Satisfiability Modulo Theories (SMT) solving

Amar Shah

Carnegie Mellon University
Computer Science Department

Advanced Topics in Logic: Automated Reasoning and Satisfiability

Come to my Office Hours!!

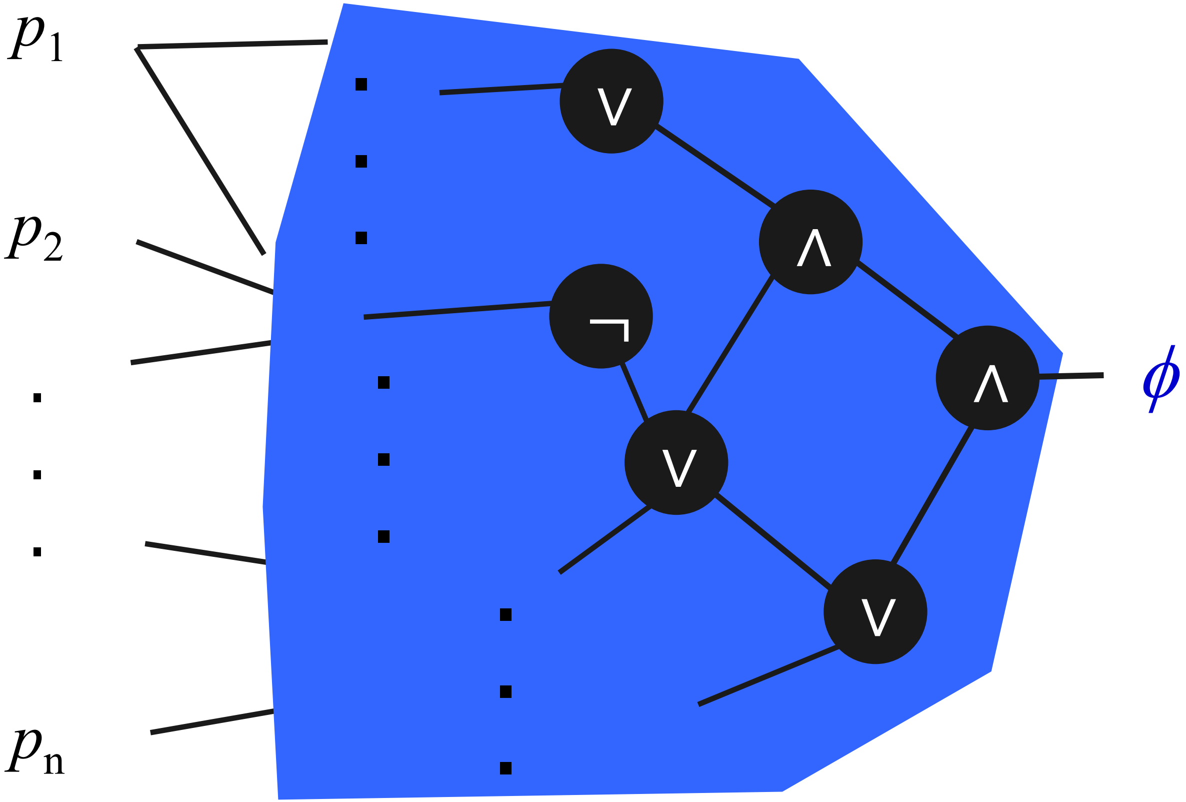
(CIC 2206, Wednesdays 11am – noon)

- Slides inspire by previous presentations by Sanjit Seshia, Clark Barrett, Andrew Reynolds, Leonardo de Moura, and Alberto Oliveras
- SMT solving is a pretty broad field, which is continuing to evolve. This presentation is only a taste

1. **Introduction**
2. Defining SMT
3. Eager Approach
4. Lazy Approach
5. Universal Quantifiers
6. Future of SMT solving

This class so far: Boolean SAT solving

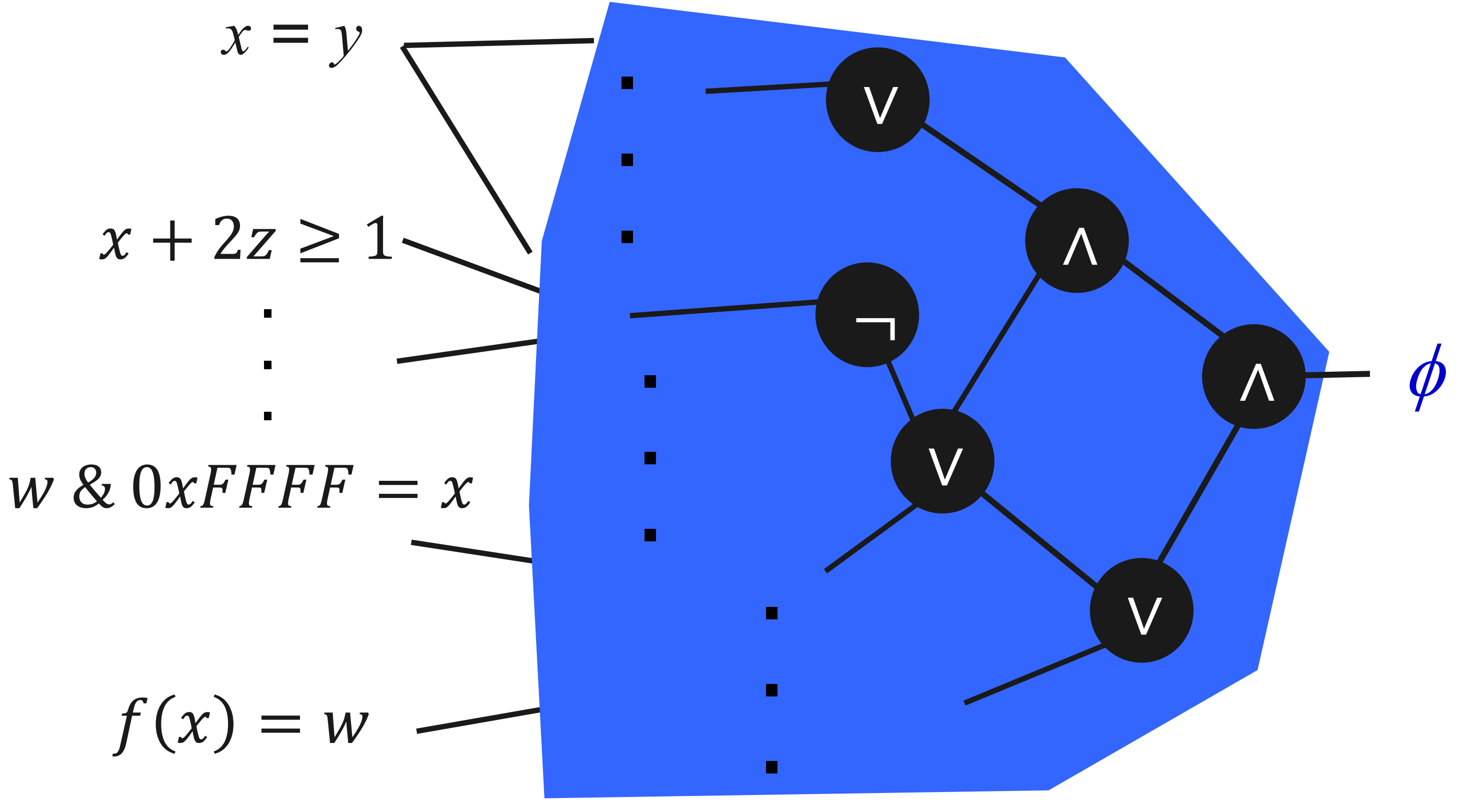
Is there an assignment to the $p_1, p_2, \ldots, p_n$ variables such that ϕ evaluates to true?



