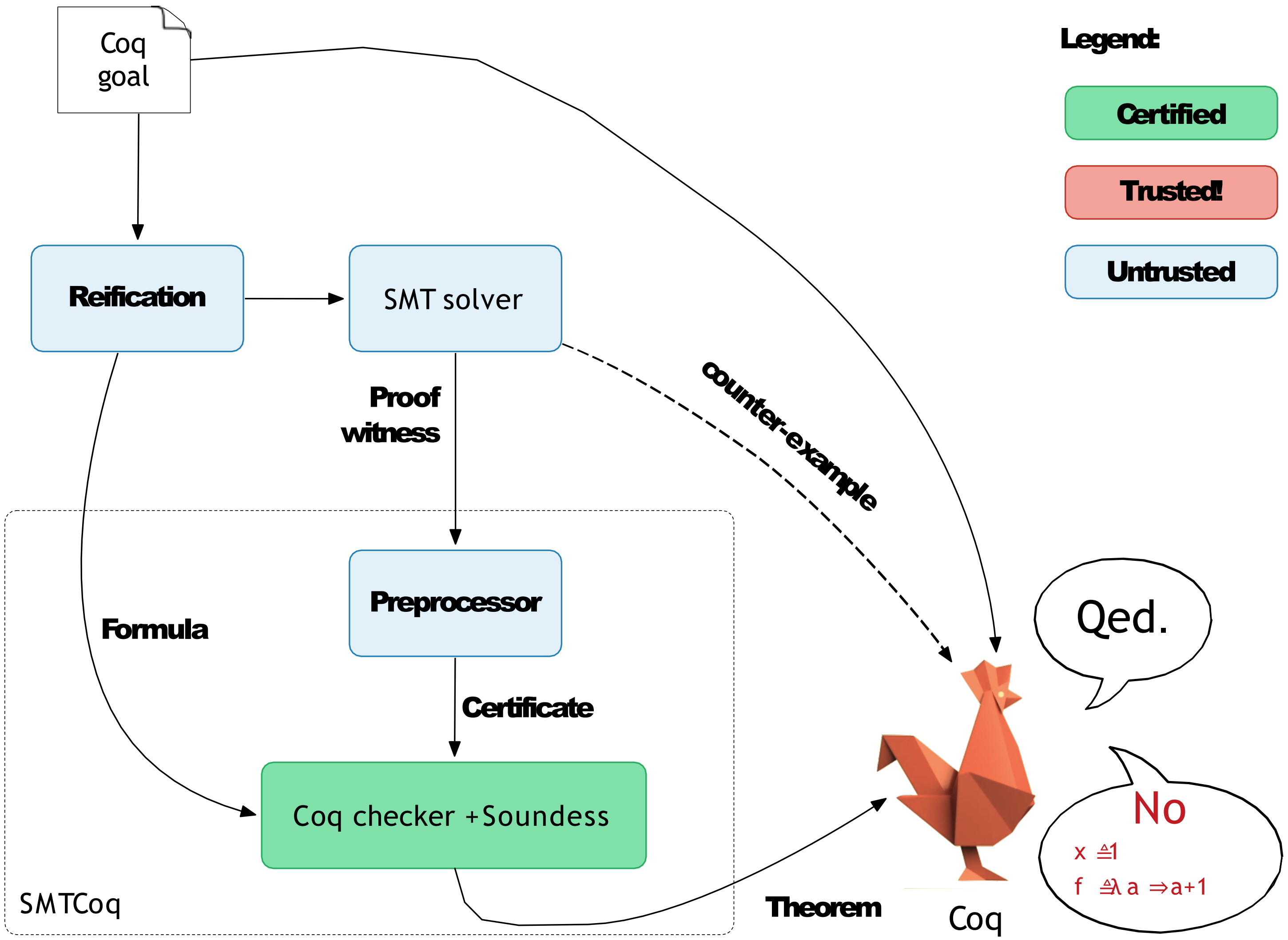
SMTCoq as a stand-alone checker



SMTCoq from within Coq



SMTCoq from within Coq



Outline

- Improving Core AR Engines
- The Many Uses of Proofs
- Improving Usability

The Usability Challenge

- Experts can do great things with formal tools
- Need to find more ways for non-experts to benefit from formal tools
- Develop Tools and Techniques that use formal under the hood but expose a simple interface for users

