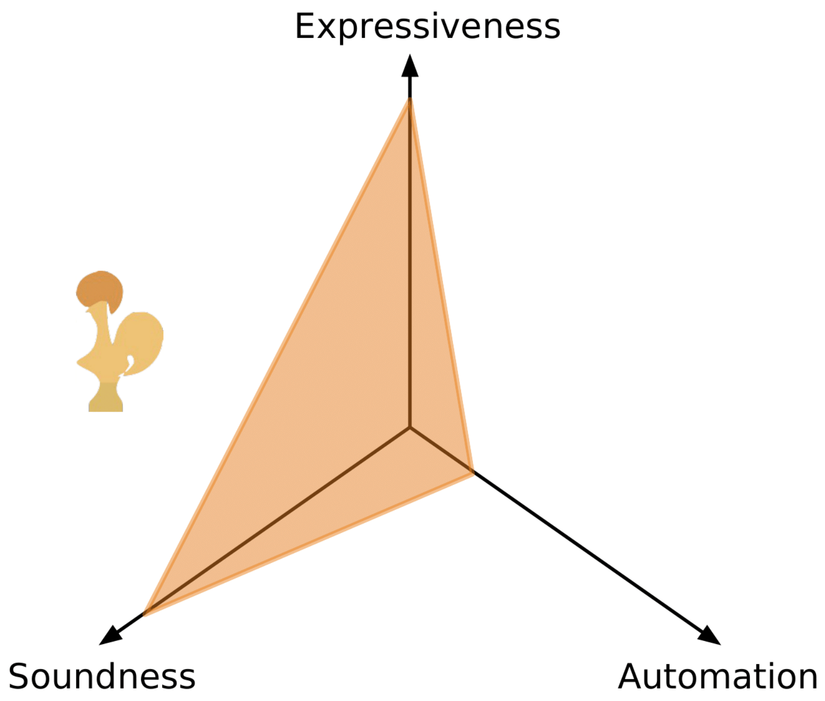
An Interactive SMT Tactic in Coq using Abductive Reasoning

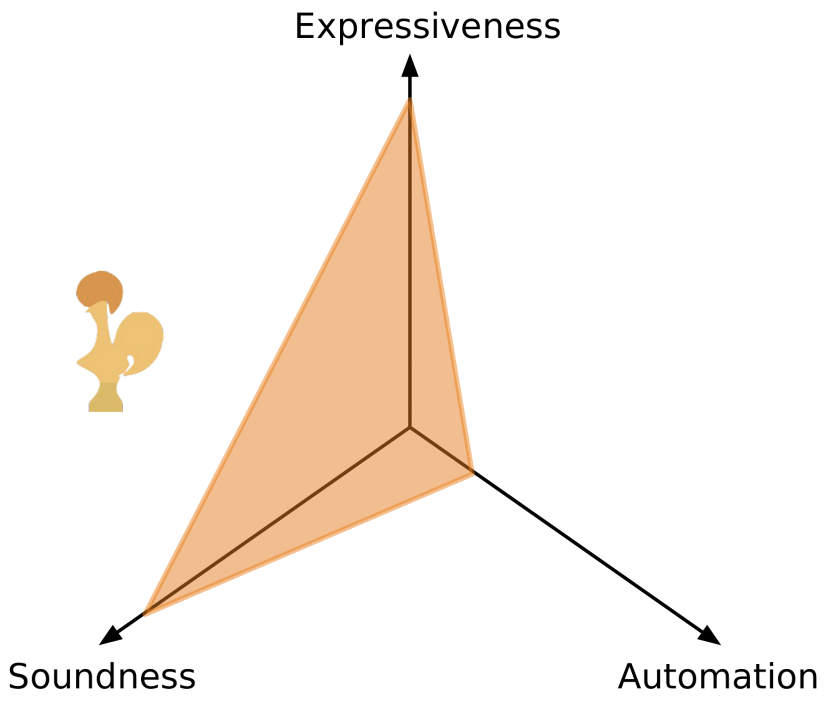
Haniel Barbosa, Chantal Keller, Andrew Reynolds,
Arjun Viswanathan, Cesare Tinelli, Clark Barrett

Proof Assistants



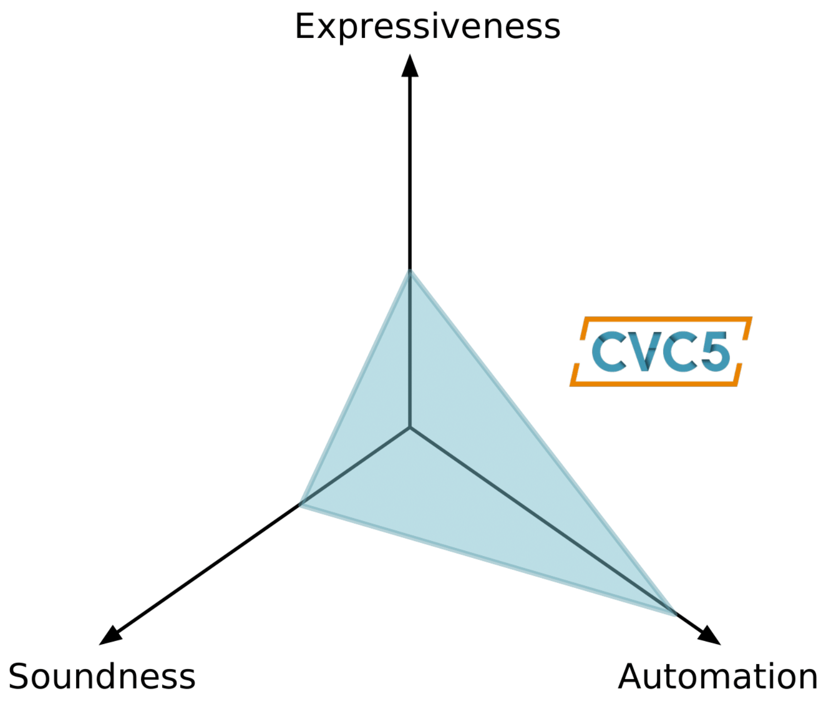
Proof Assistants

- Mechanized proofs
- Strong guarantees
- Trusted computing base
- Limited automation



SMT Solvers

- Mechanized proofs
- Strong guarantees
- Trusted computing base
- Limited automation