Satisfiability Modulo Theories

Is there an assignment to f, w, x, y, z such that ϕ evaluates to true?

i.e. is the formula satisfiable?



What is SMT?

$$p_1 \vee (p_2 \wedge p_3)$$

- SAT: use **propositional logic** as the formalization language

  \+ high degree of efficiency

  \- expressive (all NP-complete) but involved encodings

$$x = y \vee (x + 2z \geq 1 \wedge f(x) = w)$$

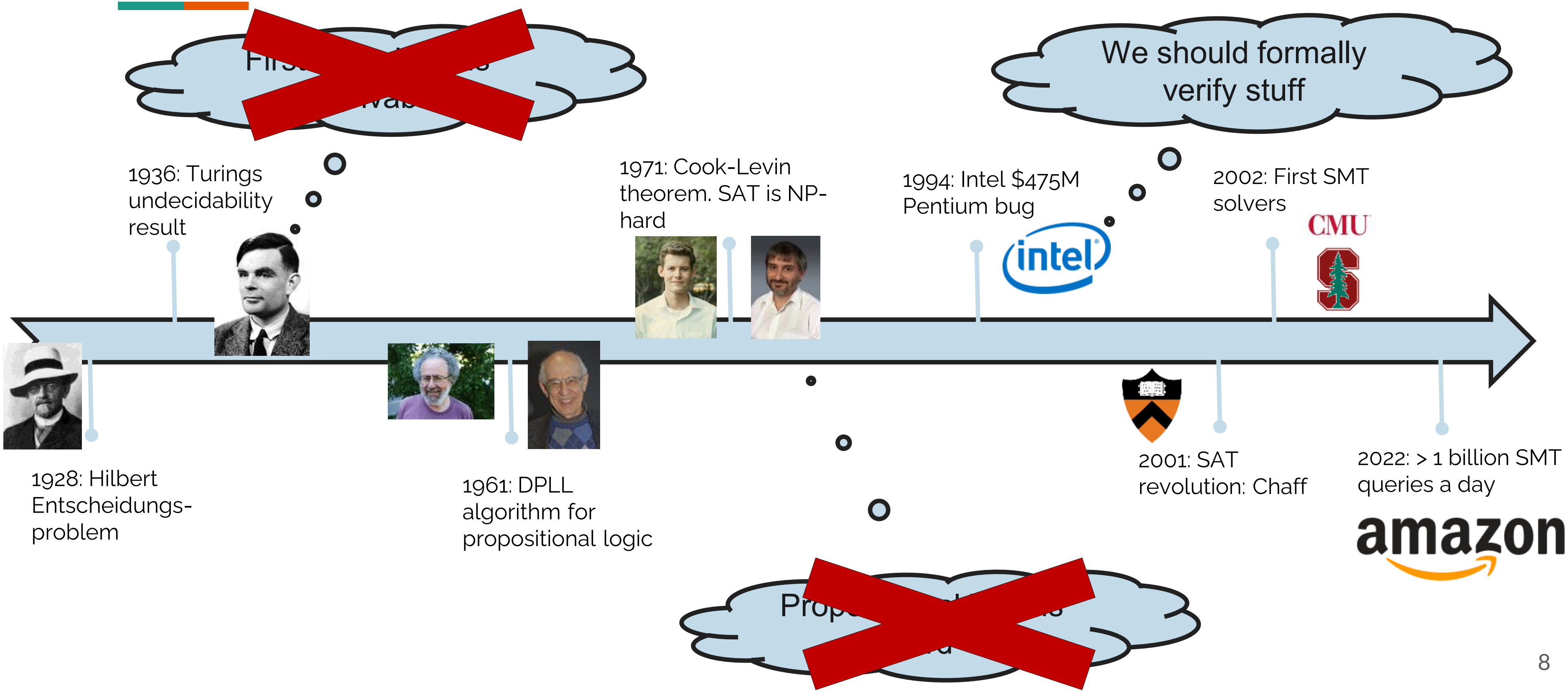
- SMT: propositional logic + **domain-specific reasoning (multi-sorted first-order logic)**

  \+ improves the expressivity

  \- certain (but acceptable) loss of efficiency

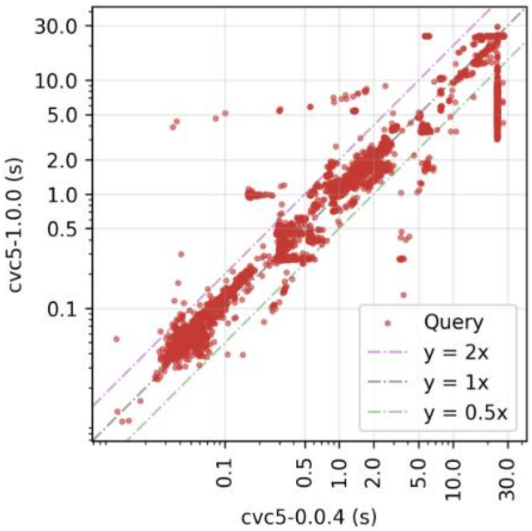
<u>Goal:</u> Introduce SMT solving and its main techniques

History Lesson: trying to solve first-order logic



A Billion SMT Queries a Day (Invited Paper)

(roughly the same number as ChatGPT)



- How did we get here?
  - “Just because a problem is undecidable, it doesn’t stop being important”
  - Modelling solvers after the problems we want to solve
  - Combination of multiple approaches: eager and lazy

- Where do we go next?
  - Find the right problems
  - A trillion SMT queries a day?

