SMT Solving for Verification

(or rather an overview of SMT and its applications)

Haniel Barbosa

UFMG

CVC5

ATVA 2024

Oct 21, 2024, Kyoto

Acknowledgments

Many thanks to the ATVA Program Committee for the invitation

Many thanks to local organizers for the wonderful hospitality and support

The work on proofs mentioned in the last part is funded in (large) part by AWS, DARPA, NSF, and Stanford Center for AR. For more general information on cvc5's funding as a whole see https://cvc5.github.io/acknowledgements.html.

Acknowledgments

Many thanks to Cesare Tinelli, Yoni Zohar, Andres Nötzli, Aina Niemetz, Mathias Preiner, Andrew Reynolds, Clark Barrett, Alberto Griggio, Liana Hadarean, Dejan Jovanovic, and Albert Oliveras for contributing some of the material used in these slides.

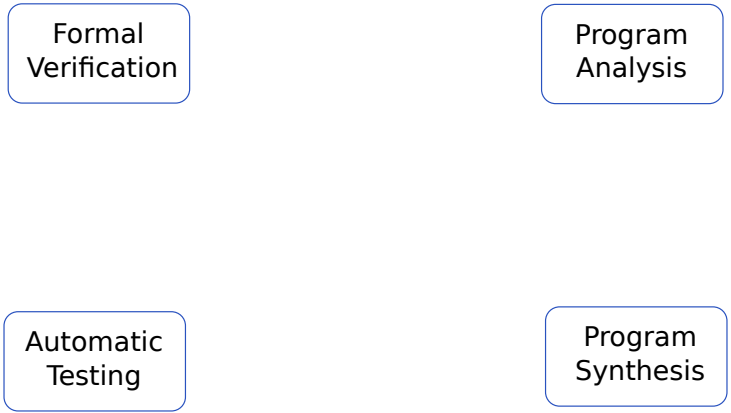
Disclaimer: The literature on SMT and its applications is vast. The bibliographic references provided here are just a small and highly incomplete sample. Apologies to all authors whose work is not cited.

Agenda

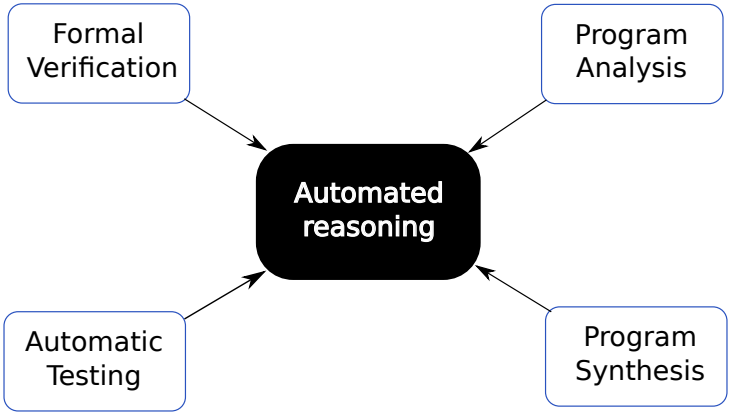
1. Introduction
2. SMT solver functionality
3. Background theories
4. Applications
   - Model Checking
   - Synthesis
   - Software Verification
   - Misc
5. Producing and checking proof certificates

Introduction

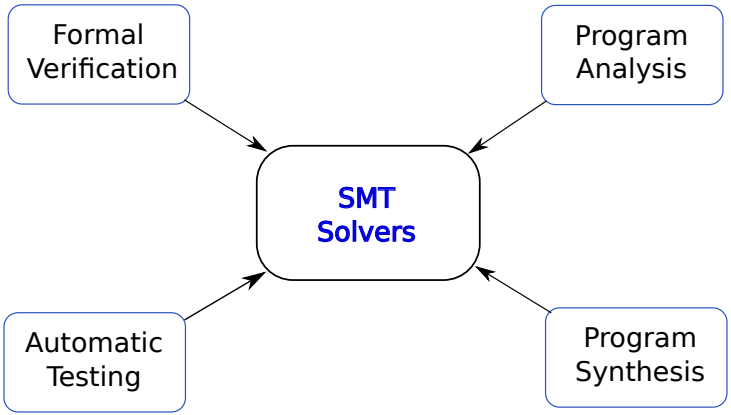
Automated Reasoning for Formal Methods



Automated Reasoning for Formal Methods



Automated Reasoning for Formal Methods



Introduction

Historically:
Automated logical reasoning achieved through uniform theorem-proving procedures for First Order Logic (e.g., resolution, superposition, and tableaux calculi)

Limited success:
Uniform proof producedure for FOL are not always the best compromise between expressiveness and efficiency

Introduction

Last 20+ years: R&D has focused on

- expressive enough decidable fragments of various logics
- incorporating domain-specific reasoning, e.g., on:
  - temporal reasoning
  - arithmetic reasoning
  - equality reasoning
  - reasoning about certain data structures (arrays, lists, finite sets, ...)
- combining specialized reasoners modularly

Introduction

Two successful examples of this trend:

SAT: propositional formalization, Boolean reasoning

- \+ high degree of efficiency
- − expressive (all NP-complete problems) but involved encodings

Introduction

Two successful examples of this trend:

SAT: propositional formalization, Boolean reasoning

- \+ high degree of efficiency
- − expressive (all NP-complete problems) but involved encodings

SMT: first-order formalization, Boolean + domain-specific reasoning

- \+ improves expressivity and scalability
- − some (but acceptable) loss of efficiency

Introduction

Two successful examples of this trend:

SAT: propositional formalization, Boolean reasoning
- \+ high degree of efficiency
- − expressive (all NP-complete problems) but involved encodings

SMT: first-order formalization, Boolean + domain-specific reasoning
- \+ improves expressivity and scalability
- − some (but acceptable) loss of efficiency

This tutorial: an overview of SMT and its applications

The Basic SMT Problem

Determining the satisfiability of a logical formula wrt some combination T of background theories

Example

$$n > 3 * m + 1 \ \wedge \ (\ f \ (n) \leq \ head \ (l_1) \ \vee \ l_2 = \ f \ (n) :: l_1 \ )$$

The Basic SMT Problem

Determining the satisfiability of a logical formula wrt some combination T of background theories

Example

$$n > 3 * m + 1 \ \wedge \ (f(n) \leq head(l_1) \ \vee \ l_2 = f(n) :: l_1 \ )$$

The Basic SMT Problem

Determining the satisfiability of a logical formula wrt some combination T of background theories

Example

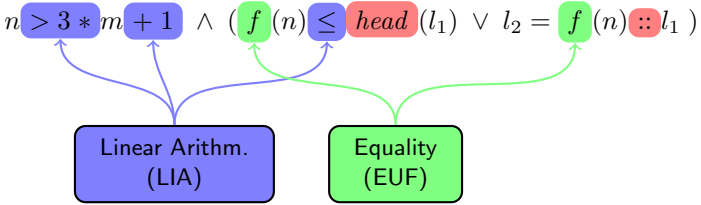
$$n > 3 * m + 1 \ \wedge \ (f(n) \leq head(l_1) \ \vee \ l_2 = f(n) :: l_1 \ )$$

Linear Arithm. (LIA)

The Basic SMT Problem

Determining the satisfiability of a logical formula wrt some combination T of background theories

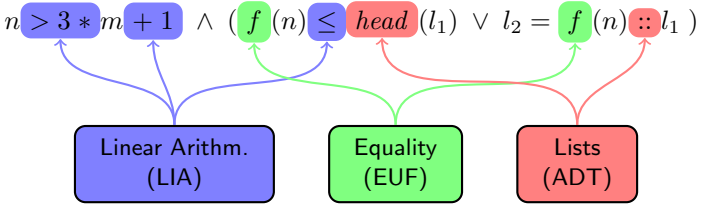
Example



The Basic SMT Problem

Determining the satisfiability of a logical formula wrt some combination T of background theories

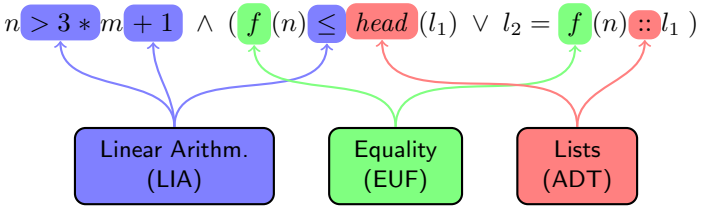
Example



The Basic SMT Problem

Determining the satisfiability of a logical formula wrt some combination T of background theories

Example



SMT formulas are formulas in
many-sorted FOL with built-in symbols

SMT solvers

Are highly efficient tools for the SMT problem based on specialized logic engines

SMT solvers

Are highly efficient tools for the SMT problem based on specialized logic engines

Are changing the way people solve problems in Computer Science and beyond:

- ▷ instead of building a special-purpose tool
- ▷ translate problem into a logical formula
- ▷ use an SMT solver as backend reasoner

SMT solvers

Are highly efficient tools for the SMT problem based on specialized logic engines