The Usability Challenge

- Experts can do great things with formal tools
- Need to find more ways for non-experts to benefit from formal tools
- Develop Tools and Techniques that use formal under the hood but expose a simple interface for users
    - Example: Symbolic QED for hardware verification

Quick Error Detection

QED

- Technique developed by *Subhasish Mitra*'s group
- Key idea
  - Use regular and shadow values for registers and memory
  - Apply *duplicate and check* transformation to improve tests

Quick Error Detection

QED

- Technique developed by *Subhasish Mitra*'s group
- Key idea
  - Use regular and shadow values for registers and memory
  - Apply *duplicate and check* transformation to improve tests

Example

	Regular	Shadow
Registers	R0…R15	R16…R31
Memory	0x10000 – 0x1FFFF	0x20000 – 0x2FFFF

Quick Error Detection

QED

- Technique developed by *Subhasish Mitra*'s group
- Key idea
  - Use regular and shadow values for registers and memory
  - Apply *duplicate and check* transformation to improve tests

Example

	Regular	Shadow
Registers	R0…R15	R16…R31
Memory	0x10000 – 0x1FFFF	0x20000 – 0x2FFFF

```
...
LD R1, [0x10000]
LD R2, [0x10040]
...
```

→

```
LD R1, [0x10000]
LD R2, [0x10040]
LD R17, [0x20000]
LD R18, [0x20040]
CMP R1 == R17
CMP R2 == R18
```

Quick Error Detection

QED features

- Improves *coverage* and *speed* of bug detection with respect to standard testing
- Reduces *error latency* (time between when bug is activated and detected)

QED limitations

- *Not exhaustive* - might miss bugs
- Error latency can still be *hundreds of instructions*

Quick Error Detection

QED features

- Improves *coverage* and *speed* of bug detection with respect to standard testing
- Reduces *error latency* (time between when bug is activated and detected)

QED limitations

- *Not exhaustive* - might miss bugs
- Error latency can still be *hundreds of instructions*

SoC Verification

Idea

- Combine QED with *Bounded Model Checking*

Symbolic QED

Result: *Symbolic QED*

- Collaboration with Subhasish Mitra's group [13]
- *Idea*: use BMC to search through *all possible* QED tests
  - Initial state: *QED-consistent* (regular and shadow values match)
  - Input must be sequence of regular instructions followed by duplicate instructions
  - Property: *final state must be QED-consistent*

Addresses limitations of QED

- Exhaustively covers *all possible* QED tests
- Finds *minimum length* QED test that triggers bug

Symbolic QED

Result: *Symbolic QED*

- Collaboration with Subhasish Mitra's group [13]
- *Idea*: use BMC to search through *all possible* QED tests
  - Initial state: *QED-consistent* (regular and shadow values match)
  - Input must be sequence of regular instructions followed by duplicate instructions
  - Property: *final state must be QED-consistent*

Addresses limitations of QED

- Exhaustively covers *all possible* QED tests
- Finds *minimum length* QED test that triggers bug

A GENERIC APPROACH TO SYMBOLIC QED

Designer Hat: Fill out Template Format File

ISA/Design Format File

Demo:
RIDECORE (RISC-V)

