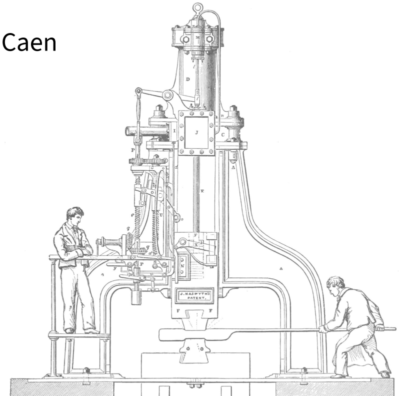
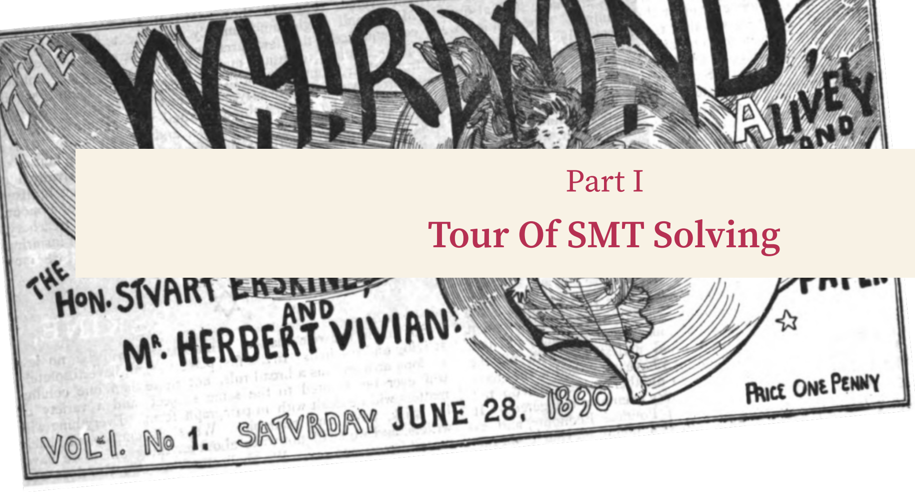
Understanding SMT Unsatisfiability Proofs