Many Applications

Policy Access Analyzer — aws Zelkova

Hardware Verification  

Model checking — CBMC, kani

Program Verification — Dafny

...

SMT solvers — Z3, CVC5

1. Introduction
2. **Defining SMT**
3. Eager Approach
4. Lazy Approach
5. Universal Quantifiers
6. Future of SMT solving

Formally: multi-sorted first-order logic

- Can define a set of sorts S for example $S = \{Bool, Int, Real, String\}$
  - Can have user-defined (uninterpreted) sorts
- A signature Σ with:
  - Function symbols $\Sigma_F = \{f : \sigma_1 \times \cdots \times \sigma_l \rightarrow \sigma, \dots\}$
  - Predicate Symbols $\Sigma_R = \{P : \sigma_1 \times \cdots \times \sigma_k\}$
  - $\sigma_i \in S$
- Function symbols with arity 0 are called constants
- A set of variables V
- From these build up a set of terms, atomic formulas, and formulas
- A theory is a set of formulas closed under logical deduction
- $Bool$ and $=$ are part of all theories

Example: Uninterpreted Functions (UF)

```
(declare-sort T 0)
(declare-fun x () T)
(declare-fun f (T) T)
(assert (= (f (f (f x))) x))
(assert (= (f (f (f (f (f x))))) x))
(assert (not (= (f x) x)))
```

This is SMT syntax. Sorry!

- Also called the “free theory”
- Because function symbols can take any meaning
- Only property required is *congruence*: that functions map identical arguments to identical values i.e., x = y => f(x) = f(y)
- $S = \{Bool,\ T, \ldots\}$ where T is a user defined sorts
- $\Sigma_F = \{f, g, \ldots\}$ where f, g are user defined functions

Example: Linear Integer Arithmetic

```
(declare-fun x () Int)

(declare-fun y () Int)

(declare-fun z () Int)

(assert (or (<= (+ x y) 10)

        (= x z))

(assert (<= (- (+ x x) y) 5))
```

- $S = \{Bool,\ Int\}$
- $\Sigma_F = \{0, 1, +,\ -\}$
- $\Sigma_P = \{\leq, =\}$
- Can express larger integers and $\geq, <, >$
- Boolean combination of linear constraints of the form
  $(a_1 x_1 + a_2 x_2 + \ldots + a_n x_n \sim b)$
- There are also nonlinear and real arithmetic theories

Example: (Real) Integer Difference Logic

```
(declare-fun x () Int)

(declare-fun y () Int)

(declare-fun z () Int)

(assert (or (>= (- x y) 10)

        (= x 2))

(assert (<= (- x z) 5))
```

- $S = \{Bool,\ Int\}$
- $\Sigma_F = \{0, 1, -\}$
- $\Sigma_P = \{\leq, =\}$
- Boolean combination of linear constraints of the form
  $x_i - x_j \sim c_{ij}$ or $x_i \sim c_i$
- Applications
  - Processor datapath verification
  - Job shop scheduling / real-time systems
  - Timing verification for circuits

Example: Bitvectors

```
(declare-fun x () (_ BitVec 8))
(declare-fun y () (_ BitVec 8))
(declare-fun z () (_ BitVec 8))
(assert (= x #b10101010))
(assert (= z (bvand x y)))
(assert (= (bvadd x y) #xff))
```

- $S = \{Bool,\ BV1, BV2,\ BV3 \ldots\}$
- Fixed width data words
  - Can model int, short, long, .
- Arithmetic operations
  - add/subtract/multiply/divide & comparisons
  - Two's complement and unsigned operations
- Bit-wise logical operations
  - E.g., and/or/xor, shift/extract and equality
- Boolean connectives

Example: Arrays

```
(declare-fun arr () (Array Int Int))

(declare-fun i () Int

(declare-fun j () Int)

(assert (= (select arr i) 42))

(assert (= (select (store arr j 10) i) 10))

(assert (not (= i j)))
```

- Two interpreted functions: select and store
  - select(A,i) ->Read from A at index i
  - store(A,i,d)-> Write d to A at index i
- Two main axioms:
  - for i ≠ j
- One other axiom:
  - $(\forall i. select(A, i) = select(B, i)) \rightarrow A = B$

Others

- Strings
- Floating Points
- Algebraic Datatypes
- Higher Order Functions
- Finite Fields

1. Introduction
2. Defining SMT
3. **Eager Approach**
4. Lazy Approach
5. Universal Quantifiers
6. Future of SMT solving

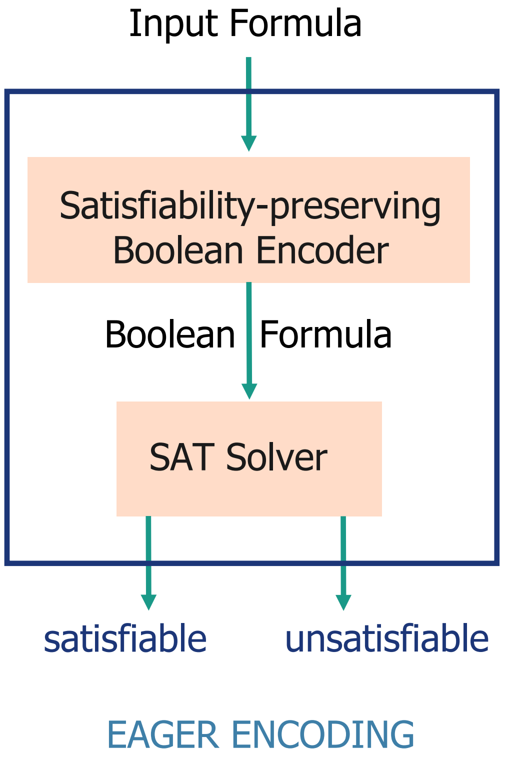
Have to combine theory reasoning with SAT solving

Two main approaches:

1. “Eager”
   - translate into an equisatisfiable propositional formula
   - feed it to any SAT solver

2. “Lazy”
   - abstract the input formula to a propositional one
   - feed it to a (DPLL-based) SAT solver
   - use a theory decision procedure to refine the formula and guide the SAT solver

Eager Approach to SMT



SAT Solver involved in Theory Reasoning

<u>Key Ideas:</u>
Small-domain encoding
Constrain model search
Rewrite rules
Abstraction-based methods (eager + lazy)

<u>Example Solvers:</u>
NFA2SAT, Bitwuzla, Algaroba, UCLID, STP
...