Design

Conclusions

- Improvements in core AR engines have big payoffs
- Significant progress can be made by evolving engines driven by new applications
  - But need to build more talent and expertise in solver development
- An ecosystem of interchangeable proofs would enable high-trust interoperability
- Need to find creative ways to make formal power accessible to non-experts

```
(assert (forall ((lambda Real)) (let ((v_17 (+ x4 (* 60 lambda))) (v_11 (not bool.b19)) (v_10 (not bool.b18)) (v_6 (not bool.b17)) (v_8 (not bool.b21)) (v_3 (not bool.b23)) (v_
9 (not bool.b20)) (v_7 (not bool.b22))) (let ((v_4 (and v_9 v_7))) (let ((v_2 (not v_4)) (v_13 (and v_10 v_11))) (let ((v_12 (not v_13)) (v_14 (and v_9 v_13))) (let ((v_15 (and
 v_8 v_14))) (let ((v_16 (and v_7 v_15)) (v_40 (<= (* 1 v_17) 4820))) (let ((v_72 (not v_40)) (v_99 (not bool.b24)) (v_127 (+ x3 (* (/ (- 1) 20) lambda)))) (let ((v_1 (* 1 v_12
7))) (let ((v_0 (+ v_1 (* (/ 1 1200) v_17)))) (let ((v_96 (<= v_0 (/ 20 3))) (v_5 (<= v_1 0))) (let ((v_42 (not v_5)) (v_137 (<= v_1 40))) (let ((v_19 (not v_137)) (v_21 (* (-
1) v_17))) (let ((v_43 (<= v_21 (- 4100)))) (let ((v_38 (not v_43)) (v_20 (not (<= v_1 33)))) (let ((v_35 (and bool.b17 (not (and v_19 (and v_38 v_20)))))) (let ((v_18 (not v_3
5)) (v_45 (<= v_21 (- 4500)))) (let ((v_78 (not v_45)) (v_39 (<= v_21 (- 4910)))) (let ((v_94 (not v_39))) (let ((v_57 (not (and bool.b19 (not (and v_19 (and v_20 v_94)))))) (
let ((v_22 (not (and (and v_18 (not (and bool.b18 (not (and v_19 (and v_20 v_78)))))) v_57)))) (let ((v_32 (and bool.b23 v_22))) (let ((v_36 (not v_32)) (v_29 (and bool.b22 v_2
2))) (let ((v_33 (not v_29)) (v_26 (and bool.b21 v_22))) (let ((v_30 (not v_26)) (v_24 (and bool.b20 v_22))) (let ((v_27 (not v_24)) (v_23 (and bool.b18 v_22)) (v_47 (and bool.
b19 v_22))) (let ((v_28 (and (not v_23) (not v_47)))) (let ((v_25 (not v_28)) (v_31 (and v_27 v_28))) (let ((v_34 (and v_30 v_31))) (let ((v_37 (and v_33 v_34)) (v_123 (<= (+ v
_1 (* (/ 1 15) v_17)) (/ 964 3)))) (let ((v_121 (not v_123))) (let ((v_101 (and v_5 v_121))) (let ((v_67 (not v_101)) (v_49 (and v_38 v_35)) (v_48 (not (and bool.b19 v_39)))) (
let ((v_50 (and v_48 v_36)) (v_41 (and v_5 v_24))) (let ((v_59 (not (and v_40 v_41)))) (let ((v_52 (and v_33 v_59)) (v_58 (not (and v_72 v_41)))) (let ((v_54 (and v_30 v_58)) (
v_44 (not (and bool.b17 v_43))) (v_56 (and bool.b18 v_45))) (let ((v_46 (not v_56))) (let ((v_129 (and v_44 v_46))) (let ((v_61 (and v_44 (not (and v_129 v_23)))) (v_77 (and v_
46 (not (and v_47 (and v_46 v_48)))))) (let ((v_55 (and (not (and v_42 v_24)) (and v_61 v_77)))) (let ((v_53 (and v_54 v_55))) (let ((v_51 (and v_52 v_53))) (let ((v_68 (not (a
nd v_49 (not (and v_50 v_51))))) (v_69 (not v_49)) (v_62 (and bool.b24 (not (and v_18 v_57))))) (let ((v_60 (not (and v_56 (not v_62))))) (let ((v_65 (and v_60 (not (and v_24 (
and v_58 (and v_59 v_60)))))) (v_63 (not (and v_56 v_62)))) (let ((v_64 (and v_63 (not (and v_47 (and v_48 v_63)))))) (let ((v_66 (not (and v_61 v_64))) (v_79 (not v_61)) (v_70