SMT Solvers

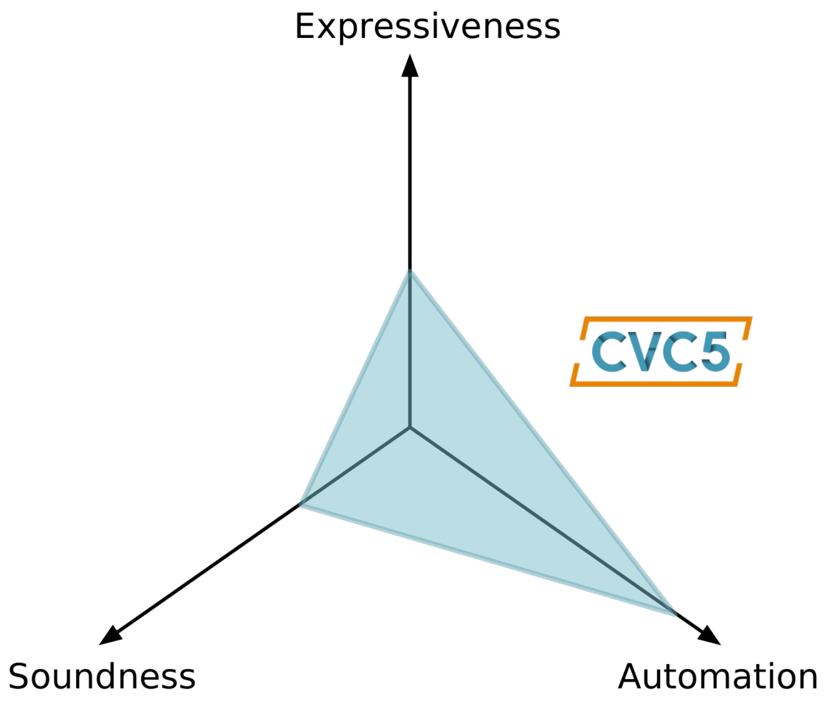
- Mechanized proofs
- Strong guarantees
- Trusted computing base
- Limited automation

Automated proofs

Vulnerable to bugs

Large code base

High automation



Can we do better?

Proof Assistants:

- Mechanized proofs
- Strong guarantees
- Trusted computing base
- Limited automation

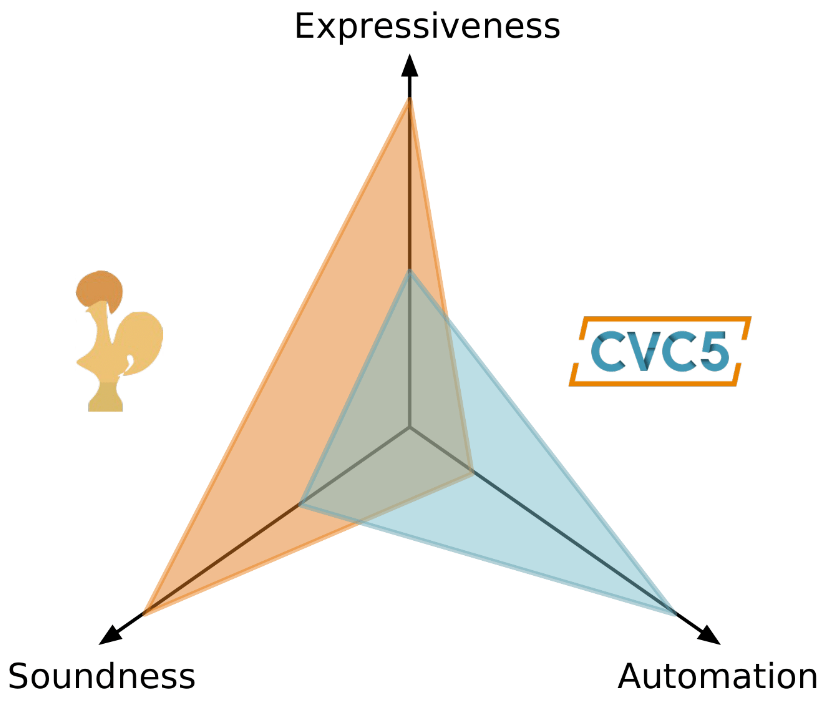
SMT Solvers:

Automated proofs

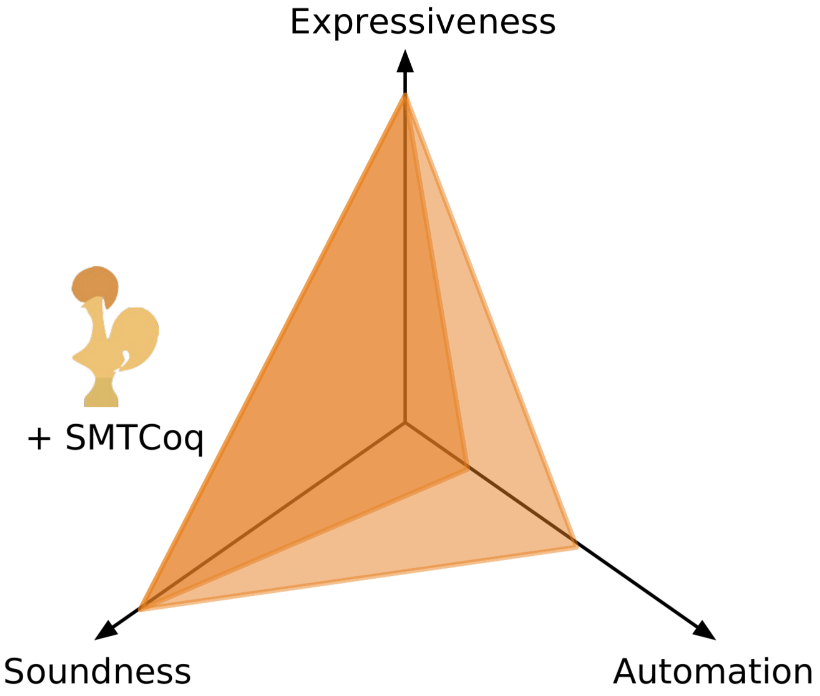
Vulnerable to bugs

Large code base

High automation

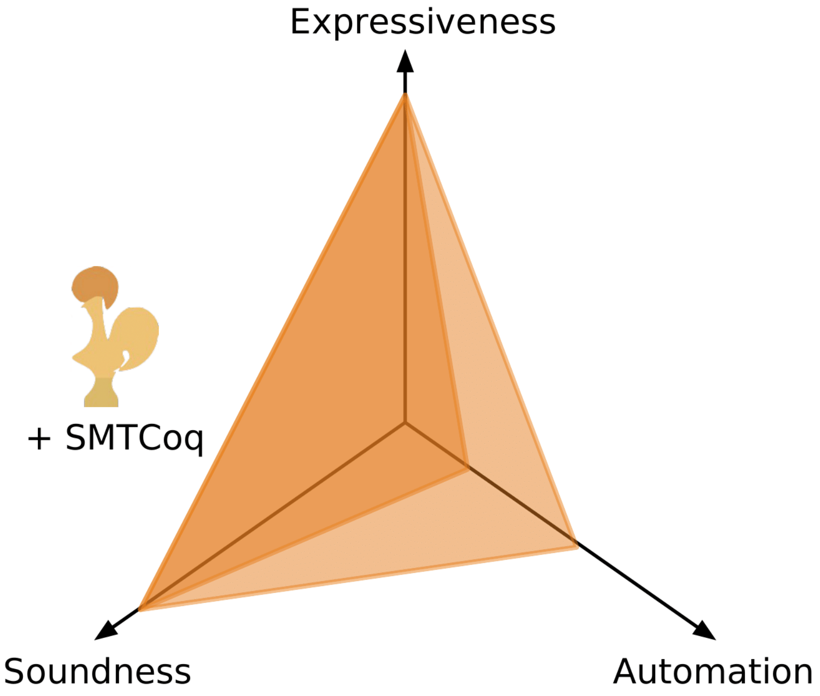


SMTCoq



SMTCoq

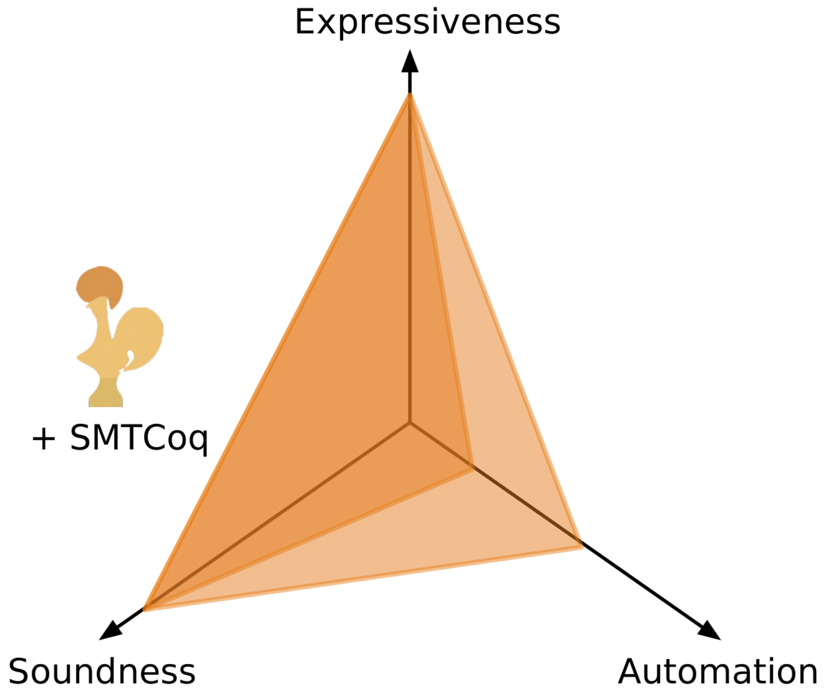
- Certified checker
- Automate subgoals
- Uncompromised trusted computing base



SMTCoq

- Certified checker
- Automate subgoals
- Uncompromised trusted computing base

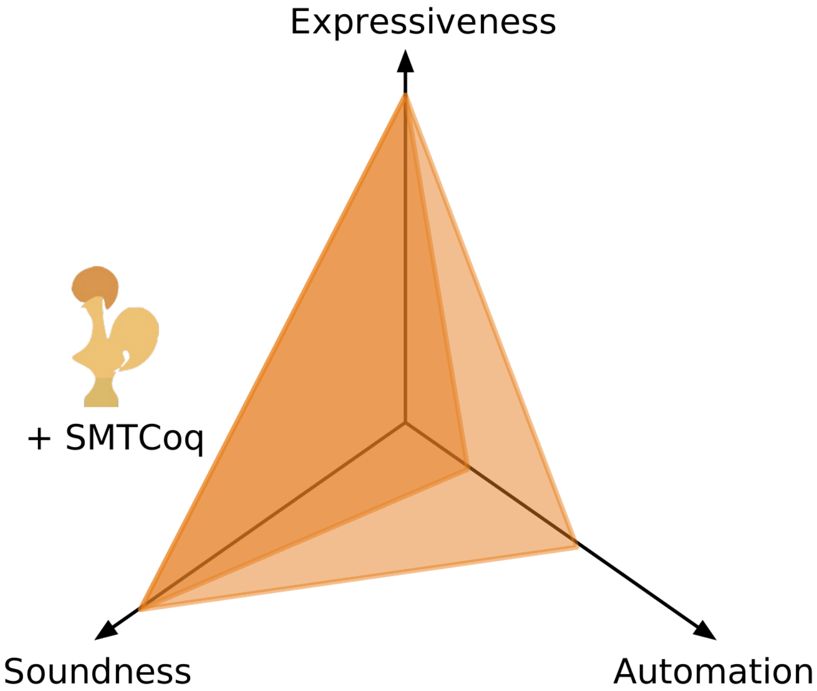
```
Goal forall (x y: Z) (f: Z → Z),
    x = y + 1 → f y = f (x − 1).
Proof.
  intros. rewrite H. rewrite Z.add_simpl_r.
  reflexivity.
Qed.
```



SMTCoq

- Certified checker
- Automate subgoals
- Uncompromised trusted computing base

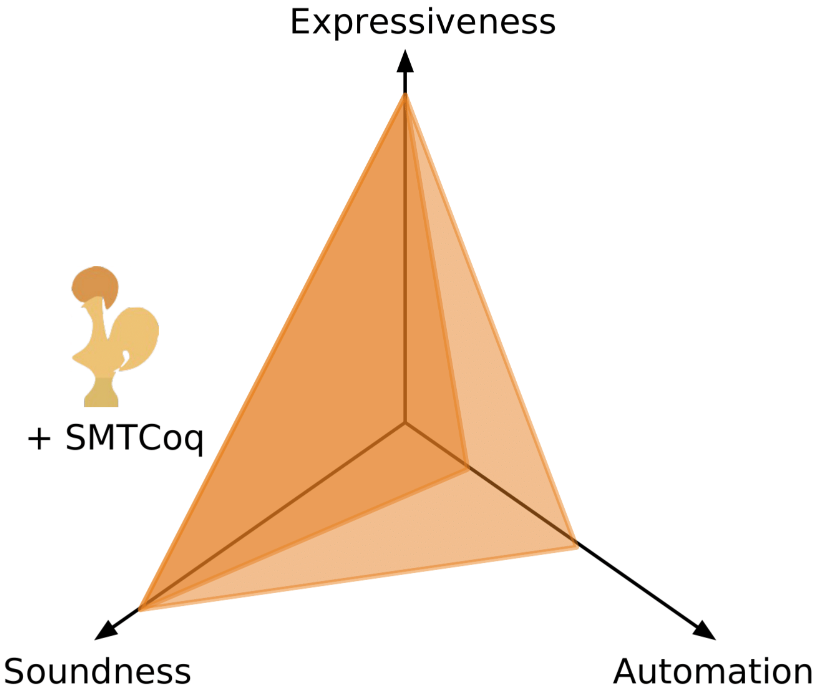
```
Goal forall (x y: Z) (f: Z → Z),
    x = y + 1 → f y = f (x − 1).
Proof. smt. Qed.
```