Hans-Jörg Schurr
CODAG Seminar – Université de Caen
14.01.2025

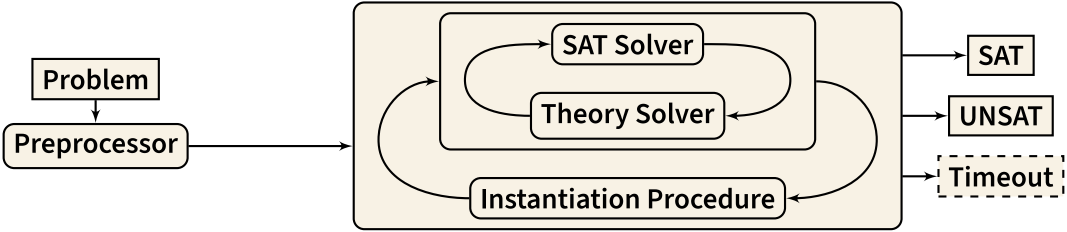


Part I

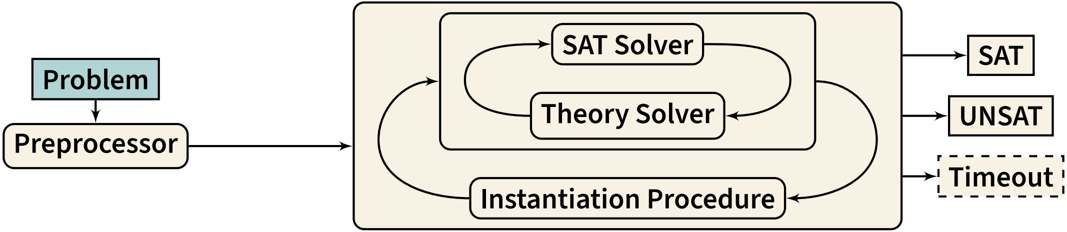
Tour Of SMT Solving



SMT Solving As A Diagram



SMT Solving As A Diagram



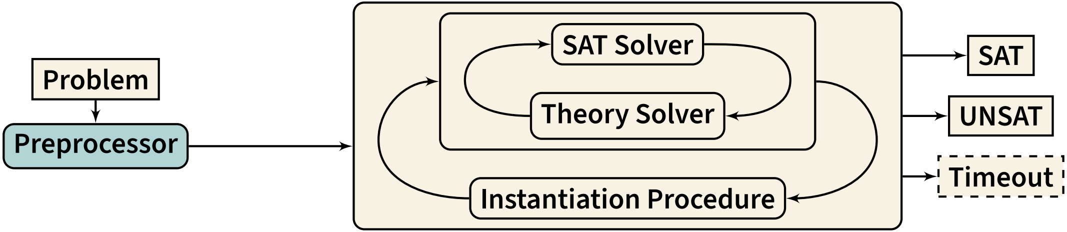
An Example: Problem Specification

1. We produce 1L, 2L, and 3L bottles.
2. The price of a bottle is the volume plus four times the wall thickness (in mm).
3. The price must be less than 4\$.
4. If the new machine is broken, we cannot produce 3L bottles, and the wall thickness must be more than 1mm.
5. The new machine is broken.
6. For all bottle sizes, the wall thickness in millimetre can at most be the volume in liters.

An Example: Problem Specification

1. We produce 1L, 2L, and 3L bottles.	1. $v = 1 \lor v = 2 \lor v = 3$
2. The price of a bottle is the volume plus four times the wall thickness (in mm).	2. $v + 2t < p$
3. The price must be less than 4\$.	3. $p = 4$
4. If the new machine is broken, we cannot produce 3L bottles, and the wall thickness must be more than 1mm.	4. $b \rightarrow (v \neq 3 \land t > 1)$
5. The new machine is broken.	5. b
6. For all bottle sizes, the wall thickness in millimetre can at most be the volume in liters.	6. $\forall z. v = z \rightarrow t \leq z$

SMT Solving As A Diagram



An Example: Preprocessing

1. $v = 1 \vee v = 2 \vee v = 3$
2. $v + 2t < p$
3. $p = 4$
4. $b \rightarrow (\neg v = 3 \wedge t > 1)$

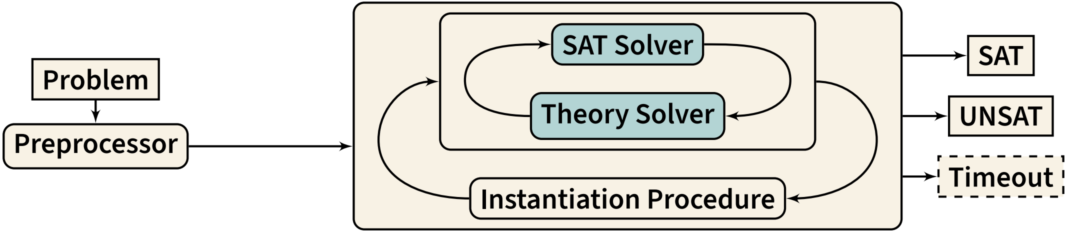
5. b
6. $\forall z.\, v = z \rightarrow t \leq z$

An Example: Preprocessing

1. $v = 1 \lor v = 2 \lor v = 3$
2. $v + 2t < p$
3. $p = 4$
4. $b \to (\neg v = 3 \land t > 1)$
5. b
6. $\forall z. v = z \to t \leq z$

1. $v = 1 \lor v = 2 \lor v = 3$
2. $v + 2t < 4$
3.
4. $\neg b \lor \neg v = 3$
   $\neg b \lor 1 < t$
5. b
6. $\forall z. \neg v = z \lor \neg(z < t)$

SMT Solving As A Diagram



An Example: The Ground Solver

- $v = 1 \vee v = 2 \vee v = 3$
- $v + 2t < 4$
- $\neg b \vee \neg v = 3$
- $\neg b \vee 1 < t$
- b
- $\forall z. \neg v = z \vee \neg(z < t)$