Ackerman's Encoding: UF to SAT

$f(x) \longrightarrow fx$

$f(y) \longrightarrow fy$

For each pair of function applications add:

$$x = y \Rightarrow fx = fy$$

```
(declare-sort T 0)

(declare-fun x () T)

(declare-fun f (T) T)

(assert (= (f (f (f x))) x))

(assert (= (f (f (f (f (f x))))) x))

(assert (not (= (f x) x))
```

What is the Ackerman encoding of this?

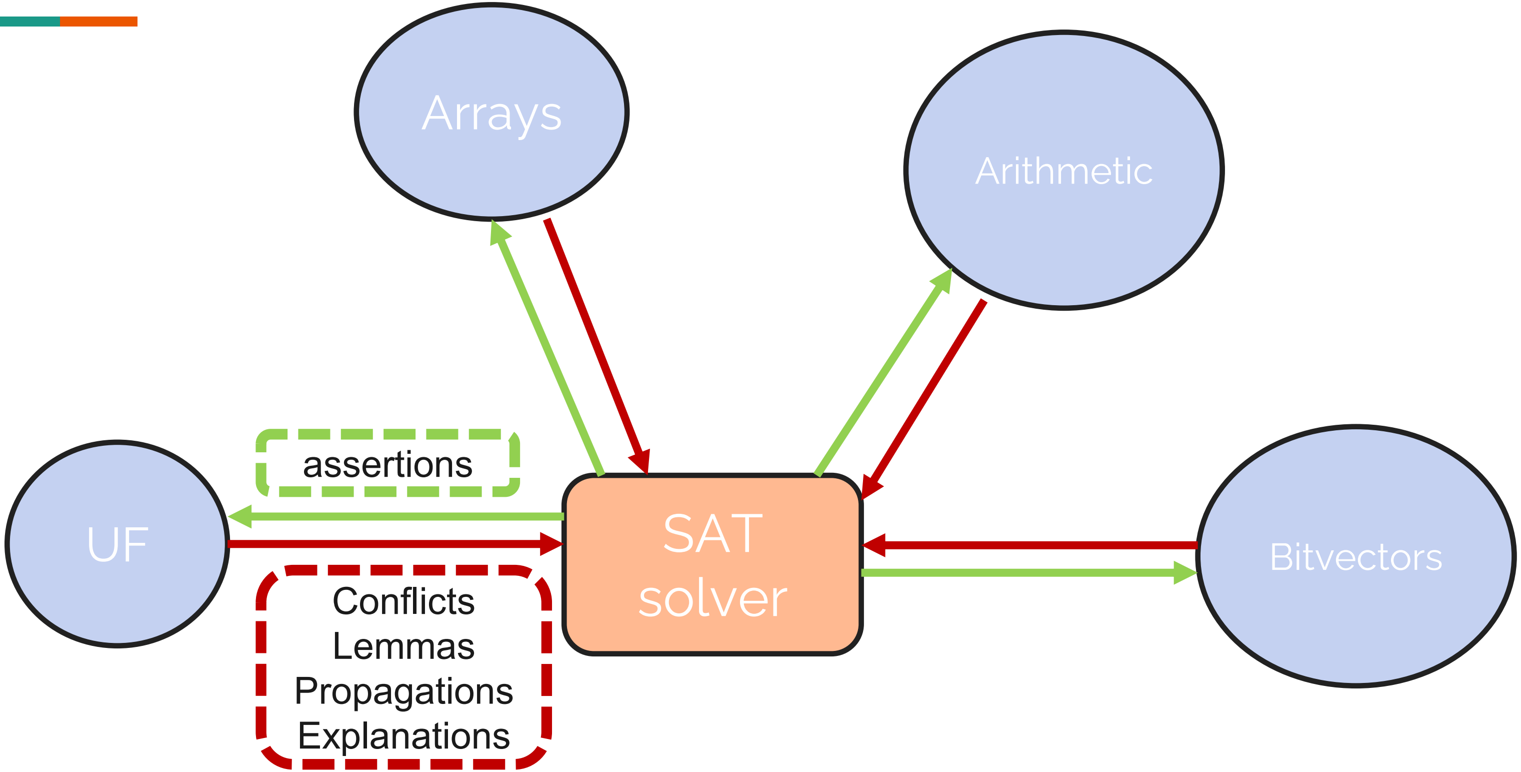
Another example: bitvectors

```
(declare-fun x () (_ BitVec 8))

(declare-fun y () (_ BitVec 8))

(declare-fun z () (_ BitVec 8))

(assert (= x #b10101010))

(assert (= z (bvand x y))
```