Are changing the way people solve problems in Computer Science and beyond:

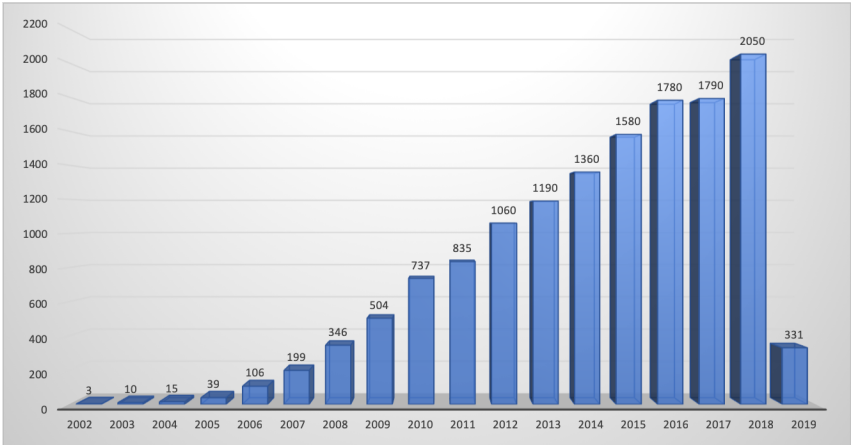
- ▷ instead of building a special-purpose tool
- ▷ translate problem into a logical formula
- ▷ use an SMT solver as backend reasoner

Not only easier, often better

The Explosion of SMT

Google Scholar

"Satisfiability Modulo Theories" OR "SMT Solver"



Popular SMT Solvers

	Citations	Google Scholar Hits
Z3	11,116[1]	≈17k
CVC, **CVC Lite**, **CVC 3**, **4**, **cvc5**	3,236[2]	≈4k
Yices 1, **2**	1,618[3]	≈3k
MathSat 3, **4**, **5**	1,153[4]	≈1.5k

[1] Moura and Bjørner 2008b, 2018 ETAPS Test of Time Award to Z3 developers

[2] Barbosa et al. 2022a; Barrett et al. 2011; Barrett and Berezin 2004; Barrett and Tinelli 2007; Stump et al. 2002

[3] Dutertre 2014; Dutertre and Moura 2006

[4] Bozzano et al. 2005b; Bruttomesso et al. 2008; Cimatti et al. 2013

Some Applications of SMT

Model Checking
- (in)finite-state systems
- hybrid systems
- abstraction refinement
- state invariant generation
- interpolation

Type Checking
- dependent types
- semantic subtyping
- type error localization

Program Analysis
- symbolic execution
- program verification
- verification in separation logic
- (non-)termination
- loop invariant generation
- procedure summaries
- race analysis
- concurrency errors detection

Software Synthesis
- syntax-guided function synthesis
- automated program repair
- synthesis of reactive systems
- synthesis of self-stabilizing systems
- network schedule synthesis

More Applications of SMT

Security
- automated exploit generation
- protocol debugging
- protocol verification
- analysis of access control policies
- run-time monitoring

Compilers
- compilation validation
- optimization of arithmetic computations

Planning
- motion planning
- nonlinear PDDL planning

Software Engineering
- system model consistency
- design analysis
- test case generation
- verification of ATL transformations
- semantic search for code reuse
- interactive (software) requirements prioritization
- generating instances of meta-models
- behavioral conformance of web services

Machine Learning
- verification of deep NNs

Business
- verification of business rules
- spreadsheet debugging

More Applications of SMT

Security

- automated exploit generation
- protocol debugging
- protocol verification
- **analysis of access control policies**
- run-time monitoring

Compilers

- compilation validation
- optimization of arithmetic computations

Planning

- motion planning
- nonlinear PDDL planning

Software Engineering

- system model consistency
- design analysis
- test case generation
- verification of ATL
- reuse
- tion
- meta-models
- behavioral conformance of web services

Machine Learning

- verification of deep NNs

Business

- verification of business rules
- spreadsheet debugging

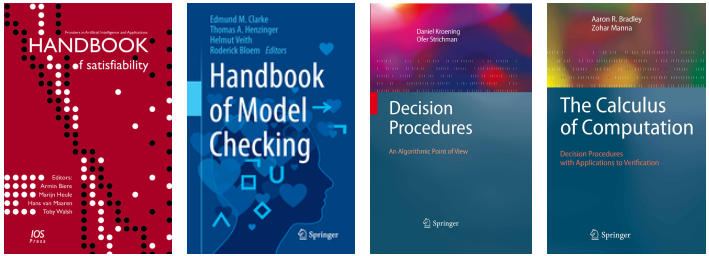
Heavily used at AWS

A billions SMT queries a day via Zelkova[a]

[a] Backes et al. 2018; Rungta 2022

More Information on SMT

Handbook chapters and books Barrett et al. 2009; Barrett and Tinelli 2018; Bradley and Manna 2007; Kroening and Strichman 2008



Online

- SMT-LIB at http://smt-lib.org
- SMT-COMP at http://smt-comp.org
- Satisfiability Modulo Theories: A Beginner's Tutorial at https://cvc5.github.io/tutorials/beginners

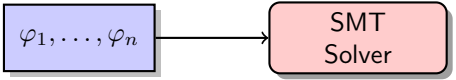
SMT solver functionality

Legend

v	value — i.e., distinguished variable-free term
$\varphi[\vec{x}]$	formula with free vars from $\vec{x} = (x_1, \ldots, x_n)$
$\varphi[\vec{x} \mapsto \vec{v}]$	formula obtained by replacing free occurrences of variables from $\vec{x}$ in φ with corresponding values from $\vec{v} = (v_1, \ldots, v_n)$
$\vec{x} = \vec{v}$	$x_1 = v_1 \wedge \cdots \wedge x_n = v_n$
$\vec{z} \subseteq \vec{x}$	every element of $\vec{z}$ occurs in $\vec{x}$
$M \models \varphi$	model M satisfies formula φ
$\varphi \models_T \psi$	formula φ entails formula ψ in theory T

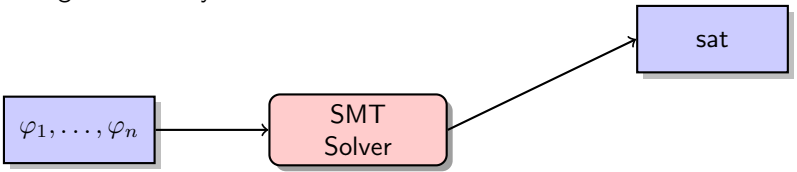
SMT Solver Basic Functionality

Background theory T



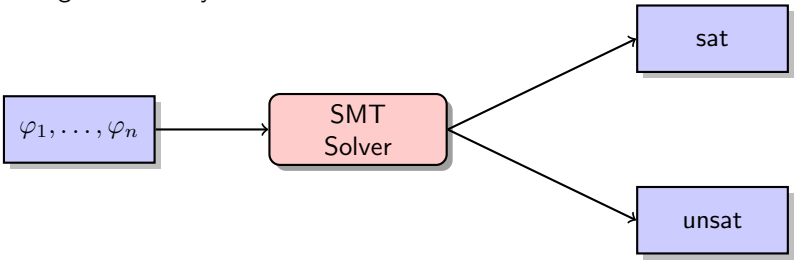
SMT Solver Basic Functionality

Background theory T



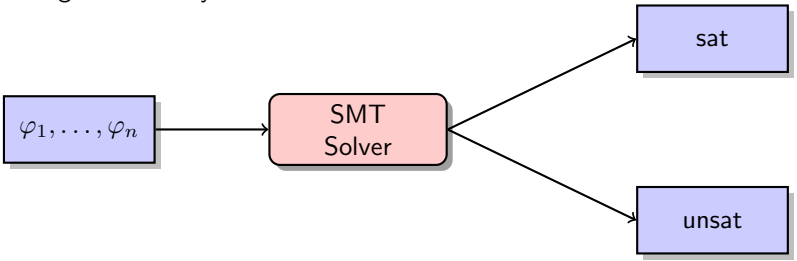
SMT Solver Basic Functionality

Background theory T



SMT Solver Basic Functionality

Background theory T

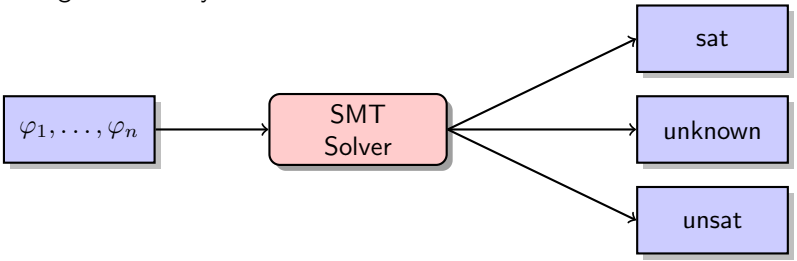


sat/unsat: there is a/no model M of T such that

$$M \models \varphi_1 \wedge \cdots \wedge \varphi_n$$

SMT Solver Basic Functionality

Background theory T



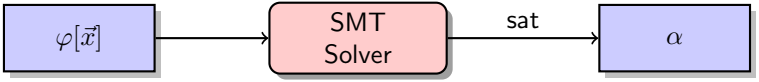
sat/unsat: there is a/no model M of T such that

$$M \models \varphi_1 \wedge \cdots \wedge \varphi_n$$

unknown: inconclusive — because of resource limits or incompleteness

SMT Solver Output: Satisfying Assignments

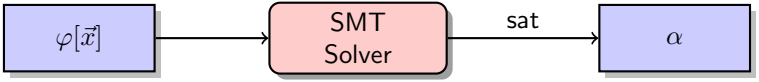
Background theory T



α is a satisfying assignment for $\vec{x} = (x_1, \ldots, x_n)$:

SMT Solver Output: Satisfying Assignments

Background theory T

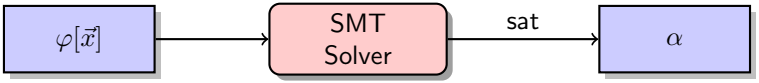


α is a satisfying assignment for $\vec{x} = (x_1, \ldots, x_n)$:

1. $\alpha = \{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\}$ for some values $\vec{v} = (v_1, \ldots, v_n)$
2. $M \models \varphi[\vec{x} \mapsto \vec{v}]$ for some model M of T

SMT Solver Output: Satisfying Assignments

Background theory T



α is a satisfying assignment for $\vec{x} = (x_1, \ldots, x_n)$:

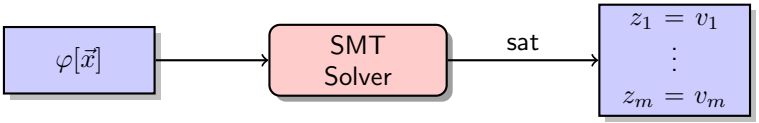
1. $\alpha = \{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\}$ for some values $\vec{v} = (v_1, \ldots, v_n)$
2. $M \models \varphi[\vec{x} \mapsto \vec{v}]$ for some model M of T

Note.

$\vec{x}$ may consist of first- and second-order variables
(aka, uninterpreted constants and function symbols)

SMT Solver Output: Sat Cores

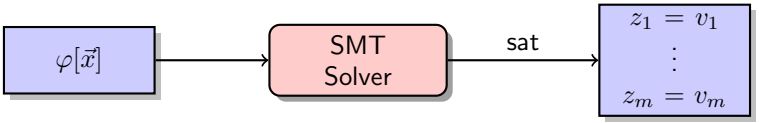
Background theory T



$\vec{z} = \vec{v}$ is a sat core for φ:

SMT Solver Output: Sat Cores

Background theory T

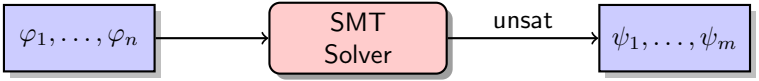


$\vec{z} = \vec{v}$ is a sat core for φ:

1. $\vec{z} \subseteq \vec{x}$
2. $\vec{y} = \vec{x} \setminus \vec{z}$
3. $\forall \vec{y} (\varphi \wedge \vec{z} = \vec{v})$ is satisfiable in T
4. $\vec{z}$ is minimal (or smallish)

SMT Solver Output: Unsat Cores

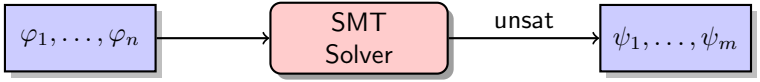
Background theory T



$\psi_1, \ldots, \psi_m$ is a unsat core of $\{\varphi_1, \ldots, \varphi_n\}$:

SMT Solver Output: Unsat Cores

Background theory T

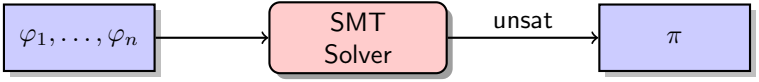


$\psi_1, \ldots, \psi_m$ is a unsat core of $\{\varphi_1, \ldots, \varphi_n\}$:

1. $\{\psi_1, \ldots, \psi_m\} \subseteq \{\varphi_1, \ldots, \varphi_n\}$
2. $\{\psi_1, \ldots, \psi_m\}$ is unsat in T
3. $\{\psi_1, \ldots, \psi_m\}$ is minimal (or smallish)

SMT Solver Output: Proofs

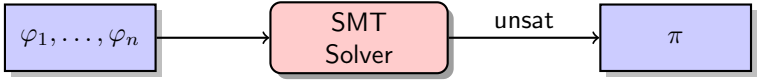
Background theory T



π is a checkable proof object for $\{\varphi_1, \ldots, \varphi_n\}$:

SMT Solver Output: Proofs

Background theory T

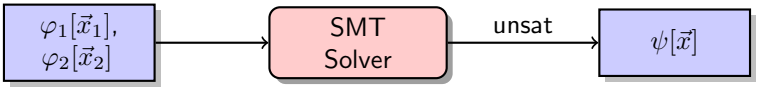


π is a checkable proof object for $\{\varphi_1, \ldots, \varphi_n\}$:

1. π is a proof term in some formal proof system
2. π expresses a refutation of $\{\varphi_1, \ldots, \varphi_n\}$
3. π can be efficiently checked by an external proof checker

Extended Functionality: Interpolation

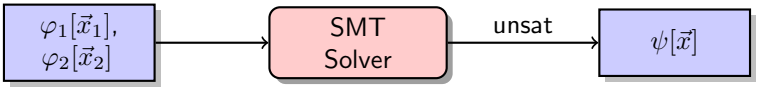
Background theory T



ψ is a logical interpolant of φ_1 and φ_2:

Extended Functionality: Interpolation

Background theory T

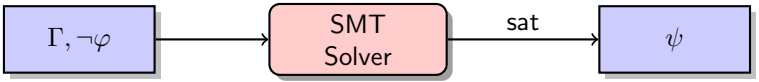


ψ is a logical interpolant of φ_1 and φ_2:

1. $\varphi_1 \models_T \psi$ and $\psi \models_T \neg\varphi_2$
2. $\vec{x} = \vec{x}_1 \cap \vec{x}_2$

Extended Functionality: Abduction

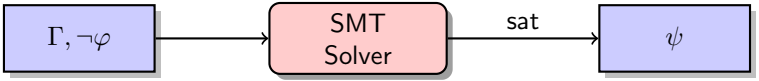
Background theory T



ψ is an abduction hypothesis for φ wrt Γ:

Extended Functionality: Abduction

Background theory T

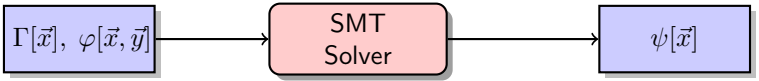


ψ is an abduction hypothesis for φ wrt Γ:

1. Γ, ψ is satisfiable in T
2. $\Gamma, \psi \models_T \varphi$
3. ψ is maximal, e.g., with respect to $\models_T$
   (if ψ' satisfies 1 and 2 and $\psi \models_T \psi'$ then $\psi' \models_T \psi$)

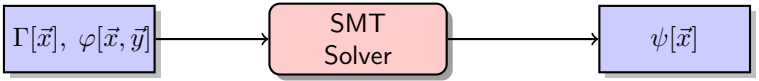
Extended Functionality: Quantifier Elimination

Background theory T



Extended Functionality: Quantifier Elimination

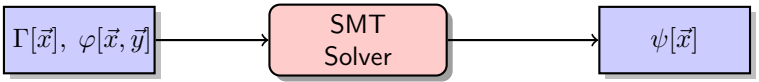
Background theory T



ψ is a projection of φ over $\vec{y}$ with respect to Γ:

Extended Functionality: Quantifier Elimination

Background theory T

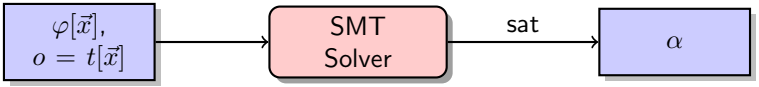


ψ is a projection of φ over $\vec{y}$ with respect to Γ:

1. $\Gamma \models_T \psi \Leftrightarrow \exists \vec{y}\, \varphi$

Extended Functionality: Optimization

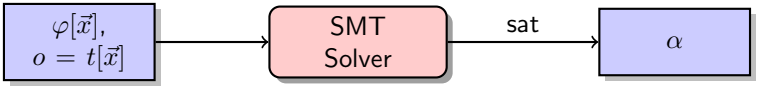
Background theory T



α is a an optimal assignment for φ:

Extended Functionality: Optimization

Background theory T



α is a an optimal assignment for φ:

1. $\alpha = \{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\}$ for some values $v_1, \ldots, v_n$
2. $M \models \varphi[\vec{x} \mapsto \vec{v}]$ for some model M of T
3. α minimizes/maximizes objective o