An Example: The Ground Solver

- $v = 1 \lor v = 2 \lor v = 3$
- $v + 2t < 4$
- $\neg b \lor \neg v = 3$
- $\neg b \lor 1 < t$
- b
- ~~$\forall z. \neg v = z \lor \neg(z < t)$~~

SAT Problem

- $p_1 \lor p_2 \lor p_3$
- p_4
- $\neg b \lor \neg p_3$
- $\neg b \lor p_5$
- b

An Example: The Ground Solver

- $v = 1 \lor v = 2 \lor v = 3$
- $v + 2t < 4$
- $\neg b \lor \neg v = 3$
- $\neg b \lor 1 < t$
- b
- ~~$\forall z. \neg v = z \lor \neg(z < t)$~~

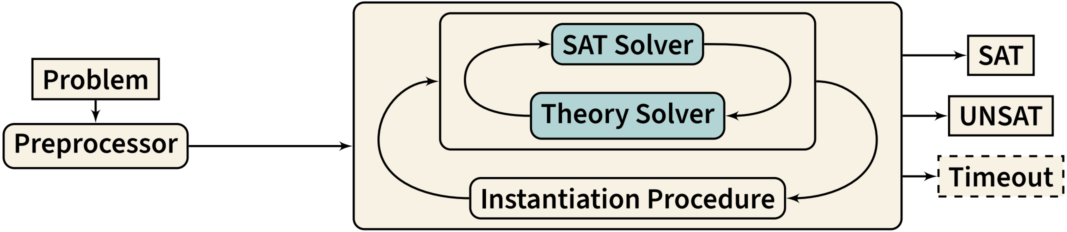
SAT Problem

- $p_1 \lor p_2 \lor p_3$
- p_4
- $\neg b \lor \neg p_3$
- $\neg b \lor p_5$
- b

Theory Literals

- $p_1 := v = 1, p_2 := v = 2, p_3 := v = 3$
- $p_4 := v + 2t < 4$
- $p_5 := t > 1$

SMT Solving As A Diagram



An Example: The SAT Solver and the Theory Solver

SAT Problem

- $p_1 \vee p_2 \vee p_3$
- p_4
- $\neg b \vee \neg p_3$
- $\neg b \vee p_5$
- b

Theory Literals

- $p_1 := v = 1, p_2 := v = 2, p_3 := v = 3$
- $p_4 := v + 2t < 4$
- $p_5 := t > 1$

An Example: The SAT Solver and the Theory Solver

SAT Problem

- $p_1 \vee p_2 \vee p_3$
- p_4
- $\neg b \vee \neg p_3$
- $\neg b \vee p_5$
- b

Theory Literals

- $p_1 := v = 1, p_2 := v = 2, p_3 := v = 3$
- $p_4 := v + 2t < 4$
- $p_5 := t > 1$

SAT Solver

I pick b, p_2, p_4, and p_5 ☺️

An Example: The SAT Solver and the Theory Solver

SAT Problem

- $p_1 \vee p_2 \vee p_3$
- p_4
- $\neg b \vee \neg p_3$
- $\neg b \vee p_5$
- b

Theory Literals

- $p_1 := v = 1, p_2 := v = 2, p_3 := v = 3$
- $p_4 := v + 2t < 4$
- $p_5 := t > 1$

SAT Solver

I pick b, p_2, p_4, and p_5 😊

Linear Arithmetic Solver

1. I get $v = 2$, $v + 2t < 4$, and $t > 1$

An Example: The SAT Solver and the Theory Solver

SAT Problem

- $p_1 \lor p_2 \lor p_3$
- p_4
- $\neg b \lor \neg p_3$
- $\neg b \lor p_5$
- b

Theory Literals

- $p_1 := v = 1, p_2 := v = 2, p_3 := v = 3$
- $p_4 := v + 2t < 4$
- $p_5 := t > 1$