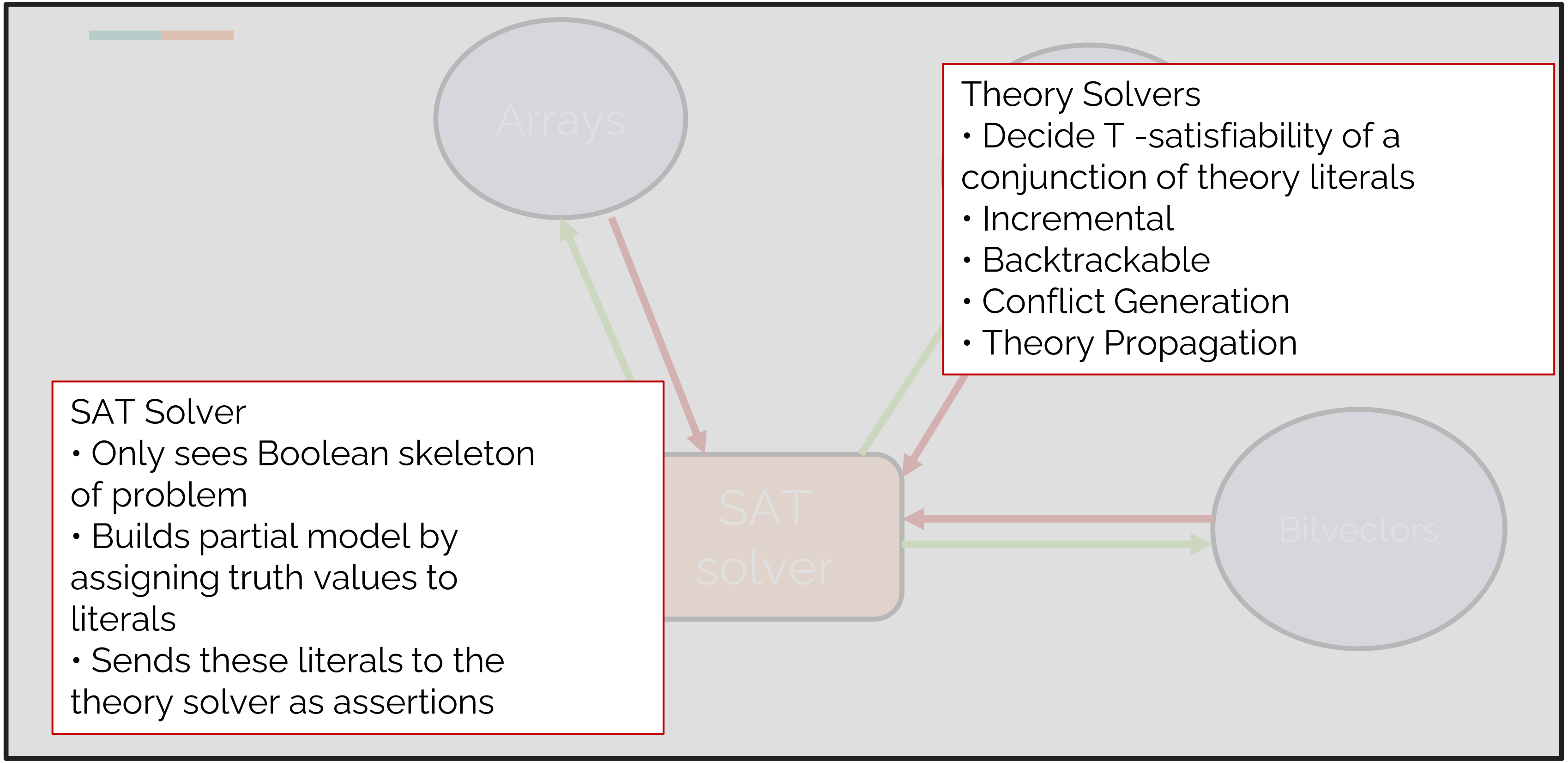
Can we bit-blast this to SAT?

1. Introduction
2. Defining SMT
3. Eager Approach
4. **Lazy Approach**
5. Universal Quantifiers
6. Future of SMT solving

Lazy SMT solving: often called CDCL(T)



Lazy SMT solving: often called CDCL(T)



Theory Solvers

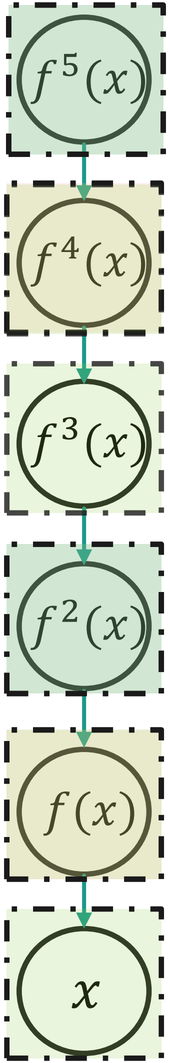
- Decide T -satisfiability of a conjunction of theory literals
- Incremental
- Backtrackable
- Conflict Generation
- Theory Propagation

SAT Solver

- Only sees Boolean skeleton of problem
- Builds partial model by assigning truth values to literals
- Sends these literals to the theory solver as assertions

Congruence Closure Algorithm (for UF)

$$x = f(f(f(x))) \wedge$$
$$x = f(f(f(f(f(x))))) \wedge$$
$$x \neq f(x)$$



Notice $x = f^2(x)$

Hence $f(x) = f^3(x)$

Conflict

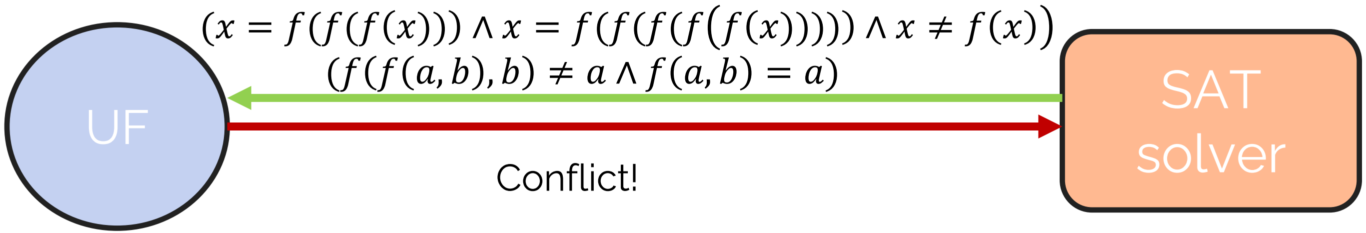
We learn a conflict clause and backtrack!

More examples

1. $f(f(a,b),b) \neq a \wedge f(a,b) = a$
2. $f(a,a) = b \wedge g(c,a) = c \wedge g\big(c, f(a,a)\big) = f(g(c,a), g(c,a)) \wedge f(c,c) \neq g(c,b)$
3. $f(a,a) = b \wedge g(c,a) = c \wedge g\big(c, f(a,a)\big) = f\big(g(c,a), g(c,a)\big) \wedge f(h(d),d) = f\big(g(a,c), g(c,a)\big) \wedge h(a) = h(c)$

Putting it all together: lazy SMT solving

$$[(x = f(f(f(x))) \land x = f(f(f(f(f(x))))) \land x \neq f(x))$$
$$\lor (f(f(a,b),b) \neq a \land f(a,b) = a)]$$
$$\land [(x \neq f(f(f(x))) \lor x \neq f(f(f(f(f(x))))) \lor x = f(x))]$$
$$\land [(f(f(a,b),b) = a \lor f(a,b) \neq a]$$