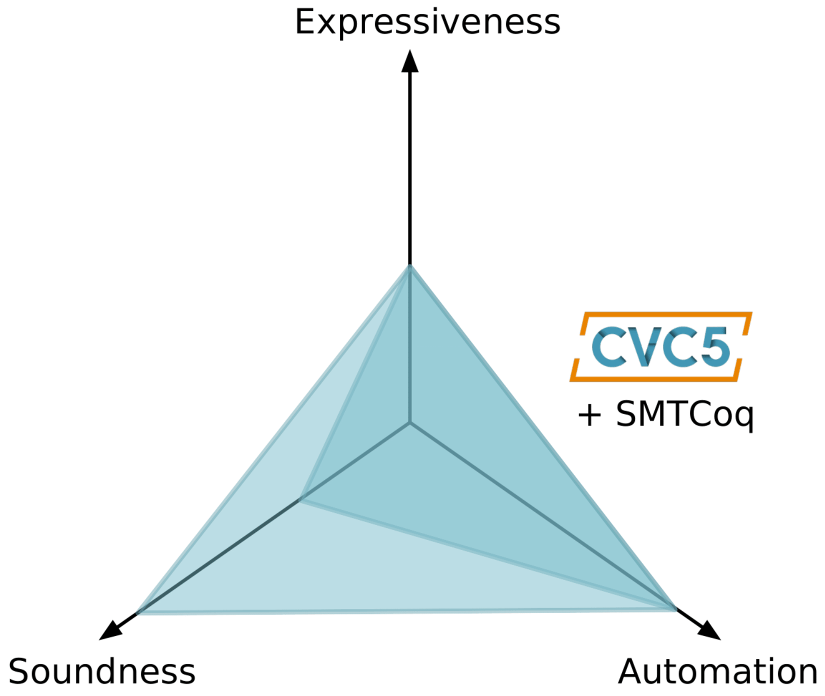
SMTCoq

- Certified checker
- Automate subgoals
- Uncompromised trusted computing base

```
Goal forall (x y: Z) (f: Z → Z),
    f y = f (x − 1).
Proof. smt.
(* Failure! Counter-example:
   x → 0
   y → 1
   f → fun x ⇒ if x = -1 then -2 else 2 *)
```

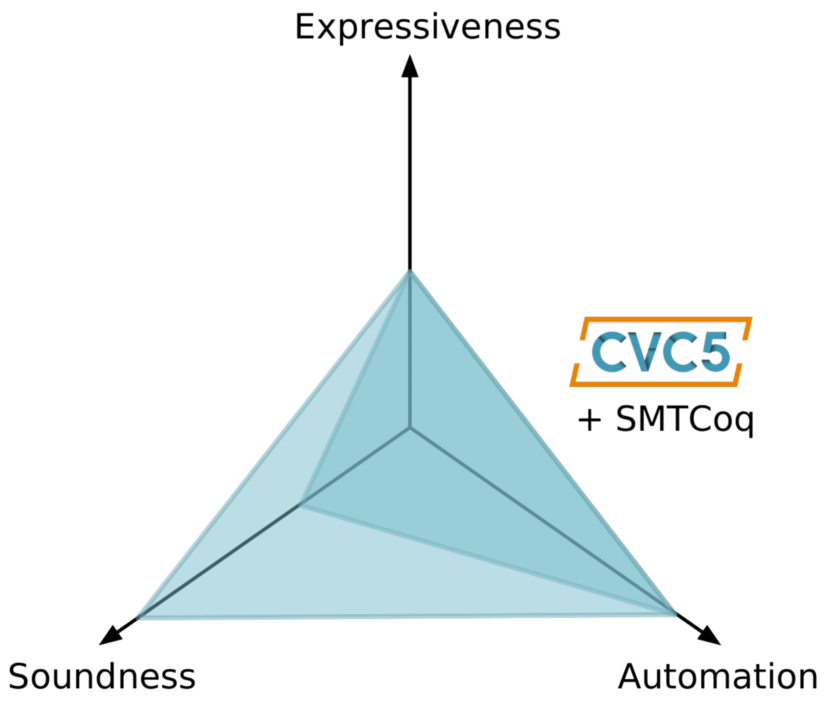


SMTCoq



SMTCoq

- Certified checker for SMT Proofs
- Implemented in Coq
- Proven correct in Coq



SMTCoq

- Solvers: zChaff, veriT, cvc5

SMTCoq

- Solvers: zChaff, veriT, cvc5
- Theories: EUF, LIA, BV, AX

SMTCoq

- Solvers: zChaff, veriT, cvc5
- Theories: EUF, LIA, BV, AX

```
Goal forall (a b : bool) (x y : Z),
      (ifb a
           (ifb b (2*x + 1 =? 2*y + 1) (2*x + 1 =? 2*y))
           (ifb b (2*x =? 2*y + 1) (2*x =? 2*y)))
      --->
      ((a ---> b) && (b ---> a) && (x =? y)).
Proof. smt. Qed.
```

```
Goal forall (x y : Z), x = y + 1 → x * x = (y + 1) * x.
Proof. smt.
```

```
Goal forall (x y : Z), x = y + 1 -> x * x = (y + 1) * x.
Proof. smt.
(* Solver error: A non-linear fact was asserted
   to arithmetic in a linear logic. *)
```

```
Goal forall (x y : Z), x = y + 1 -> x * x = (y + 1) * x.
Proof. smt.
(* Solver error: A non-linear fact was asserted
   to arithmetic in a linear logic. *)

Definition mul' := Z.mul.
Notation "x *' y" := (mul' x y).
```

```
Goal forall (x y : Z), x = y + 1 -> x * x = (y + 1) * x.
Proof. smt.
(* Solver error: A non-linear fact was asserted
   to arithmetic in a linear logic. *)

Definition mul' := Z.mul.
Notation "x *' y" := (mul' x y).
Goal forall (x y : Z), x = y + 1 -> x *' x = (y + 1) *' x.
Proof. smt. Qed.
```

```
Goal forall (x y : Z), x = y + 1 -> x * x = (y + 1) * x.
Proof. smt.
(* Solver error: A non-linear fact was asserted
   to arithmetic in a linear logic. *)

Definition mul' := Z.mul.
Notation "x *' y" := (mul' x y).
Goal forall (x y : Z), x = y + 1 -> x *' x = (y + 1) *' x.
Proof. smt. Qed.

Goal forall (x y z: Z), x = y + 1 -> y *' z = z *' (x - 1).
Proof. smt.
```

```
Goal forall (x y : Z), x = y + 1 → x * x = (y + 1) * x.
Proof. smt.
(* Solver error: A non-linear fact was asserted
   to arithmetic in a linear logic. *)

Definition mul' := Z.mul.
Notation "x *' y" := (mul' x y).
Goal forall (x y : Z), x = y + 1 → x *' x = (y + 1) *' x.
Proof. smt. Qed.

Goal forall (x y z: Z), x = y + 1 → y *' z = z *' (x − 1).
Proof. smt.
(* Failure! Counter-example:
   x → 0, y → -1, z → 1,
   mul' → fun x y ⇒ if x = 1 then if y = -1 then -2
                    else 2 else 2 *)
```

The `abduce` Tactic

- Present abducts that entail the goal
- Uses abductive reasoning by cvc5

The `abduce` Tactic

- Present abducts that entail the goal
- Uses abductive reasoning by cvc5

```
Goal forall (x y z: Z), x = y + 1 → y *' z = z *' (x − 1).
Proof. (* smt. Failure! *) abduce 3.
```

The abduce Tactic

- Present abducts that entail the goal
- Uses abductive reasoning by cvc5

```
Goal forall (x y z: Z), x = y + 1 → y *' z = z *' (x − 1).
Proof. (* smt. Failure! *) abduce 3.
(* cvc5 returned SAT.
   The solver cannot prove the goal, but one
   of the following hypotheses would make it provable:
   y = z
   -1 + x = z
   (mul' z y) = (mul' y z) *)
```