SAT Solver

I pick b, p_2, p_4, and p_5 😊

Linear Arithmetic Solver

1. I get $v = 2$, $v + 2t < 4$, and $t > 1$
2. Doesn't work:
   $\neg v = 2 \lor \neg(v + 2t < 4) \lor \neg t > 1$ 🤔

An Example: The SAT Solver and the Theory Solver

SAT Problem

- $p_1 \vee p_2 \vee p_3$
- p_4
- $\neg b \vee \neg p_3$
- $\neg b \vee p_5$
- b

Theory Literals

- $p_1 := v = 1, p_2 := v = 2, p_3 := v = 3$
- $p_4 := v + 2t < 4$
- $p_5 := t > 1$

SAT Solver

I pick b, p_2, p_4, and p_5 😊

Linear Arithmetic Solver

1. I get $v = 2$, $v + 2t < 4$, and $t > 1$
2. Doesn't work:
   $\neg v = 2 \vee \neg(v + 2t < 4) \vee \neg t > 1$ 🤔

SAT Solver

I have to pick b, p_1, p_4, and p_5 🤞

An Example: The SAT Solver and the Theory Solver

SAT Problem

- $p_1 \lor p_2 \lor p_3$
- p_4
- $\neg b \lor \neg p_3$
- $\neg b \lor p_5$
- b

Theory Literals

- $p_1 := v = 1, p_2 := v = 2, p_3 := v = 3$
- $p_4 := v + 2t < 4$
- $p_5 := t > 1$

SAT Solver

I pick b, p_2, p_4, **and** p_5 ☺️

Linear Arithmetic Solver

1. I get $v = 2$, $v + 2t < 4$, **and** $t > 1$
2. Doesn't work:
   $\neg v = 2 \lor \neg(v + 2t < 4) \lor \neg t > 1$ 🤔

SAT Solver

I have to pick b, p_1, p_4, **and** p_5 🤞

Linear Arithmetic Solver

1. I get $v = 1$, $v + 2t < 4$, **and** $t > 1$

An Example: The SAT Solver and the Theory Solver

SAT Problem

- $p_1 \lor p_2 \lor p_3$
- p_4
- $\neg b \lor \neg p_3$
- $\neg b \lor p_5$
- b

Theory Literals

- $p_1 := v = 1, p_2 := v = 2, p_3 := v = 3$
- $p_4 := v + 2t < 4$
- $p_5 := t > 1$

SAT Solver

I pick b, p_2, p_4, and p_5 ☺️

Linear Arithmetic Solver

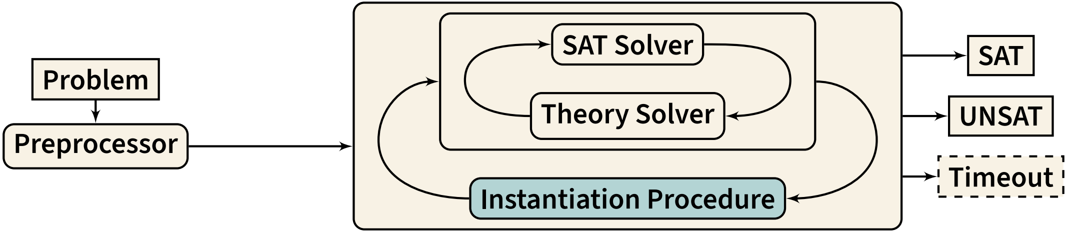
1. I get $v = 2$, $v + 2t < 4$, and $t > 1$
2. Doesn't work:
   $\neg v = 2 \lor \neg(v + 2t < 4) \lor \neg t > 1$ 🤔

SAT Solver

I have to pick b, p_1, p_4, and p_5 🤞

Linear Arithmetic Solver

1. I get $v = 1$, $v + 2t < 4$, and $t > 1$
2. That works! 🎉


Problem
Preprocessor
SAT Solver
Theory Solver
Instantiation Procedure
SAT
UNSAT
Timeout

An Example: Quantifier Instantiation

SAT Problem

- $p_1 \vee p_2 \vee p_3$
- p_4
- $\neg b \vee \neg p_3$
- $\neg b \vee p_5$
- b

Theory Literals

- $p_1 := v = 1, p_2 := v = 2, p_3 := v = 3$
- $p_4 := v + 2t < 4$
- $p_5 := t > 1$