Background theories

Using the cvc5 SMT Solver

Choose one option from each interface

Python Interface

- Terminal

```
python3 -m venv atva-tutorial
source atva-tutorial/bin/activate
python3 -m pip install cvc5-gpl
python3
from cvc5.pythonic import *
```

- Online
  - https://colab.research.google.com/
  - !pip install cvc5-gpl
  - **from** cvc5.pythonic **import** *

Text Interface

- Terminal – download, unzip, run `./<...>/bin/cvc5`
  - cvc5-Linux-arm64-static-gpl.zip
  - cvc5-Linux-x86_64-static-gpl.zip
  - cvc5-macOS-arm64-static-gpl.zip
  - cvc5-macOS-x86_64-static-gpl.zip
  - cvc5-Win64-x86_64-static.zip (incomplete, better to use WSL)
- Online
  - https://cvc5.github.io/app/

Background Theories

Uninterpreted Funs	$x = y \Rightarrow f(x) = f(y)$
Integer/Real Arithmetic	$2x + y = 0 \wedge 2x - y = 4 \rightarrow x = 1$
Floating Point Arithmetic	$x + 1 \neq NaN \wedge x < \infty \Rightarrow x + 1 > x$
Bit-vectors	$4 \cdot (x \gg 2) = x \,\&\, \sim 3$
Strings and RegExs	$x = y \cdot z \wedge z \in ab^* \Rightarrow \lvert x\rvert > \lvert y\rvert$
Arrays	$i = j \Rightarrow \mathsf{store}(a, i, x)[j] = x$
Algebraic Data Types	$x \neq Leaf \Rightarrow \exists\, l, r : Tree(\alpha).\ \exists\, a : \alpha.$ $x = Node(l, a, r)$
Finite Sets	$e_1 \in x \wedge e_2 \in x \setminus e_1 \Rightarrow \exists y, z : Set(\alpha).$ $\lvert y\rvert = \lvert z\rvert \wedge x = y \cup z \wedge y \neq \emptyset$
Finite Relations	$(x, y) \in r \wedge (y, z) \in r \Rightarrow (x, z) \in r \bowtie s$

Equality and Uninterpreted Functions (EUF)(Nelson and Oppen 1980; Nieuwenhuis and Oliveras 2007)

Simplest first-order theory with equality, applications of uninterpreted functions, and variables of uninterpreted sorts

For all sorts σ, σ' and function symbols $f : \sigma \to \sigma'$

Reflexivity: $\forall x : \sigma.\ x = x$
Symmetry: $\forall x : \sigma.\ x = y \Rightarrow y = x$
Transitivity: $\forall x, y : \sigma.\ x = y \wedge y = z \Rightarrow x = z$
Congruence: $\forall \vec{x}, \vec{y} : \vec{\sigma}.\ \vec{x} = \vec{y} \Rightarrow f(\vec{x}) = f(\vec{y})$

Congruence closure decision procedure can efficiently handle conjunctions of equality literals.

Example

$$f(f(f(a))) = b \qquad g(f(a), b) = a \qquad f(a) = a$$

Arrays

(Bofill et al. 2008; McCarthy 1993; Moura and Bjørner 2009; Stump et al. 2001)

Operates over sorts $\mathsf{Array}(\sigma_i, \sigma_e)$, σ_i, σ_e and function symbols

$$\_[\_] : \mathsf{Array}(\sigma_i, \sigma_e) \times \sigma_i \to \sigma_e$$
$$\mathsf{store} : \mathsf{Array}(\sigma_i, \sigma_e) \times \sigma_i \times \sigma \to \mathsf{Array}(\sigma_i, \sigma_e)$$

For any index sort σ_i and element sort σ_e

Read-Over-Write-1: $\forall a, i, e.\ \mathsf{store}(a, i, e)[i] = e$
Read-Over-Write-2: $\forall a, i, j, e.\ i \neq j \Rightarrow \mathsf{store}(a, i, e)[j] = a[j]$
Extensionality: $\forall a, b, i.\ a \neq b \Rightarrow \exists i.\ a[i] \neq b[i]$

Efficient decision procedure based on congruence closure to handle equality reasoning and strong filters for restricting the application of inferences capturing the above axioms.

Example

$$\mathsf{store}(\mathsf{store}(a, i, a[j]), j, a[i]) = \mathsf{store}(\mathsf{store}(a, j, a[i]), i, a[j])$$

Arithmetic

Restricted fragments, over the reals or the integers, support efficient methods:

- Bounds: $x \bowtie k$ with $\bowtie \in \{<, >, \leq, \geq, =\}$ (Bozzano et al. 2005a)
- Difference constraints: $x - y \bowtie k$, with $\bowtie \in \{<, >, \leq, \geq, =\}$ (Cotton and Maler 2006; Nieuwenhuis and Oliveras 2005; Wang et al. 2005)
- UTVPI: $\pm x \pm y \bowtie k$, with $\bowtie \in \{<, >, \leq, \geq, =\}$ (Lahiri and Musuvathi 2005)
- Linear arithmetic, e.g: $2x - 3y + 4z \leq 5$ (Bjørner and Nachmanson 2024; Dutertre and Moura 2006)
- Non-linear arithmetic, e.g: $2xy + 4xz^2 - 5y \leq 10$ (Ábrahám et al. 2021; Borralleras et al. 2009; Jovanović and Moura 2012; Zankl and Middeldorp 2010)

Example

Are there real solutions for $x^2y + yz + 2xyz + 4xy + 8xz + 16 = 0$?

Machine Arithmetic — Bit-vectors (Brummayer and Biere 2009; Niemetz and Preiner 2023)

Combines arithmetic operations, bit-wise operations, shift, extraction, concatenation.

Most effective decision procedures rely primarily on bit-blasting, i.e., converting the bit-vector problem to an equisatisfiable Boolean representation and leveraging state-of-the-art SAT solvers.

Example

Consider the following implementations of the absolute value opeartor for 32-bit integers:

0. $abs_0(x) := x < 0 \; ? \; -x : x$
1. $abs_1(x) := (x \oplus (x \gg_a 31)) - (x \gg_a 31)$
2. $abs_2(x) := (x + (x \gg_a 31)) \oplus (x \gg_a 31)$
3. $abs_3(x) := x - ((x \ll 1) \;\&\; (x \gg_a 31))$

How do we prove that all four are equivalent to one another?

Machine Arithmetic – Floating-Points

(Brain et al. 2019, 2014; Conchon et al. 2017)

FP in SMT

- Follows IEEE 754-2019
- FP number = triple of bit-vectors
- Wide range of operators
  - take a rounding mode as input
- E.g., addition, multiplication, fused-multiplication-addition
- As with bit-vectors, most effective procedures rely on bit-blasting.

Example

Is addition associative in floating-point arithmetic, i.e., is $a + (b + c) \neq (a + b) + c$ valid?

(Co-)Algebraic Data Types

(Barrett et al. 2007; Reynolds and Blanchette 2017)

Family of user-definable theories

Example

$$\mathsf{Tree} \quad := \quad \mathsf{nil} \mid \mathsf{node}(\mathsf{data} : \mathsf{Int}, \mathsf{left} : \mathsf{Tree}, \mathsf{right} : \mathsf{Tree})$$

Distinctiveness: $\forall h, t.\ \mathsf{nil} \neq h :: t$

Exhaustiveness: $\forall l.\ l = \mathsf{nil} \vee \exists h, t.\ h :: t$

Injectivity: $\forall h_1, h_2, t_1, t_2.$
$h_1 :: t_1 = h_2 :: t_2 \Rightarrow h_1 = h_2 \wedge t_1 = t_2$

Selectors: $\forall h, t.\ \mathsf{head}(h :: t) = h \wedge \mathsf{tail}(h :: t) = t$

(**Non-circularity**: $\forall l, x_1, \ldots, x_n.\ l \neq x_1 :: \cdots :: x_n :: l$)

Strings and regular expressions (Abdulla et al. 2015; Kiezun et al. 2009; Liang et al. 2014)

SMT Strings

- Represent common programming languages Unicode strings
- Supports a wide range of operators
  - concatenation, length, substring, etc
- Regular expressions crucial for some applications, such as analysis of access control policies

Example

Can we have a string with at most three characters that also contains the string "ATVA"?

Other Interesting Theories

- Finite sets with cardinality (Bansal et al. 2016)
- Finite relations (Meng et al. 2017)
- Transcendental Functions (Cimatti et al. 2017; Gao et al. 2013)
- Ordinary differential equations (Gao et al. 2013)
- Finite Fields (Hader et al. 2023; Ozdemir et al. 2023)
- ...