The `abduce` Tactic

- Present abducts that entail the goal
- Uses abductive reasoning by cvc5

```
Goal forall (x y z: Z), x = y + 1 → y *' z = z *' (x - 1).
Proof. (* smt. Failure! abduce 3. *)
  assert ((mul' z y) = (mul' y z)).
  { apply Z.mul_comm. } smt.
Qed.
```

Abduction

- Find A such that
  - $H_1, \ldots, H_n \not\models_T G$
  - $H_1, \ldots, H_n, A \models_T G$

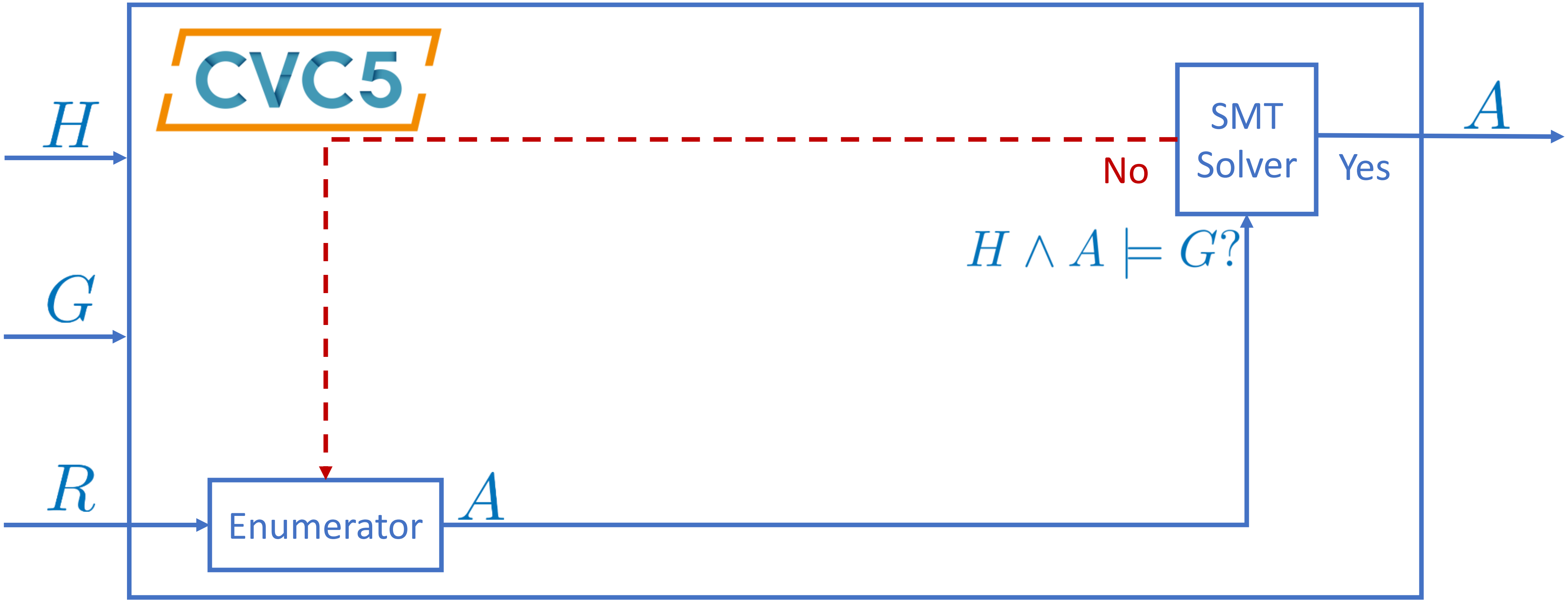
Abduction

- Find A such that
  - $H_1, \ldots, H_n \not\models_T G$
  - $H_1, \ldots, H_n, A \models_T G$
  - $H_1 \wedge \cdots \wedge H_n \wedge A$ is T-satisfiable

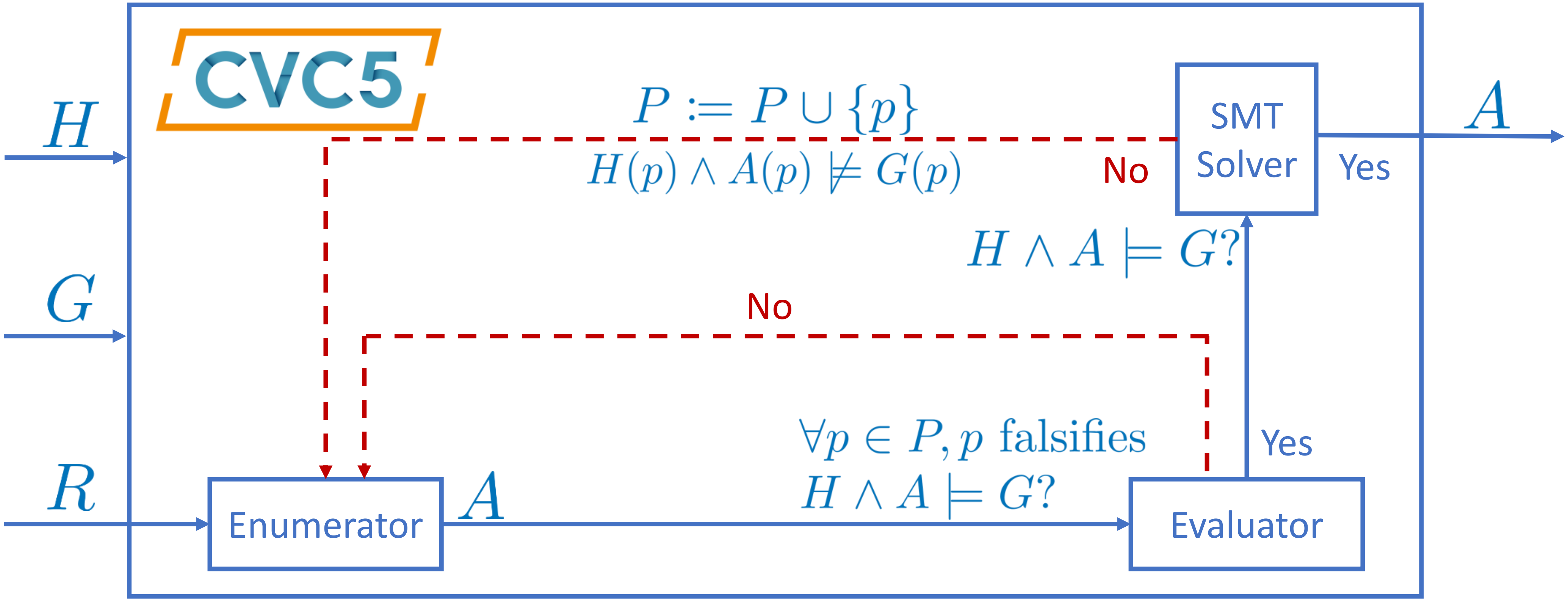
Abduction

- Find A such that
  - $H_1, \ldots, H_n \not\models_T G$
  - $H_1, \ldots, H_n, A \models_T G$
  - $H_1 \wedge \cdots \wedge H_n \wedge A$ is T-satisfiable
  - A is generated by grammar R