Instantiation Procedure

- I have $\forall z. \neg v = z \vee \neg z < t$

An Example: Quantifier Instantiation

SAT Problem

- $p_1 \lor p_2 \lor p_3$
- p_4
- $\neg b \lor \neg p_3$
- $\neg b \lor p_5$
- b

Theory Literals

- $p_1 := v = 1, p_2 := v = 2, p_3 := v = 3$
- $p_4 := v + 2t < 4$
- $p_5 := t > 1$

Instantiation Procedure

- I have $\forall z. \neg v = z \lor \neg z < t$
- What happens if I pick $z \leftarrow 1$? 😈

An Example: Quantifier Instantiation

SAT Problem

- $p_1 \lor p_2 \lor p_3$
- p_4
- $\neg b \lor \neg p_3$
- $\neg b \lor p_5$
- b

Theory Literals

- $p_1 := v = 1, p_2 := v = 2, p_3 := v = 3$
- $p_4 := v + 2t < 4$
- $p_5 := t > 1$

Instantiation Procedure

- I have $\forall z. \neg v = z \lor \neg z < t$
- What happens if I pick $z \leftarrow 1$? 😈
- That's $\neg v = 1 \lor \neg t > 1$

An Example: Quantifier Instantiation

SAT Problem

- $p_1 \lor p_2 \lor p_3$
- p_4
- $\neg b \lor \neg p_3$
- $\neg b \lor p_5$
- b

Theory Literals

- $p_1 := v = 1, p_2 := v = 2, p_3 := v = 3$
- $p_4 := v + 2t < 4$
- $p_5 := t > 1$

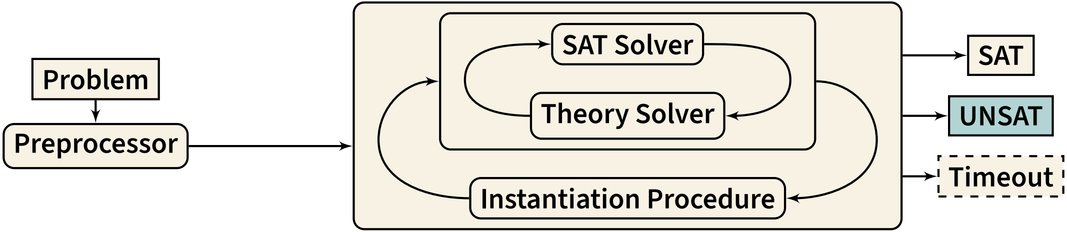
Instantiation Procedure

- I have $\forall z. \neg v = z \lor \neg z < t$
- What happens if I pick $z \leftarrow 1$? 😈
- That's $\neg v = 1 \lor \neg t > 1$

SAT Solver

- That's $\neg p_1 \lor \neg p_5$
- Oh no 😢

SMT Solving As A Diagram



Using SMT-LIB

```
(set-logic LRA)
(declare-const v Real) (declare-const t Real)
(declare-const b Bool)
(assert (or (= v 1) (= v 2) (= v 3)))
(assert (< (+ v (* 2 t)) p))
(assert (= p 4))
(assert (=> b (and (not (= v 3)) (> t 1))))
(assert b)
(assert (forall ((z Real)) (=> (= v z) (<= t z))))
(check-sat)
```

- 👩‍✈️ Most SMT solvers support SMT-LIB
- 🧮 Theories: arithmetic, arrays, data-types, bit-vectors, strings, ...
- 🏅 Yearly competition (SMT-COMP)
- 📚 Large benchmark library

Some Solvers You Can Try (a Biased List)

Some Solvers You Can Try (a Biased List)

veriT

- Small solver
- Excellent proofs, good quantifier support
- `www.verit-solver.org`

Some Solvers You Can Try (a Biased List)

veriT

- Small solver
- Excellent proofs, good quantifier support
- `www.verit-solver.org`

CVC5

- Industrial strength
- Supports everything
- `cvc5.github.io`

Some Solvers You Can Try (a Biased List)

veriT

- Small solver
- Excellent proofs, good quantifier support
- `www.verit-solver.org`