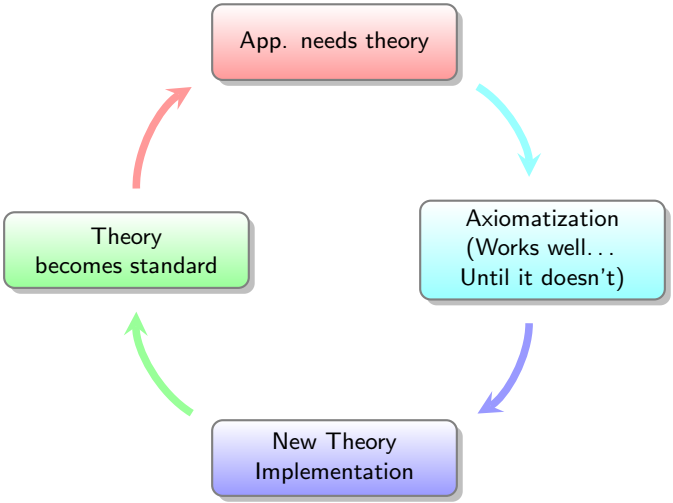
Some SMT solvers also allow you to axiomatize your own theory

- The effective procedures discussed so far generally assume quantifier-free logical fragments
- However new applications may not fit directly into existing theories, which necessitates reasoning about user-defined axioms
- Some solvers (notably, cvc5 and Z3) support them, but this support has caveats
  - Undecidable in general
  - Explosive heuristics
  - Users want it to work as well as on quantifier-free problems

Example

What if we did not have a theory of arrays but wanted to reason about them?

The SMT Cycle



Applications

Agenda

1. Introduction
2. SMT solver functionality
3. Background theories
4. Applications
   - Model Checking
   - Synthesis
   - Software Verification
   - Misc
5. Producing and checking proof certificates

Bounded Model Checking

To check the reachability of a class S of bad states
for a system model M:

Bounded Model Checking

To check the reachability of a class S of bad states for a system model M:

1. Choose a theory T decided by an SMT solver (e.g., quantifier-free linear arithmetic and EUF)

Bounded Model Checking

To check the reachability of a class S of bad states
for a system model M:

1. Choose a theory T decided by an SMT solver
   (e.g., quantifier-free linear arithmetic and EUF)
2. Represent system states as values for a tuple $\vec{x}$ of state vars

Bounded Model Checking

To check the reachability of a class S of bad states for a system model M:

1. Choose a theory T decided by an SMT solver (e.g., quantifier-free linear arithmetic and EUF)
2. Represent system states as values for a tuple $\vec{x}$ of state vars
3. Encode system M as T-formulas $(I[\vec{x}], R[\vec{x}, \vec{x}'])$ where
   - I encodes M's initial state condition and
   - R encodes M's transition relation

Bounded Model Checking

To check the reachability of a class S of bad states
for a system model M:

1. Choose a theory T decided by an SMT solver
   (e.g., quantifier-free linear arithmetic and EUF)
2. Represent system states as values for a tuple $\vec{x}$ of state vars
3. Encode system M as T-formulas $(I[\vec{x}], R[\vec{x}, \vec{x}'])$
   where
   - I encodes M's initial state condition and
   - R encodes M's transition relation
4. Encode S as a T-formula $B[\vec{x}]$

Bounded Model Checking

To check the reachability of a class S of bad states
for a system model M:

1. Choose a theory T decided by an SMT solver
   (e.g., quantifier-free linear arithmetic and EUF)
2. Represent system states as values for a tuple $\vec{x}$ of state vars
3. Encode system M as T-formulas $(I[\vec{x}], R[\vec{x}, \vec{x}'])$
   where
   - I encodes M's initial state condition and
   - R encodes M's transition relation
4. Encode S as a T-formula $B[\vec{x}]$
5. Find a k such that $I[\vec{x}_0] \wedge R[\vec{x}_0, \vec{x}_1] \wedge \cdots \wedge R[\vec{x}_{k-1}, \vec{x}_k] \wedge B[\vec{x}_k]$ is satisfiable in T

Bounded Model Checking

We can for example check if safety property P holds for 10 iterations.

- ▷ Unroll the loop 10 times or until property P is violated
- ▷ Check for each iteration if property P holds

C Code

```
int main () {
  bool turn;                    // input
  uint32_t a = 0, b = 0;        // states
  for (;;) {
   turn = read_bool ();
   assert (a != 3 || b != 3); // property P
   if (turn) a = a + 1;         // next(a)
   else      b = b + 1;         // next(b)
  }
}
```

Unroll

$a_0 = 0 \wedge b_0 = 0$
...check if P holds for a_0, b_0
$a_1 = next(a_0) \wedge b_1 = next(b_0)$
...check if P holds for a_1, b_1
$a_2 = next(a_1) \wedge b_2 = next(b_1)$
...check if P holds for a_2, b_2
...

Symbolic Model Checking

To check the invariance of a state property S
for a system model M:

1. Choose a theory T decided by an SMT solver
   (e.g., quantifier-free linear arithmetic and EUF)
2. Represent system states as values for a tuple $\vec{x}$ of state vars
3. Encode system M as T-formulas $(I[\vec{x}], R[\vec{x}, \vec{x}'])$
   where
   - I encodes M's initial state condition and
   - R encodes M's transition relation
4. Encode S as a T-formula $P[\vec{x}]$

Symbolic Model Checking

To check the invariance of a state property S
for a system model M:

1. Choose a theory T decided by an SMT solver
   (e.g., quantifier-free linear arithmetic and EUF)
2. Represent system states as values for a tuple $\vec{x}$ of state vars
3. Encode system M as T-formulas $(I[\vec{x}], R[\vec{x}, \vec{x}'])$
   where
   - I encodes M's initial state condition and
   - R encodes M's transition relation
4. Encode S as a T-formula $P[\vec{x}]$
5. Prove that $P[\vec{x}]$ holds in all reachable states of $(I[\vec{x}], R[\vec{x}, \vec{x}'])$

Symbolic Model Checking

Example: *Parametric Resettable Counter*

System

Vars

input pos int, n_0
input bool r
int c, n

Initialization

c := 1
n := n_0

Transitions

n' := n
c' := if (r' or c = n)
then 1
else c + 1

Property

$c \leq n + 1$

Symbolic Model Checking

Example: *Parametric Resettable Counter*

System

Vars

input pos int, n_0
input bool r
int c, n

Initialization

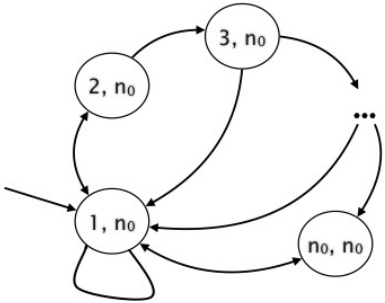
c := 1
n := n_0

Transitions

n' := n
c' := if (r' or c = n)
then 1
else c + 1

Property

$c \leq n + 1$



The transition relation contains infinitely many instances of the schema above, one for each $n_0 > 0$

Symbolic Model Checking

Example: *Parametric Resettable Counter*

System

Vars

input pos int, n_0
input bool r
int c, n

Initialization

c := 1
n := n_0

Transitions

n' := n
c' := if (r' or c = n)
then 1
else c + 1

Property

$c \leq n + 1$

Encoding in T = LIA

$$\begin{aligned}
\vec{x} &:= (c, n, r, n_0) \\
I[\vec{x}] &:= c = 1 \\
&\quad \wedge \; n = n_0 \\
R[\vec{x}, \vec{x}'] &:= n' = n \\
&\quad \wedge \; (\neg r' \wedge c \neq n \vee c' = 1) \\
&\quad \wedge \; (r' \vee c = n \vee c' = c + 1) \\
P[\vec{x}] &:= c \leq n + 1
\end{aligned}$$

Inductive Reasoning

$M = (I[\vec{x}], R[\vec{x}, \vec{x}'])$

Inductive Reasoning

$M = (I[\vec{x}], R[\vec{x}, \vec{x}'])$

To prove $P[x]$ invariant for M it suffices
to show that it is inductive for M,
i.e.,

(1) $I[\vec{x}] \models_T P[\vec{x}]$ (base case)
and
(2) $P[\vec{x}] \wedge R[\vec{x}, \vec{x}'] \models_T P[\vec{x}']$ (inductive step)

Inductive Reasoning

Problem: Not all invariants are inductive

For the parametric resettable counter,
$P := c \leq n + 1$ is invariant but (2) is falsifiable
e.g., by $(c, n, r) = (4, 3, false)$ and $(c, n, r)' = (5, 3, false)$

$M = (I[\vec{x}$

To prove $P[\vec{x}]$ invariant for M it suffices
to show that it is inductive for M,
i.e.,

(1) $I[\vec{x}] \models_T P[\vec{x}]$ (base case)
and
(2) $P[\vec{x}] \wedge R[\vec{x}, \vec{x}'] \models_T P[\vec{x}']$ (inductive step)

Strengthening Inductive Reasoning

(1) $I[\vec{x}] \models_T P[\vec{x}]$ (2) $P[\vec{x}] \wedge R[\vec{x}, \vec{x}'] \models_T P[\vec{x}']$

Various approaches:

Strengthening Inductive Reasoning

(1) $I[\vec{x}] \models_T P[\vec{x}]$ (2) $P[\vec{x}] \wedge R[\vec{x}, \vec{x}'] \models_T P[\vec{x}']$