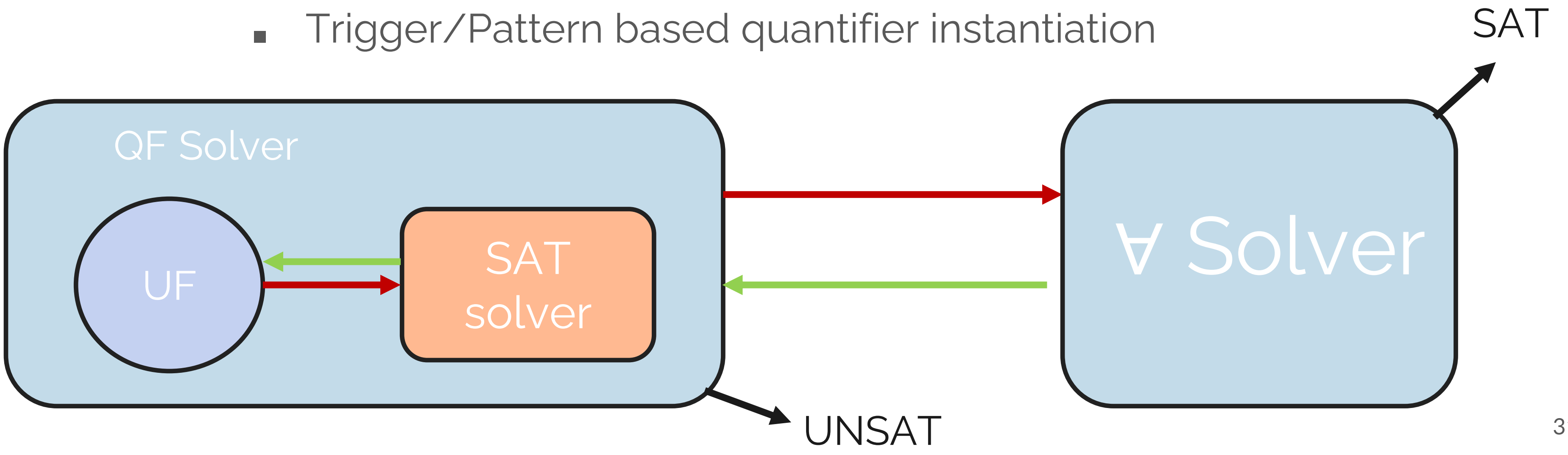


UNSAT!

1. Introduction
2. Defining SMT
3. Eager Approach
4. Lazy Approach
5. **Universal Quantifiers**
6. Future of SMT solving

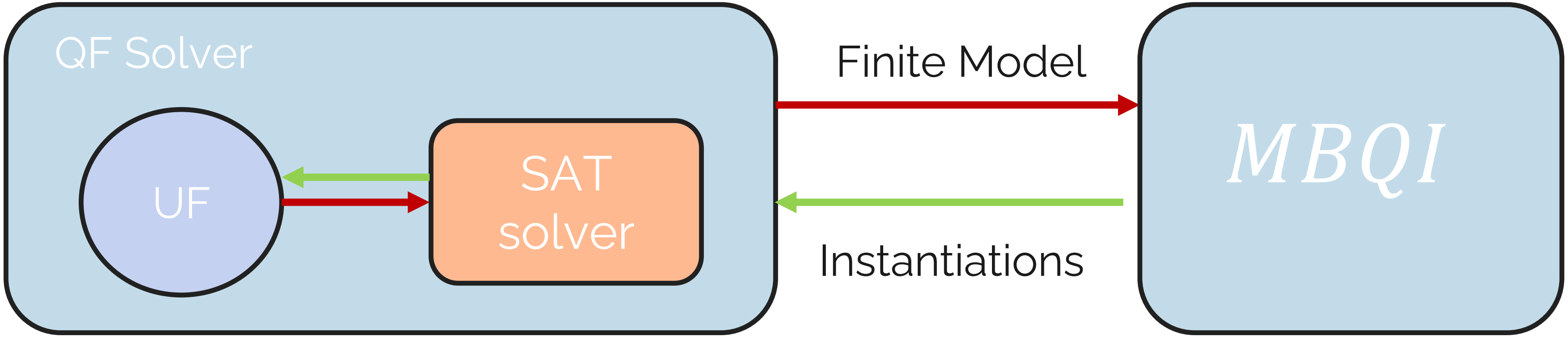
FOL is undecidable: because of quantifiers

- How do we reason about"forall" statements?
- Could enumerate all cases?
- Two smarter approaches?
  - Model-based quantifier instantiation
  - Trigger/Pattern based quantifier instantiation



Model-based quantifier instantiation

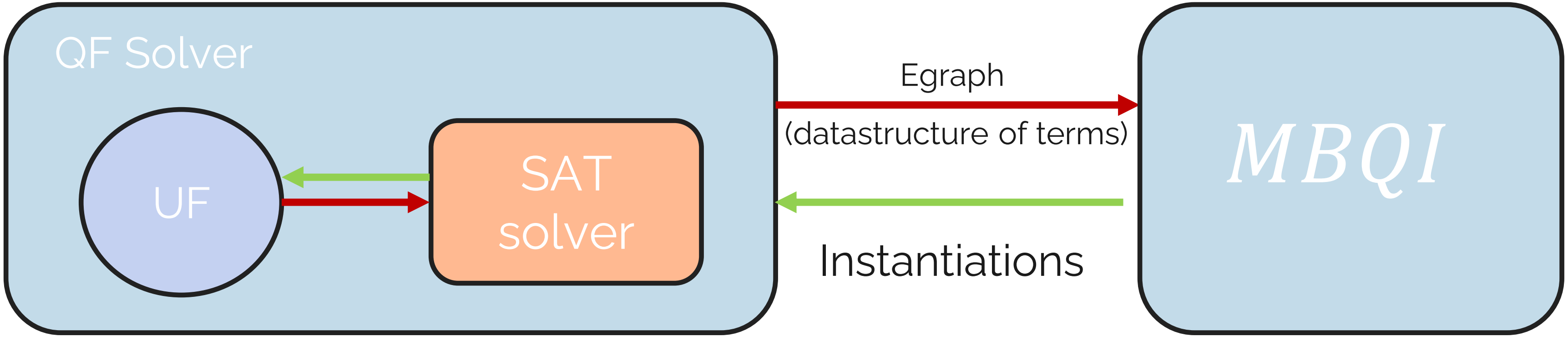
- Idea: Instantiate quantifiers based on (complete) QF models
- Complete for certain fragments, e.g. EPR, essentially uninterpreted
- Can be useful for answering “sat”



Example: $f(t) \land g(s) \land [\forall x. \neg f(x) \lor \neg g(x)]$

Pattern/Trigger based quantifier instantiation

- Idea: Instantiations found by pattern matching
- Implemented in early SMT solvers (e.g. simplify) as well as z3, cvc5
- Key applications: Software verification (dafny, Verus)



Example: $f(t) \wedge g(s) \wedge [\forall x. f\big(g(x)\big) : pattern\ g(x)]$ $\longrightarrow$ Instantiation: $f(g(s))$