CVC5

- Industrial strength
- Supports everything
- `cvc5.github.io`

Bitwuzla

- Specialized on bit-vectors, and floating-points
- Very fast
- `bitwuzla.github.io`

Some Solvers You Can Try (a Biased List)

veriT

- Small solver
- Excellent proofs, good quantifier support
- `www.verit-solver.org`

CVC5

- Industrial strength
- Supports everything
- `cvc5.github.io`

Bitwuzla

- Specialized on bit-vectors, and floating-points
- Very fast
- `bitwuzla.github.io`

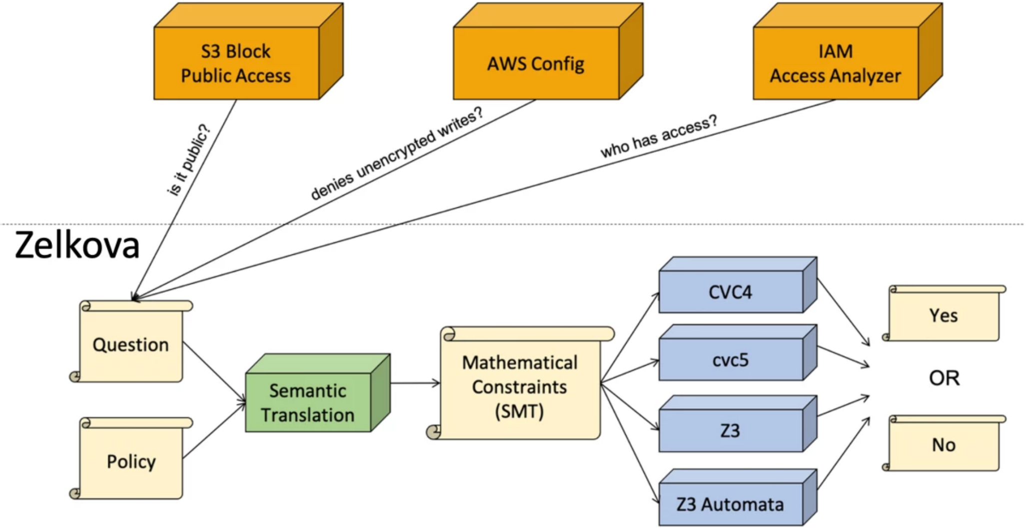
Z3

- Very established
- Also supports everything
- `https://github.com/Z3Prover/z3`

Part II

SMT Proofs

Example Application: aws Zelkova



Zelkova Style SMT Constraints

$$\begin{aligned} \mathit{Policy} \Rightarrow \mathit{Query} &\text{ is valid} \\ \neg(\mathit{Policy} \Rightarrow \mathit{Query}) &\text{ is unsatisfiable} \\ \mathit{Policy} \wedge \neg\mathit{Query} &\text{ is unsatisfiable} \end{aligned}$$

Zelkova Style SMT Constraints

$$Policy \Rightarrow Query \text{ is valid}$$
$$\neg(Policy \Rightarrow Query) \text{ is unsatisfiable}$$
$$Policy \wedge \neg Query \text{ is unsatisfiable}$$

- Query is against policy: satisfiable!
  - Evidence: counter**model**
  - Easy to check by evaluation.
- Query follows policy: unsatisfiable!
  - Evidence: refutation **proof**
  - Hard!

SMT Proofs: Basic Structure

$$\dfrac{\dfrac{\dfrac{t_2}{t_3}}{\vdots}\quad}{}$$

$$\dfrac{t_1 \qquad t_4}{t_1 \land t_4}\ \text{andI}$$

$$t_1, t_2 \vdash t_1 \land t_4$$

```
(assume a0 t1)
(assume a1 t2)
(step  s1 t3
        :premises (a1)      :rule rule1)
...
(step s20 t4
        :premises (s19)     :rule rule2)
(step s21 (and t1 t4)
        :premises (a0 s20) :rule andI)
```

SMT Proofs: Alternative View

Proofs as Terms

- Proofs are terms of a dedicated `Proof` type.
- The `Proof` type depends on the formula it proves.