Various approaches:

Strengthen P: find a property Q such that $Q[\vec{x}] \models_T P[\vec{x}]$ and prove Q inductive
(ex., interpolation-based MC, IC3, CHC)

Strengthening Inductive Reasoning

(1) $I[\vec{x}] \models_T P[\vec{x}]$ (2) $P[\vec{x}] \wedge R[\vec{x}, \vec{x}'] \models_T P[\vec{x}']$

Various approaches:

Strengthen P: find a property Q such that $Q[\vec{x}] \models_T P[\vec{x}]$ and prove Q inductive
(ex., interpolation-based MC, IC3, CHC)

Strengthen R: find an auxiliary invariant $Q[\vec{x}]$ and use $Q[\vec{x}] \wedge R[\vec{x}, \vec{x}'] \wedge Q[\vec{x}']$ instead of $R[\vec{x}, \vec{x}']$
(ex:, Houdini, invariant sifting)

Strengthening Inductive Reasoning

(1) $I[\vec{x}] \models_T P[\vec{x}]$ (2) $P[\vec{x}] \wedge R[\vec{x}, \vec{x}'] \models_T P[\vec{x}']$

Various approaches:

Strengthen P: find a property Q such that $Q[\vec{x}] \models_T P[\vec{x}]$ and prove Q inductive
(ex., interpolation-based MC, IC3, CHC)

Strengthen R: find an auxiliary invariant $Q[\vec{x}]$ and use $Q[\vec{x}] \wedge R[\vec{x}, \vec{x}'] \wedge Q[\vec{x}']$ instead of $R[\vec{x}, \vec{x}']$
(ex:, Houdini, invariant sifting)

Lengthen R: Consider increasingly longer R-paths $R[\vec{x}_0, \vec{x}_1] \wedge \cdots \wedge R[\vec{x}_{k-1}, \vec{x}_k] \wedge R[\vec{x}_k, \vec{x}_{k+1}]$
(ex:, k-induction)

Agenda

1. Introduction
2. SMT solver functionality
3. Background theories
4. Applications
   - Model Checking
   - Synthesis
   - Software Verification
   - Misc
5. Producing and checking proof certificates

Program Synthesis

Synthesis

- Synthesize a function that satisfies a given high-level specification
- Already used extensively for hardware systems, but particularly challenging for software
- Recent direction: syntax-guided synthesis (SyGuS)
  - Specification is given by (second-order) $\mathcal{T}$-formula: $\exists f. \forall \vec{x}. \varphi[f, \vec{x}]$
  - Syntactic restrictions given by context-free grammar G

Program Synthesis

Synthesis

- Synthesize a function that satisfies a given high-level specification
- Already used extensively for hardware systems, but particularly challenging for software
- Recent direction: syntax-guided synthesis (SyGuS)
  - Specification is given by (second-order) $\mathcal{T}$-formula: $\exists f. \forall \vec{x}.\ \varphi[f, \vec{x}]$
  - Syntactic restrictions given by context-free grammar G

Invariant Synthesis via SyGuS

The SyGuS invariant problem for theory T is, given state variables $\vec{x}$, *initial condition* $I[\vec{x}]$, *transition relation* $R[\vec{x}, \vec{x}']$, and *property* $P[\vec{x}]$, theory T and grammar G, to find a solution Inv such that:

- $\mathsf{I}[\vec{x}] \models_T \mathsf{Inv}[\vec{x}]$,
- $\mathsf{Inv}[\vec{x}] \wedge \mathsf{R}[\vec{x}, \vec{x}'] \models_T \mathsf{Inv}[\vec{x}']$
- $\mathsf{Inv}[\vec{x}] \models_T \mathsf{P}[\vec{x}]$
- Inv is generated by a context-free grammar G.

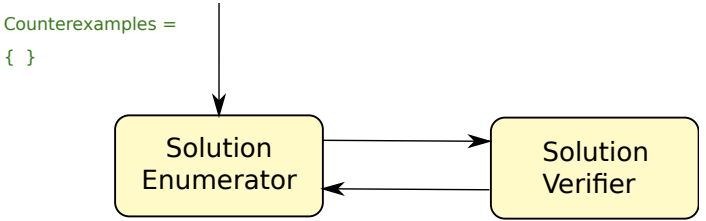
SyGuS solving: enumerative CEGIS

Solar-Lezama et al. 2006; Udupa et al. 2013

Consider the example:

$$\varphi = f(x, x) \simeq x + 1 \wedge f(x, x + 1) \simeq x$$

$$R = \begin{array}{l} A \to 0 \mid 1 \mid x \mid y \mid A + A \mid \mathrm{ite}(B, A, A) \\ B \to A \leq A \mid \neg B \end{array}$$



▷ De facto approach to SyGuS solving given its simplicity and efficacy

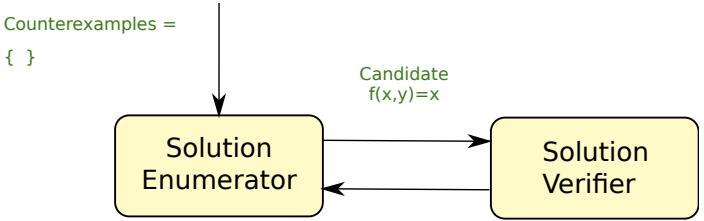
SyGuS solving: enumerative CEGIS

Solar-Lezama et al. 2006; Udupa et al. 2013

Consider the example:

$$\varphi = f(x, x) \simeq x + 1 \wedge f(x, x + 1) \simeq x$$

$$R = \begin{array}{l} A \to 0 \mid 1 \mid x \mid y \mid A + A \mid \text{ite}(B, A, A) \\ B \to A \leq A \mid \neg B \end{array}$$



▷ De facto approach to SyGuS solving given its simplicity and efficacy

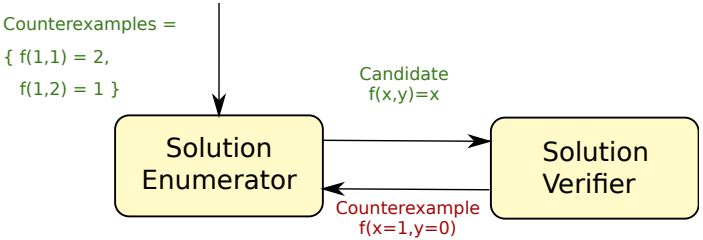
SyGuS solving: enumerative CEGIS

Solar-Lezama et al. 2006; Udupa et al. 2013

Consider the example:

$$\varphi \quad = \quad f(x,\, x) \simeq x + 1 \;\wedge\; f(x,\, x+1) \simeq x$$

$$R \quad = \quad \begin{array}{l} A \to 0 \mid 1 \mid x \mid y \mid A + A \mid \mathrm{ite}(B,\, A,\, A) \\ B \to A \leq A \mid \neg B \end{array}$$



▷ De facto approach to SyGuS solving given its simplicity and efficacy

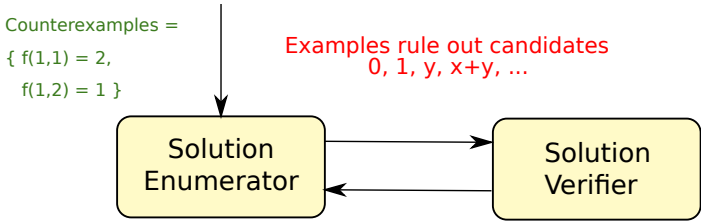
SyGuS solving: enumerative CEGIS

Solar-Lezama et al. 2006; Udupa et al. 2013

Consider the example:

$$\varphi = f(x, x) \simeq x + 1 \wedge f(x, x + 1) \simeq x$$

$$R = \begin{array}{l} A \to 0 \mid 1 \mid x \mid y \mid A + A \mid \mathrm{ite}(B, A, A) \\ B \to A \leq A \mid \neg B \end{array}$$



▷ De facto approach to SyGuS solving given its simplicity and efficacy

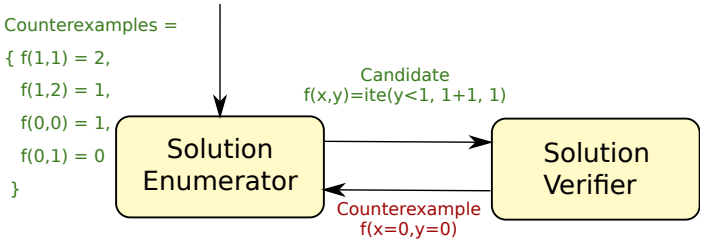
SyGuS solving: enumerative CEGIS

Solar-Lezama et al. 2006; Udupa et al. 2013

Consider the example:

$$\varphi \quad = \quad f(x,\, x) \simeq x + 1 \;\wedge\; f(x,\, x+1) \simeq x$$

$$R \quad = \quad \begin{array}{l} A \to 0 \mid 1 \mid x \mid y \mid A + A \mid \mathrm{ite}(B,\, A,\, A) \\ B \to A \leq A \mid \neg B \end{array}$$