Pattern-based Instantiations Example (from Verus)

We have the assertion

```
(assert
 (not (=> (and (is_member x (set_union s1 s2))
                  (not (is_member x s1)))
     (is_member x s2))))
```

Trying to instantiate quantifier

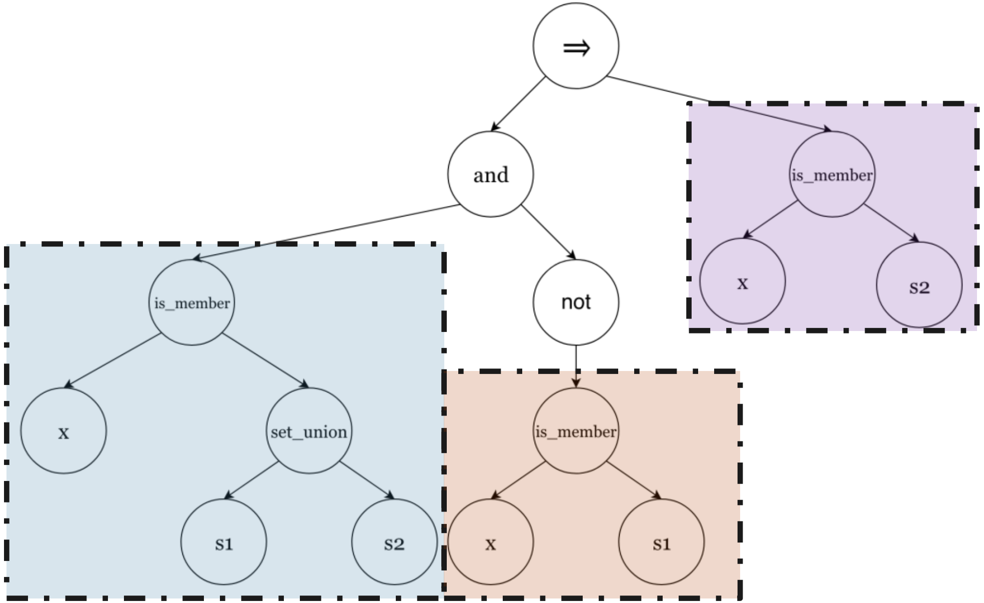
```
(forall ((y Poly) (set Poly)) (!
   (= (is_member y set) (Set.contains set y))
   :pattern ((is_member y set)))))
```

Adding the instantiations:

```
(= (is_member x (set_union s1 s2))
   (Set.contains (set_union s1 s2) x))
```

```
(= (is_member x s1) (Set.contains s1 x))
```

```
(= (is_member x s2) (Set.contains s2 x))
```



Other quantifier instantiation techniques

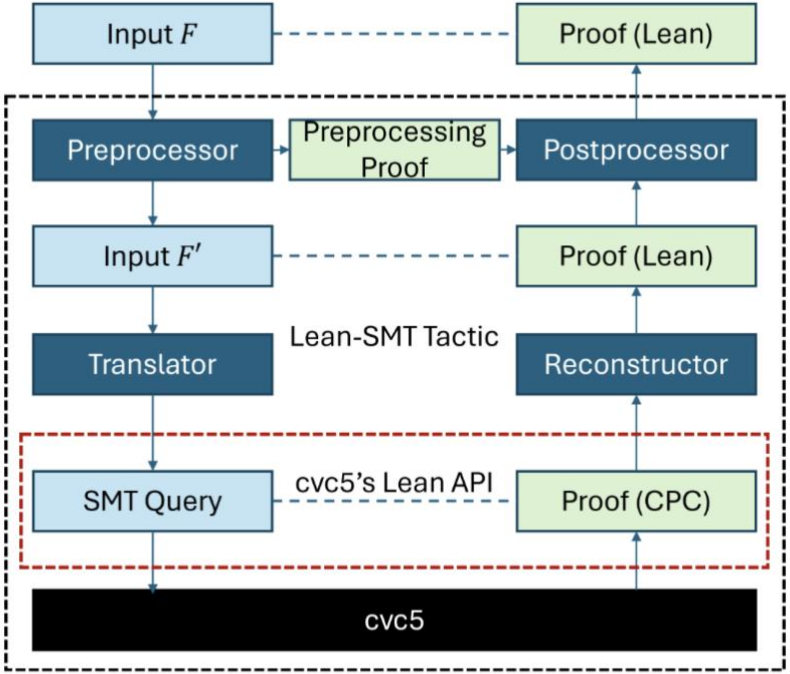
- Enumerative instantiation
- Counter-example guided instantiation
- Syntax Guided instantiation

1. Introduction
2. Defining SMT
3. Eager Approach
4. Lazy Approach
5. Universal Quantifiers
6. **Future of SMT solving**

Where else are SMT solvers useful?

- Disclaimer: SMT landscape is vast and I only gave you a brief taste of it
- Recent Applications in cryptography, programming languages, systems, and machine learning