Abduction in cvc5 via SyGuS

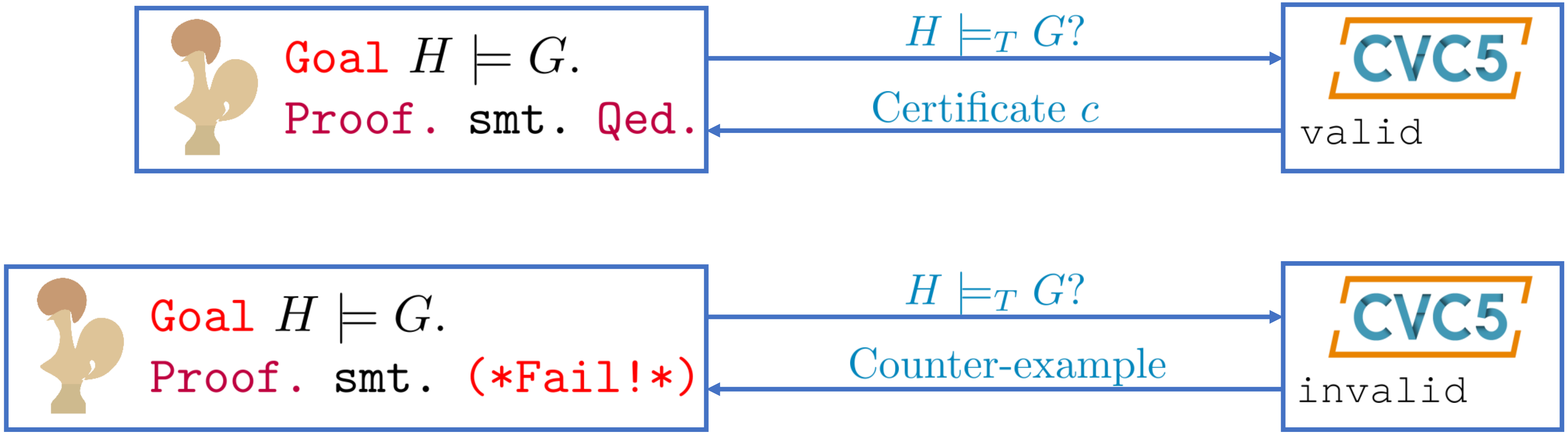


Abduction in cvc5 via SyGuS

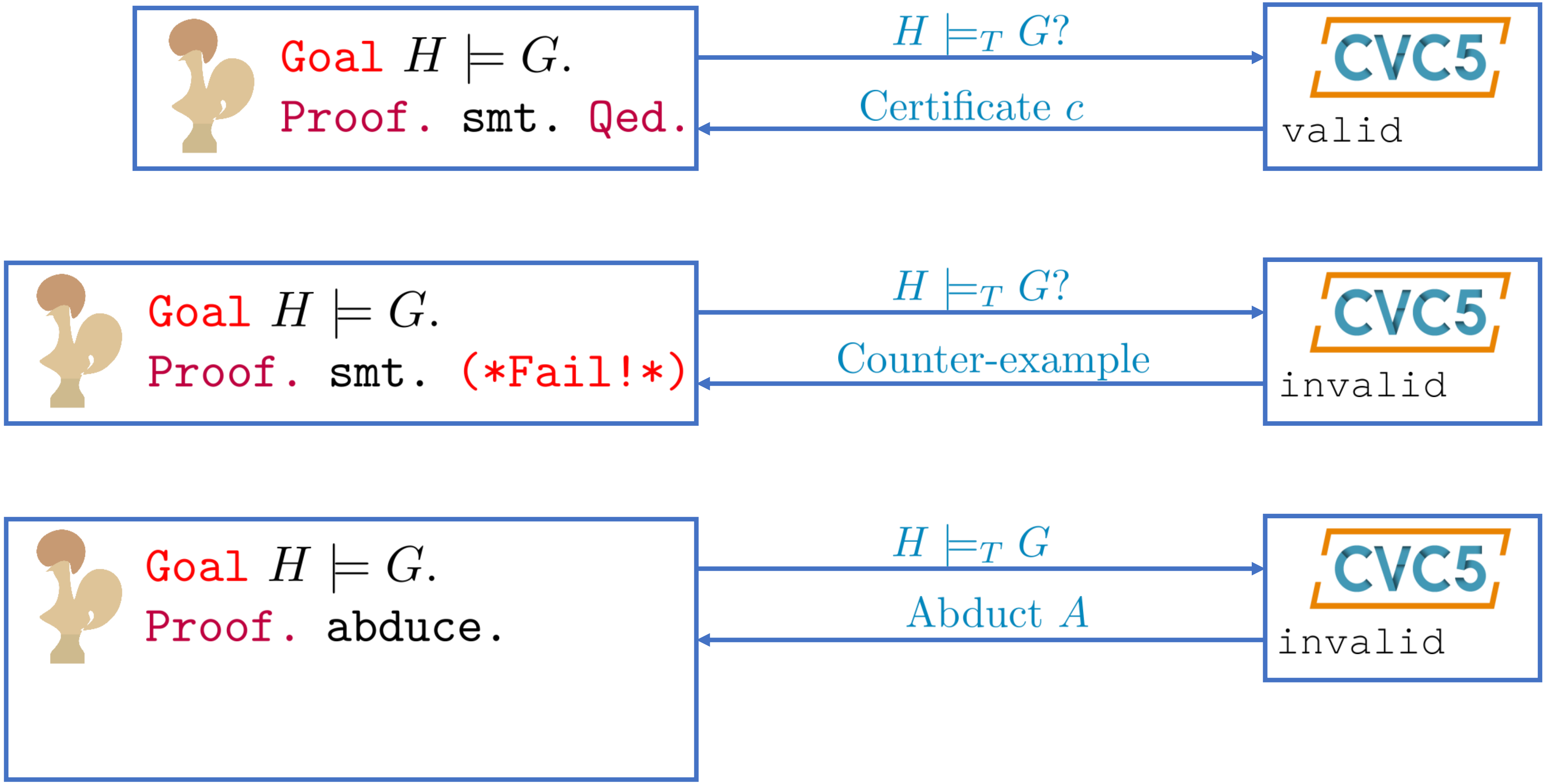


abduce Tactic

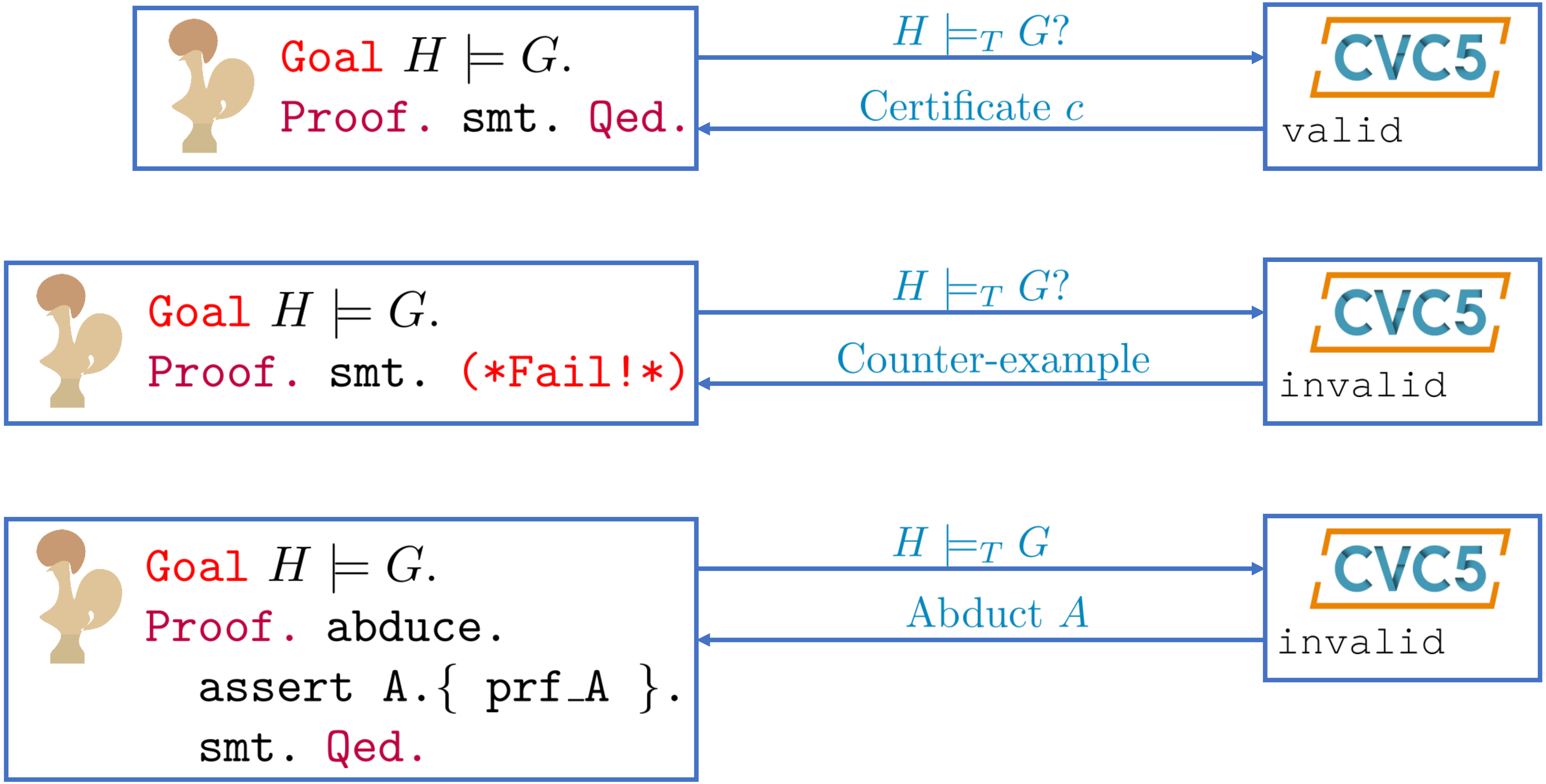
abduce Tactic



abduce Tactic



abduce Tactic



Evaluation

- On Zorder Coq library

Goals	`smt` Successes	Returns cex	`abduce` Successes	Timeouts
59	33	26	13	13

Evaluation

- On Zorder Coq library
- Successor (Z.succ) and predecessor (Z.pred) and functions.

Goals	`smt` Successes	Returns cex	`abduce` Successes	Timeouts
59	33	26	13	13

Evaluation

```
Lemma Zle_gt_succ n m : n <= m -> Z.succ m > n.
Proof. smt.
```

```
CVC4 returned sat. Here is the model:

n := 0
m := 0
Z.succ := fun _ => 0
```

Evaluation

```
Lemma Zle_gt_succ n m : n <= m -> Z.succ m > n.
Proof. smt.
```

```
CVC4 returned sat. Here is the model:

n := 0
m := 0
Z.succ := fun _ => 0
```

```
Lemma Zle_gt_succ n m : n <= m -> Z.succ m > n.
Proof. abduce 3.
```

```
cvc5 returned SAT.
The solver cannot prove the goal, but one
 of the following hypotheses would make it provable:
(Z.succ m) = 1 + m
(Z.succ m) = n + 1
n + 1 <= (Z.succ m)
```

Evaluation

```
Lemma Zle_gt_succ n m : n <= m -> Z.succ m > n.
Proof. smt.
```

```
CVC4 returned sat. Here is the model:

n := 0
m := 0
Z.succ := fun _ => 0
```

```
Lemma Zle_gt_succ n m : n <= m -> Z.succ m > n.
Proof. abduce 3.
```

```
cvc5 returned SAT.
The solver cannot prove the goal, but one
 of the following hypotheses would make it provable:
(Z.succ m) = 1 + m
(Z.succ m) = n + 1
n + 1 <= (Z.succ m)
```

```
Lemma Zle_gt_succ n m : n <= m -> Z.succ m > n.
Proof. (* abduce 3. *)
  assert ((Z.succ m) = 1 + m). { unfold Z.succ. smt. }
  smt.
Qed.
```

Future Directions

Future Directions

- Evaluation inside larger proofs

Future Directions

- Evaluation inside larger proofs
- Control abducts by controlling SyGuS grammar

Future Directions

- Evaluation inside larger proofs
- Control abducts by controlling SyGuS grammar
- Automatically prove entailed abducts