▷ De facto approach to SyGuS solving given its simplicity and efficacy

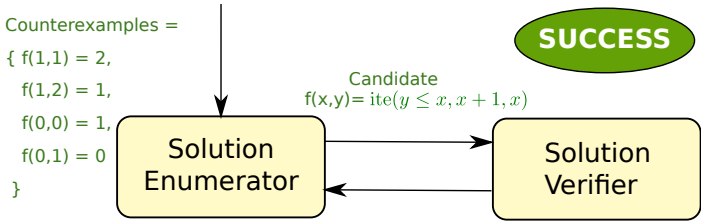
SyGuS solving: enumerative CEGIS

Solar-Lezama et al. 2006; Udupa et al. 2013

Consider the example:

$$\varphi \quad = \quad f(x,\, x) \simeq x + 1 \;\wedge\; f(x,\, x+1) \simeq x$$

$$R \quad = \quad \begin{array}{l} A \to 0 \mid 1 \mid x \mid y \mid A + A \mid \mathrm{ite}(B,\, A,\, A) \\ B \to A \leq A \mid \neg B \end{array}$$



▷ De facto approach to SyGuS solving given its simplicity and efficacy

Enumerative SyGuS in SMT

(Reynolds et al. 2017, 2018)

- Encode problem using a deep embedding into datatypes

$$\varphi = f(x, x) \simeq x + 1 \wedge f(x, x + 1) \simeq x$$

$$R = \begin{array}{l} A \to 0 \mid 1 \mid x \mid y \mid A + A \mid \mathrm{ite}(B,\ A,\ A) \\ B \to A \leq A \mid \neg B \end{array}$$

Becomes

$$[\![\varphi]\!] = \mathrm{eval_a}(d, x, x) \simeq x + 1 \wedge \mathrm{eval_a}(d, x, x + 1) \simeq x$$

$$[\![R]\!] = \begin{array}{ll} \mathrm{a} & = \mathrm{Zero} \mid \mathrm{One} \mid \mathrm{X} \mid \mathrm{Y} \mid \mathrm{Plus(a, a)} \mid \mathrm{Ite(b,\ a,\ a)} \\ \mathrm{b} & = \mathrm{Leq(a,\ a)} \mid \mathrm{Neg(b)} \end{array}$$

- eval maps datatype terms to their corresponding theory terms
  - $\mathrm{eval_a}(\mathrm{Plus}(\mathrm{X}, \mathrm{X}), 2, 3)$ is interpreted as $(x + x)\{x \mapsto 2, y \mapsto 3\} = 4$

Enumerative SyGuS in SMT

(Reynolds et al. 2017, 2018)

- Encode problem using a deep embedding into datatypes

$$\varphi = f(x, x) \simeq x + 1 \wedge f(x, x + 1) \simeq x$$

$$R = \begin{array}{l} A \to 0 \mid 1 \mid x \mid y \mid A + A \mid \mathrm{ite}(B,\ A,\ A) \\ B \to A \leq A \mid \neg B \end{array}$$

Becomes

$$[\![\varphi]\!] = \mathrm{eval_a}(d, x, x) \simeq x + 1 \wedge \mathrm{eval_a}(d, x, x + 1) \simeq x$$

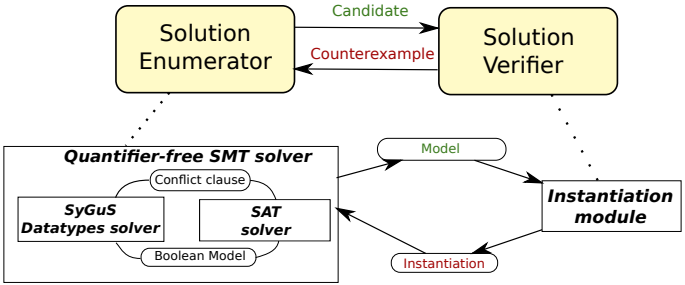
$$[\![R]\!] = \begin{array}{ll} \mathrm{a} & = \mathrm{Zero} \mid \mathrm{One} \mid \mathrm{X} \mid \mathrm{Y} \mid \mathrm{Plus(a, a)} \mid \mathrm{Ite(b,\ a,\ a)} \\ \mathrm{b} & = \mathrm{Leq(a,\ a)} \mid \mathrm{Neg(b)} \end{array}$$

- eval maps datatype terms to their corresponding theory terms
  - $\mathrm{eval_a}(\mathrm{Plus(X, X)}, 2, 3)$ is interpreted as $(x + x)\{x \mapsto 2, y \mapsto 3\} = 4$

- A solution is a model in which e.g.
  - $d = \mathrm{Ite(Leq(Y,\ X),\ Plus(X, One),\ X)}$, corresponding to
  - $f = \lambda xy.\ \mathrm{ite}(y \leq x,\ x + 1,\ x)$

Enumerative SyGuS in SMT

Reynolds et al. 2017, Reynolds et al. 2018



- An instantiation module checks candidates against the specification
  - Generates lemmas witnessing why a candidate failed
- A specialized datatypes solver for SyGuS generates candidate solutions
  - Must satisfy all lemmas
  - Dedicated pruning
  - Parameterizable fairness criteria for enumeration

Agenda

Software Verification

Example

```
void swap(int* a, int* b) {
  *a = *a + *b;
  *b = *a - *b;
  *a = *a - *b;
}
```

Check if the swap is correct:

- Heap: $Array(BV_{32}) \mapsto BV_{32}$
- Update heap line by line
- Check that
  a* = old(b*) and b* = old(a*)

Software Verification

Example

```
void swap(int* a, int* b) {
  *a = *a + *b;
  *b = *a - *b;
  *a = *a - *b;
}
```

Check if the swap is correct:

- Heap: $Array(BV_{32}) \mapsto BV_{32}$
- Update heap line by line
- Check that
  a* = old(b*) and b* = old(a*)

$$h_1 = \mathsf{store}(h_0, a, h_0[a] +_{32} h_0[b])$$
$$h_2 = \mathsf{store}(h_1, b, h_1[a] -_{32} h_1[b])$$
$$h_3 = \mathsf{store}(h_2, a, h_2[a] -_{32} h_2[b])$$
$$\neg(h_3[a] = h_0[b] \wedge h_3[b] = h_0[a])$$

Software Verification

Example

```
void swap(int* a, int* b) {
  *a = *a + *b;
  *b = *a - *b;
  *a = *a - *b;
}
```

Check if the swap is correct:

- Heap: $Array(BV_{32}) \mapsto BV_{32}$
- Update heap line by line
- Check that
  a* = old(b*) and b* = old(a*)
- Incorrect: aliasing

SMT solver solution

$$a \mapsto 0, \quad b \mapsto 0$$
$$h_0[0] \mapsto 1, \quad h_1[0] \mapsto 2$$
$$h_2[0] \mapsto 0, \quad h_3[0] \mapsto 0$$

$$_{32}\, h_0[b])$$
$$_{32}\, h_1[b])$$
$$h_3 = \mathsf{store}(h_2, a, h_2[a] -_{32} h_2[b])$$
$$\neg(h_3[a] = h_0[b] \wedge h_3[b] = h_0[a])$$

Contract-based Software Verification

Example (Binary Search)

```
//@assume 0 <= n <= |a| &&
//        foreach i in [0..n-2]. a[i] <= a[i+1]
//@ensure (0 <= res ==> a[res] = k) &&
//        (res < 0 ==> foreach i in [0..n-1]. a[i] != k)
int BinarySearch(int[] a, int n, int k) {
  int l = 0;  int h = n;
  while (l < h) { // Find middle value
    //@invariant 0 <= low < high <= len <= |a| &&
    //           foreach i in [0..low-1]. a[i]<k &&
    //           foreach i in [high..len-1]. a[i] > k
    int m = l + (h - l) / 2;  int v = a[m];
    if (k < v) { l = m + 1; } else if (v < k) { h = m; }
    else { return m; }
  }
  return -1;
}
```

Example adapted from Moura and Bjørner 2010

Contract-based Software Verification

Example (Binary Search)

```
//@as
//
//@er
//
int B
  int l = 0;  int h = n;
  while (l < h) { // Find middle value
    //@invariant 0 <= low < high <= len <= |a| &&
    //           foreach i in [0..low-1]. a[i]<k &&
    //           foreach i in [high..len-1]. a[i] > k
    int m = l + (h - l) / 2;  int v = a[m];
    if (k < v) { l = m + 1; } else if (v < k) { h = m; }
    else { return m; }
  }
  return -1;
}
```

Main approach

1. Compile source and annotations to a series of pre-conditions, commands over the state, and post-conditions.
2. Generate verification conditions on SMT