 (not v_64)) (v_71 (not v_65))) (let ((v_83 (not (and v_39 v_70)))) (let ((v_90 (and v_50 v_83)) (v_73 (and v_5 v_71))) (let ((v_76 (not (and v_40 v_73)))) (let ((v_88 (and v_5
2 v_76)) (v_74 (not (and v_72 v_73)))) (let ((v_86 (and v_54 v_74)) (v_81 (and bool.b17 v_45))) (let ((v_75 (not (and v_99 v_81)))) (let ((v_84 (and v_75 (not (and v_71 (and v_
74 (and v_75 v_76)))))) (v_80 (and v_78 v_79))) (let ((v_108 (not (and (not v_77) v_80))) (v_93 (not v_80)) (v_82 (not (and bool.b24 v_81)))) (let ((v_95 (and v_82 (not (and v_
70 (and v_82 v_83)))))) (let ((v_87 (and v_93 v_95))) (let ((v_85 (not v_87)) (v_92 (not v_84)) (v_89 (and v_84 v_87))) (let ((v_91 (and v_86 v_89)) (v_105 (and v_42 v_92))) (l
et ((v_107 (not v_105)) (v_130 (and v_94 (not v_95)))) (let ((v_106 (not v_130))) (let ((v_120 (and v_22 (and v_107 (and (and v_69 v_93) v_106)))) (v_125 (+ v_1 (* (/ 1 20) v_1
7)))) (let ((v_126 (not (<= v_125 241))) (v_97 (not (and bool.b24 v_56)))) (let ((v_98 (not (and v_97 (not (and bool.b19 (and v_48 v_97))))))) (let ((v_114 (and (not (and v_39
(not (and v_82 (not (and v_98 (and (not (and v_39 v_98)) v_82))))))) v_90)) (v_100 (not (and v_99 v_56)))) (let ((v_102 (not (and v_100 (not (and bool.b20 (and (and v_100 (not
(and v_40 (and bool.b20 v_5)))) (not (and bool.b20 v_101))))))))) (let ((v_103 (and v_5 v_102))) (let ((v_104 (and v_5 (not (and v_75 (not (and v_102 (and (not (and v_72 v_103)
) (and (not (and v_40 v_103)) v_75)))))))) (let ((v_112 (and (not (and v_40 v_104)) v_88)) (v_110 (and (not (and v_72 v_104)) v_86)) (v_111 (and v_93 v_106))) (let ((v_109 (no
t v_111)) (v_118 (not v_110)) (v_113 (and v_107 v_111)) (v_117 (not v_112))) (let ((v_115 (and v_110 v_113)) (v_116 (not v_114))) (let ((v_133 (not (and v_5 (and v_40 (not (and
 v_68 (not (and v_69 (not (and (not (and v_114 (not (and (not (and v_112 (not (and (not (and v_110 (not (and (not (and v_105 v_109)) (not (and v_107 (not (and v_108 v_109)))))
)) (not (and v_118 (not v_113)))))) (not (and v_117 (not v_115))))))) (not (and v_116 (not (and v_112 v_115)))))))))))))) (v_119 (not (and v_114 v_112))) (v_122 (<= v_1 20)))
(let ((v_135 (and v_122 (and v_121 v_105))) (v_124 (and v_122 (and v_123 v_105)))) (let ((v_143 (not v_124)) (v_128 (not (<= (* (- 1) v_127) (- 20)))) (v_138 (and bool.b17 v_38
)) (v_140 (and v_78 (not (and v_44 (not (and bool.b18 v_129))))))) (let ((v_134 (and (and (not (and v_128 v_138)) (not (and v_128 v_140))) (not (and v_128 v_130)))) (v_132 (and
 v_107 v_112))) (let ((v_131 (not v_132))) (let ((v_144 (and (not (and v_134 (not (and (not (and v_42 (not (and v_106 (and v_93 (and v_69 (and (not (and v_110 (not (and (not (a
nd v_114 (not (and v_131 (not (and v_105 v_117)))))) (not (and v_116 v_131)))))) (not (and v_118 (not (and v_114 v_132))))))))))) v_133)))) (not (and v_67 (not v_134))))) (v_13
6 (+ v_1 (* (/ 3 20) v_17))) (v_141 (<= v_1 45))) (let ((v_139 (and v_141 v_20)) (v_142 (not v_141))) (or (or (exists ((lambdaprime Real)) (let ((v_145 (* 1 (+ x3 (* (/ (- 1) 2
0) lambdaprime))))) (let ((v_146 (not (<= v_145 40))) (v_148 (* (- 1) (+ x4 (* 60 lambdaprime)))) (v_147 (not (<= v_145 33)))) (and (and (<= 0 lambdaprime) (<= lambdaprime lamb