Acknowledgements

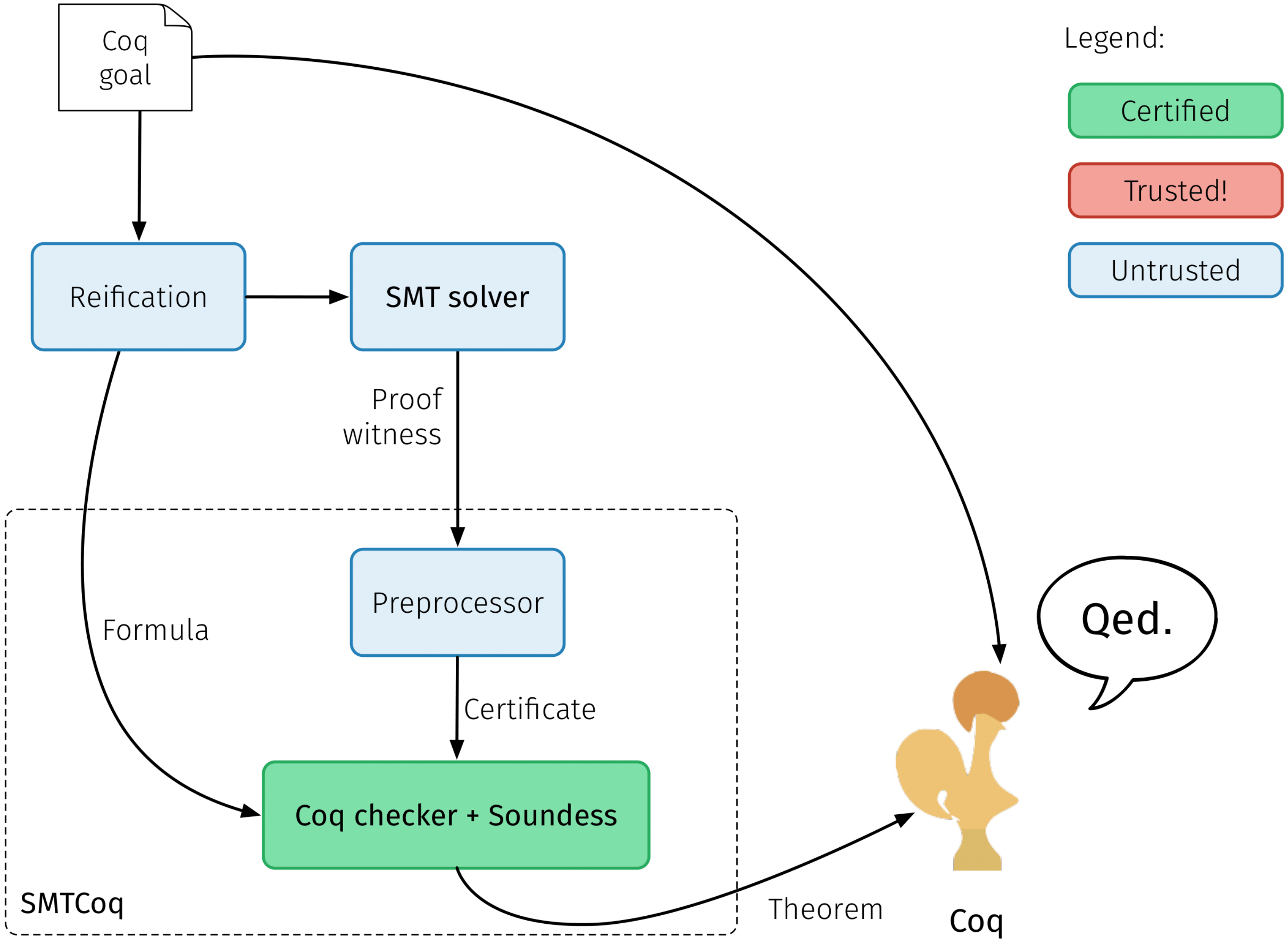
- https://smtcoq.github.io/

- Scalable Algorithms for Abduction via Enumerative Syntax-Guided Synthesis. *IJCAR 2020. Andrew Reynolds, Haniel Barbosa, Daniel Larraz, and Cesare Tinelli*

- Images borrowed from slides presented by Alain Mebsout at CAV 2017 – “SMTCoq: A plug-in for integrating SMT solvers into Coq"

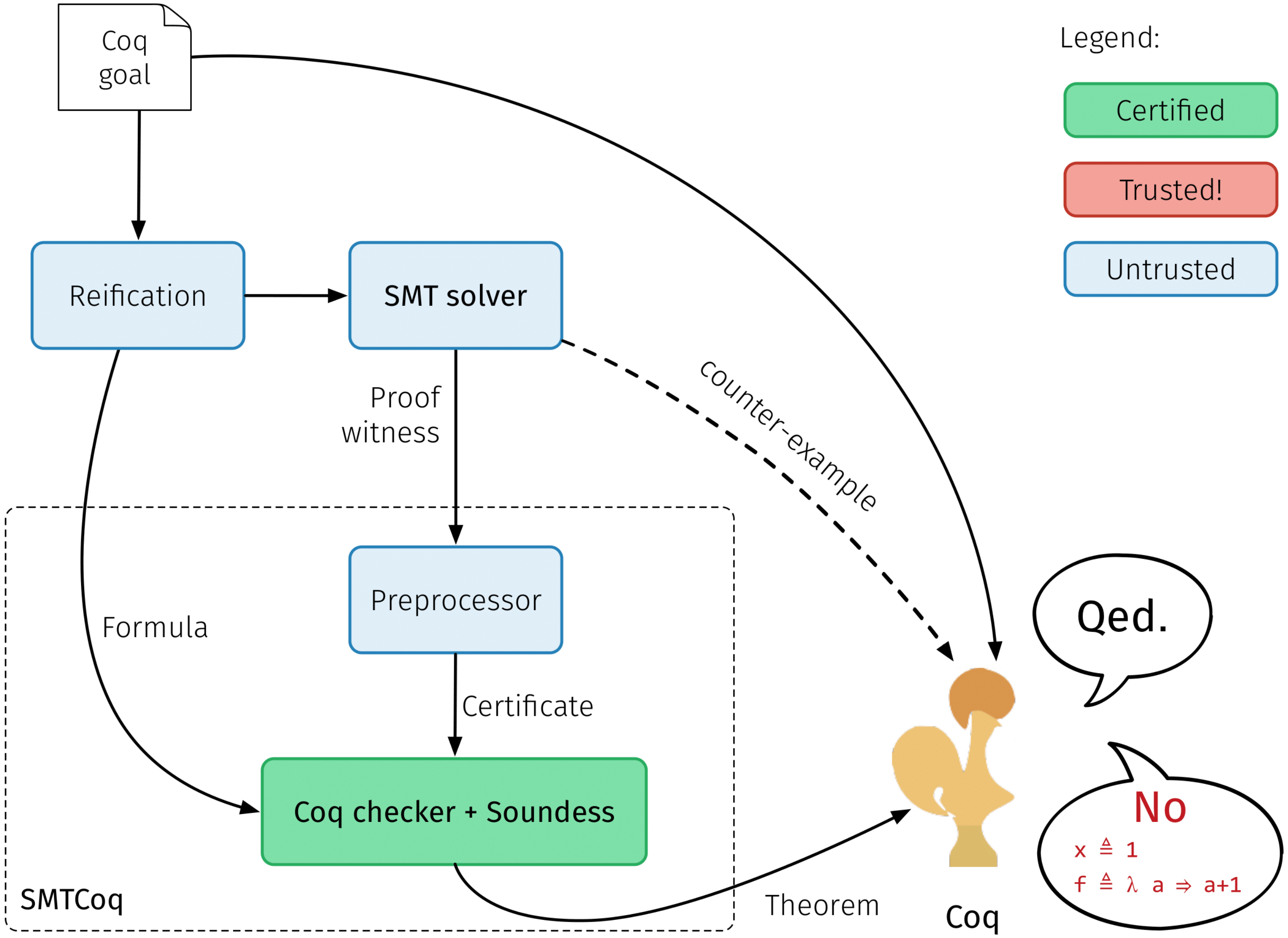
Backup Slides

SMTCoq



Coq
goal
Legend:
Certified
Trusted!
Untrusted
Reification
SMT solver
Proof
witness
Preprocessor
Formula
Certificate
Coq checker + Soundess
SMTCoq
Theorem
Qed.
Coq

SMTCoq



Coq goal
Legend:
Certified
Trusted!
Untrusted
Reification
SMT solver
Proof witness
counter-example
Formula
Preprocessor
Certificate
Coq checker + Soundess
SMTCoq
Theorem
Coq
Qed.
No
x ≜ 1
f ≜ λ a ⇒ a+1