Example

```
(andI
    ((assume t1)
        (rule2 (...(rule1 ((assume t2)))...))
    )
) : Proof (and t1 t4)
```

and introduction

```
(declare-rule andI ((F1 Bool) (F2 Bool))
    :premises (F1 F2)
    :conclusion (and F1 F2)
)
```

and introduction

```
(declare-rule andI ((F1 Bool) (F2 Bool))
    :premises (F1 F2)
    :conclusion (and F1 F2)
)
```

Resolution

```
(program $resolve ((C1 Bool) (C2 Bool) (pol Bool) (L Bool))
  (Bool Bool Bool Bool) Bool
  (($resolve C1 C2 pol L)
    (eo::list_concat or ($nary_remove or false (eo::ite pol L (not L)) C1)
                        ($nary_remove or false (eo::ite pol (not L) L) C2))
  )
)
(declare-rule resolution ((C1 Bool) (C2 Bool) (pol Bool) (L Bool))
    :premises (C1 C2)  :args (pol L)  :conclusion ($resolve C1 C2 pol L)
)
```

Eunoia: cvc5's DSL to Describe Proof Rules

This language is odd!

Eunoia: cvc5's DSL to Describe Proof Rules

This language is odd!

- No dedicated term datatype.

Eunoia: cvc5's DSL to Describe Proof Rules

This language is odd!

- No dedicated term datatype.
- Dependently typed, but there is no Π-binder.

This language is odd!

- No dedicated term datatype.
- Dependently typed, but there is no Π-binder.
- Variables are scoped over types and case branches.

Eunoia: cvc5's DSL to Describe Proof Rules

This language is odd!

- No dedicated term datatype.
- Dependently typed, but there is no Π-binder.
- Variables are scoped over types and case branches.
  - Branch variables have the same general type,

Eunoia: cvc5's DSL to Describe Proof Rules

This language is odd!

- No dedicated term datatype.
- Dependently typed, but there is no Π-binder.
- Variables are scoped over types and case branches.
  - Branch variables have the same general type,
  - but type variables instantiation is branch independent.

This language is odd!

- No dedicated term datatype.
- Dependently typed, but there is no Π-binder.
- Variables are scoped over types and case branches.
  - Branch variables have the same general type,
  - but type variables instantiation is branch independent.
- Programs can diverge.

Eunoia: cvc5's DSL to Describe Proof Rules

This language is odd!

- No dedicated term datatype.
- Dependently typed, but there is no Π-binder.
- Variables are scoped over types and case branches.
  - Branch variables have the same general type,
  - but type variables instantiation is branch independent.
- Programs can diverge.
- If there is no matching branch, they get stuck!

System Y: A Mechanized Formal Core Language For Eunoia

- Ongoing work!
- Mechanized in Agda, so we can trust the results.

- Ongoing work!
- Mechanized in Agda, so we can trust the results.
- Symbols are associated with parameter lists.

System Y: A Mechanized Formal Core Language For Eunoia

- Ongoing work!
- Mechanized in Agda, so we can trust the results.
- Symbols are associated with parameter lists.
- Binding is handled locally via Meta-vectors.

System Y: A Mechanized Formal Core Language For Eunoia

- Ongoing work!
- Mechanized in Agda, so we can trust the results.
- Symbols are associated with parameter lists.
- Binding is handled locally via Meta-vectors.
  - e.g., bit-vectors that track bound variables.

System Y: A Mechanized Formal Core Language For Eunoia

- Ongoing work!
- Mechanized in Agda, so we can trust the results.
- Symbols are associated with parameter lists.
- Binding is handled locally via Meta-vectors.
    - e.g., bit-vectors that track bound variables.
- Divergence is handled via guards

System Y: A Mechanized Formal Core Language For Eunoia

- Ongoing work!
- Mechanized in Agda, so we can trust the results.
- Symbols are associated with parameter lists.
- Binding is handled locally via Meta-vectors.
  - e.g., bit-vectors that track bound variables.
- Divergence is handled via guards
  - you must provide evidence a program evaluates in finitely many steps.

Thank You!