Programming Languages: A Hammer for Lean

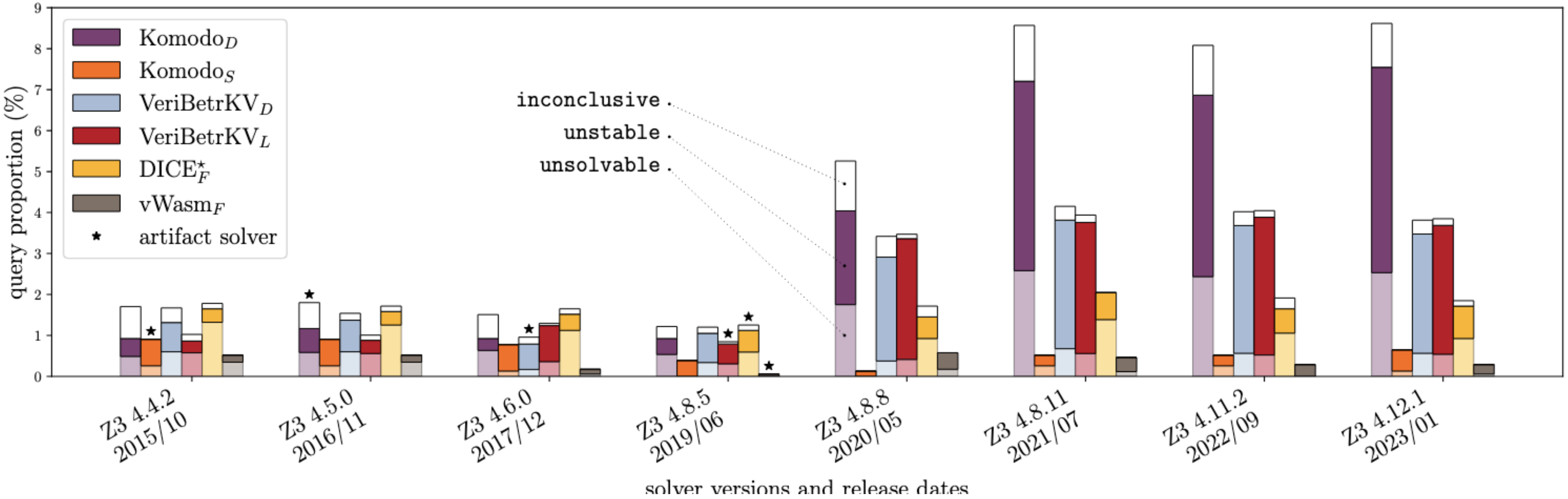


- Lean is a programming language/interactive theorem prover
- Calls tactics
- Two new tactics, lean-auto and lean-smt call SMT solvers to prove Lean goals

lean-smt: An SMT tactic for discharging proof goals in Lean (2025)

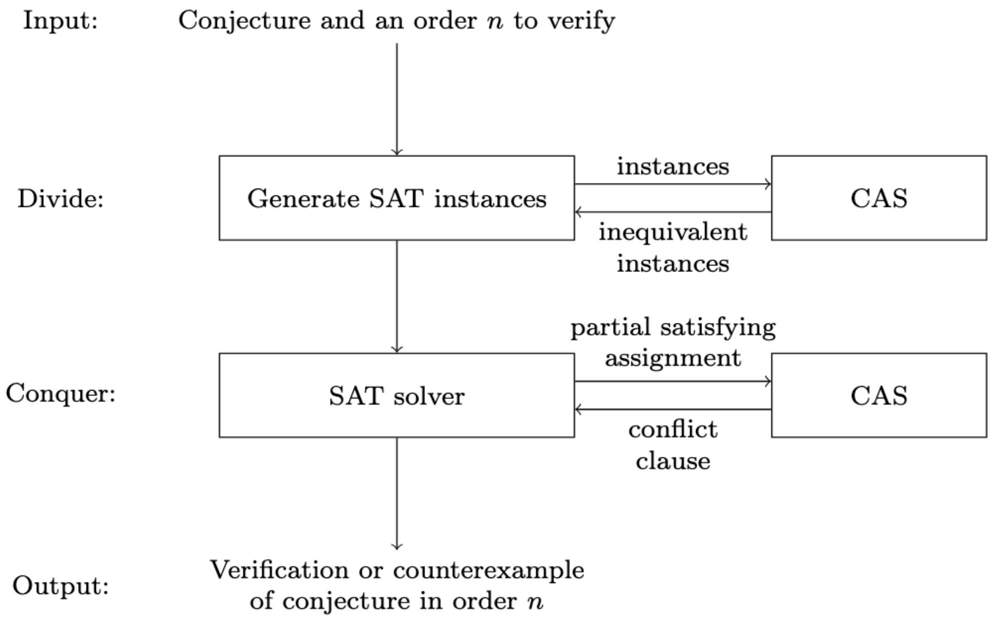
Systems: Reliability/Instability of SMT solvers

- SMT solvers used by systems with O(1 billion) queries
- They need to be robust/reliable



Mariposa: Measuring SMT Instability in Automated Program Verification (2023)

Mathematics: Enumerating matrices

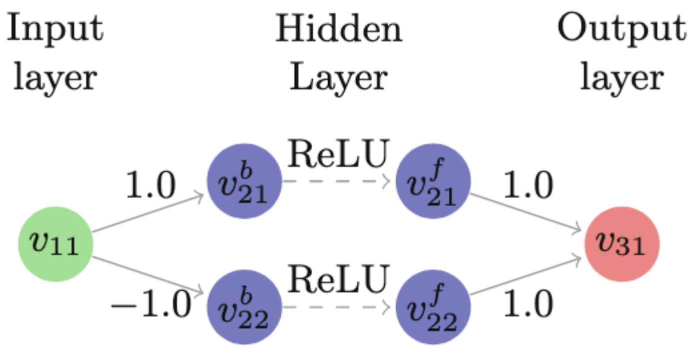


- Combining a SAT solver with a computer algebra system
- Very good with enumerating matrices of certain properties

A SAT+CAS Method for Enumerating Williamson Matrices of Even Order

Machine Learning: Verifying Neural Networks

- Verifying Neural Networks—can mostly be modeled as a linear function
- Special decision procedure to deal with ReLU



Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks (2017)

Cryptography: Satisfiability modulo Finite Fields

$$x' + y' = \sum_{i=1}^{b+1} 2^{i-1} z_i' \quad \wedge \quad z' = \sum_{i=1}^{b} 2^{i-1} z_i' \quad \wedge \quad \bigwedge_{i=1}^{b+1} z_b'(z_b' - 1) = 0$$

Used to prove the correctness of a Zero Knowledge Proof compiler!

Definition 1 (Correctness). *A ZKP compiler* $\mathsf{Compile}(\phi) \to (\phi', \mathsf{Ext}_x, \mathsf{Ext}_w)$ *is* ***correct*** *if it is demonstrably complete and demonstrably sound.*

- *demonstrable completeness*: For all $\mathsf{x} \in \mathsf{dom}(x), \mathsf{w} \in \mathsf{dom}(w)$ such that $\hat{\phi}(\mathsf{x}, \mathsf{w}) = \top$,
$$\hat{\phi}'(\mathsf{Ext}_x(\mathsf{x}), \mathsf{Ext}_w(\mathsf{x}, \mathsf{w})) = \top$$
- *demonstrable soundness*: There exists an efficient algorithm $\mathsf{Inv}(\mathsf{x}', \mathsf{w}') \to \mathsf{w}$ such that for all $\mathsf{x} \in \mathsf{dom}(x), \mathsf{w}' \in \mathsf{dom}(w')$ such that $\hat{\phi}'(\mathsf{Ext}_x(\mathsf{x}), \mathsf{w}') = \top$,
$$\hat{\phi}(\mathsf{x}, \mathsf{Inv}(\mathsf{Ext}_x(\mathsf{x}), \mathsf{w}')) = \top$$

Satisfiability modulo Finite Fields (2023)

How can SMT solvers be applied to work that you do?

Come to my Office Hours!!

(CIC 2206, Wednesdays 11am – noon)