da)) (not (and (and (not (and bool.b17 (not (and v_146 (and (not (<= v_148 (- 4100))) v_147)))))) (not (and bool.b18 (not (and v_146 (and v_147 (not (<= v_148 (- 4500)))))))) (
not (and bool.b19 (not (and v_146 (and v_147 (not (<= v_148 (- 4910)))))))))))) (< lambda 0)) (and (not (and v_96 (and (not (<= v_0 (/ 241 60))) (and (not (and v_42 (not (and
 v_11 (and v_10 (and v_6 (and (not (and v_8 (not (and (not (and v_3 (not (and v_2 (not (and bool.b20 bool.b22)))))) (not (and bool.b23 v_2)))))) (not (and bool.b21 (not (and v_
3 v_4)))))))))) (not (and v_5 (and (not (and (not (and v_6 (not (and (not (and v_3 (not (and (not (and v_7 (not (and (not (and v_8 (not (and (not (and v_9 (not (and v_12 (not
(and bool.b18 bool.b19)))))) (not (and bool.b20 v_12)))))) (not (and bool.b21 (not v_14)))))) (not (and bool.b22 (not v_15)))))) (not (and bool.b23 (not v_16)))))) (not (and
 bool.b17 (not (and v_3 v_16))))) v_40))))))) (not (and (and (not (and v_18 (not (and (not (and v_36 (not (and (not (and v_33 (not (and (not (and v_30 (not (and (not (and v_27
 (not (and v_25 (not (and bool.b19 v_23)))))) (not (and v_24 v_25)))))) (not (and v_26 (not v_31)))))) (not (and v_29 (not v_34)))))) (not (and v_32 (not v_37)))))) (not (an
d v_35 (not (and v_36 v_37))))) (not (and v_67 (not (and (and v_68 (not (and v_69 (not (and (not (and (not v_50) (not v_51))) (not (and v_50 (not (and (not (and (not v_52) (not
 v_53))) (not (and v_52 (not (and (not (and (not v_54) (not v_55))) (not (and v_54 (not (and (not (and v_65 (not (and v_66 (not (and v_79 v_70)))))) (not (and v_71 v_66)))))))
))))))))) (not (and v_67 (not (and (and v_68 (not (and v_69 (not (and (not (and v_90 (not (and (not (and v_88 (not (and (not (and v_86 (not (and (not (and v_84 (not (and v_10
8 v_85)))) (not (and v_92 v_85)))))) (not (and (not v_86) (not v_89))))))) (not (and (not v_88) (not v_91))))))) (not (and (not v_90) (not (and v_88 v_91))))))))) (not (and (no
t (and v_120 (and v_96 (and v_126 (and v_133 (not (and v_42 (not (and v_69 (and v_107 (and v_106 (and v_93 (and (not (and v_110 (not (and v_119 (not (and v_116 v_117)))))) (not
 (and v_118 v_119)))))))))))))) (not (and (not v_120) (and (not (and (not v_135) (not (and (not (and v_67 v_143)) (not (and v_124 (not (and (<= v_125 400) (and v_126 v_144))
```

Backup: Proving and Satisfying

- A formula is a theorem iff its negation is not satisfiable:

$$\models \Phi \quad \Leftrightarrow \quad \neg\Phi \textit{ is unsatisfiable}$$

- Theorem proving and satisfiability checking are dual

Backup: Case Splitting

- Linear programs (LPs) are easy to solve
- Piecewise-linear constraints are reducible to LPs
- Case Splitting:
  - Fix *each* ReLU to active or inactive state
  - Solve the resulting LP
  - If solution is found, we are done
  - Otherwise, backtrack and try other option
- State explosion: 300 ReLUs $\rightarrow 2^{300}$ checks

Backup: Soundness & Termination

- Soundness is straightforward
- Can we always find a solution using pivots and updates?
- No: sometimes get into a loop
- May have to *split* on ReLU variables
  - Do so *lazily*
  - In practice, about 10% of the ReLUs