Example adapted from Moura and Bjørner 2010

Contract-based Software Verification

$$pre = 0 \leq n \leq |a| \wedge \forall i : \mathsf{Int}\ \ 0 \leq i \wedge i \leq n-2 \Rightarrow a[i] \leq a[i+1]$$

$$\begin{aligned} post = {} & (0 \leq res \Rightarrow a[res] = k) \wedge \\ & (res < 0 \Rightarrow \forall i : \mathsf{Int}\ 0 \leq i \wedge i \leq n-1 \Rightarrow a[i] \neq k) \end{aligned}$$

$$\begin{aligned} inv = {} & 0 \leq l \wedge l \leq h \wedge h \leq n \wedge n \leq |a| \wedge \\ & \forall i : \mathsf{Int}\ \ 0 \leq i \wedge i \leq l-1 \Rightarrow a[i] < k \wedge \\ & \forall i : \mathsf{Int}\ \ h \leq i \wedge i \leq n-1 \Rightarrow a[i] > k \end{aligned}$$

Contract-based Software Verification

$$pre = 0 \leq n \leq |a| \wedge \forall i : \mathsf{Int}\ \ 0 \leq i \wedge i \leq n-2 \Rightarrow a[i] \leq a[i+1]$$

$$\begin{aligned} post = {} & (0 \leq res \Rightarrow a[res] = k) \wedge \\ & (res < 0 \Rightarrow \forall i : \mathsf{Int}\ 0 \leq i \wedge i \leq n-1 \Rightarrow a[i] \neq k) \end{aligned}$$

$$\begin{aligned} inv = {} & 0 \leq l \wedge l \leq h \wedge h \leq n \wedge n \leq |a| \wedge \\ & \forall i : \mathsf{Int}\ \ 0 \leq i \wedge i \leq l-1 \Rightarrow a[i] < k \wedge \\ & \forall i : \mathsf{Int}\ \ h \leq i \wedge i \leq n-1 \Rightarrow a[i] > k \end{aligned}$$

$$\begin{aligned} & pre \wedge \neg \textbf{let}\ l = 0,\ h = n\ \textbf{in}\ inv \wedge \forall l, h : \mathsf{Int}\ inv \Rightarrow \\ & \quad (\neg(l < h) \Rightarrow post\{res \mapsto -1\}) \wedge \\ & \quad (l < h \Rightarrow \textbf{let}\ m = l + (h-l)/2,\ v = a[m]\ \textbf{in} \\ & \qquad (k < v \Rightarrow inv\{l \mapsto m+1\}) \wedge \\ & \qquad (\neg(k < v) \wedge v < k \Rightarrow inv\{n \mapsto m\}) \wedge \\ & \qquad (\neg(k < v) \wedge \neg(v < k) \Rightarrow post\{res \mapsto m\})) \end{aligned}$$

Contract-based Software Verification

$$pre = 0 \leq n \leq |a| \wedge \forall i : \mathsf{Int}\ \ 0 \leq i \wedge i \leq n - 2 \Rightarrow a[i] \leq a[i+1]$$

$$post = (0 \leq res \Rightarrow a[res] = k) \wedge$$
$$(res < 0 \Rightarrow \forall i : \mathsf{Int}\ 0 \leq i \wedge i \leq n - 1 \Rightarrow a[i] \neq k)$$

$$inv = 0 \leq l \wedge l \leq h \wedge h \leq n \wedge n \leq |a| \wedge$$
$$\forall i : \mathsf{Int}\ \ 0 \leq i \wedge i$$
$$\forall i : \mathsf{Int}\ \ h \leq i \wedge i$$

$$pre \wedge \neg \mathbf{let}\ l = 0,\ h = n$$
$$(\neg(l < h) \Rightarrow post\{res \mapsto -1\}) \wedge$$
$$(l < h \Rightarrow \mathbf{let}\ m = l + (h - l)/2,\ v = a[m]\ \mathbf{in}$$
$$(k < v \Rightarrow inv\{l \mapsto m + 1\}) \wedge$$
$$(\neg(k < v) \wedge v < k \Rightarrow inv\{n \mapsto m\}) \wedge$$
$$(\neg(k < v) \wedge \neg(v < k) \Rightarrow post\{res \mapsto m\}))$$

SMT solver answer

Unsatisfiable

Agenda

Scheduling

Example

Schedule n jobs, each composed of m consecutive tasks, on m machines.

Schedule in 8 time slots

$d_{i,j}$	Mach. 1	Mach. 2
Job 1	2	1
Job 2	3	1
Job 3	2	3

Scheduling

Example

Schedule n jobs, each composed of m consecutive tasks, on m machines.

$$(t_{1,1} \geq 0) \wedge (t_{1,2} \geq t_{1,1} + 2) \wedge (t_{1,2} + 1 \leq 8)$$
$$(t_{2,1} \geq 0) \wedge (t_{2,2} \geq t_{2,1} + 3) \wedge (t_{2,2} + 1 \leq 8)$$
$$(t_{3,1} \geq 0) \wedge (t_{3,2} \geq t_{3,1} + 2) \wedge (t_{3,2} + 3 \leq 8)$$
$$((t_{1,1} \geq t_{2,1} + 3) \vee (t_{2,1} \geq t_{1,1} + 2))$$
$$((t_{1,1} \geq t_{3,1} + 2) \vee (t_{3,1} \geq t_{1,1} + 2))$$
$$((t_{2,1} \geq t_{3,1} + 2) \vee (t_{3,1} \geq t_{2,1} + 3))$$
$$((t_{1,2} \geq t_{2,2} + 1) \vee (t_{2,2} \geq t_{1,2} + 1))$$
$$((t_{1,2} \geq t_{3,2} + 3) \vee (t_{3,2} \geq t_{1,2} + 1))$$
$$((t_{2,2} \geq t_{3,2} + 3) \vee (t_{3,2} \geq t_{2,2} + 1))$$

Schedule in 8 time slots

$d_{i,j}$	Mach. 1	Mach. 2
Job 1	2	1
Job 2	3	1
Job 3	2	3

Scheduling

Example

Schedule n jobs, each composed of m consecutive tasks, on m machines.

$$(t_{1,1} \geq 0) \wedge (t_{1,2} \geq t_{1,1} + 2) \wedge (t_{1,2} + 1 \leq 8)$$
$$(t_{2,1} \geq 0) \wedge (t_{2,2} \geq t_{2,1} + 3) \wedge (t_{2,2} + 1 \leq 8)$$
$$(t_{3,1} \geq 0) \wedge (t_{3,2} \geq t_{3,1} + 2) \wedge (t_{3,2} + 3 \leq 8)$$
$$\vee\ (t_{2,1} \geq t_{1,1} + 2))$$
$$\vee\ (t_{3,1} \geq t_{1,1} + 2))$$
$$\vee\ (t_{3,1} \geq t_{2,1} + 3))$$
$$\vee\ (t_{2,2} \geq t_{1,2} + 1))$$
$$((t_{1,2} \geq t_{3,2} + 3) \vee (t_{3,2} \geq t_{1,2} + 1))$$
$$((t_{2,2} \geq t_{3,2} + 3) \vee (t_{3,2} \geq t_{2,2} + 1))$$

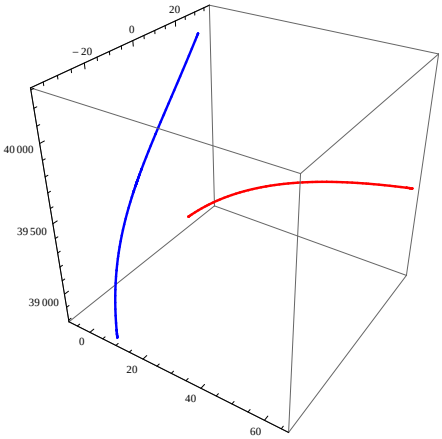
SMT solver solution

$t_{1,1} \mapsto 5,\ t_{1,2} \mapsto 7$
$t_{2,1} \mapsto 2,\ t_{2,2} \mapsto 6$
$t_{3,1} \mapsto 0,\ t_{3,2} \mapsto 3$

Schedule in 8 time slots

$d_{i,j}$	Mach. 1	Mach. 2
Job 1	2	1
Job 2	3	1
Job 3	2	3

Aircraft Trajectory Conflict Detection



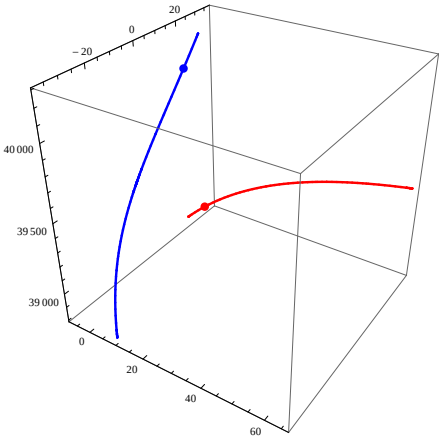
$$T_1^x(t) = 3.2484 + 270.7t + 433.12t^2 - 324.83999t^3$$
$$T_1^y(t) = 15.1592 + 108.28t + 121.2736t^2 - 649.67999t^3$$
$$T_1^z(t) = 38980.8 + 5414t - 21656t^2 + 32484t^3$$

$$T_2^x(t) = 1.0828 - 135.35t + 234.9676t^2 2 + 3248.4t^3$$
$$T_2^y(t) = 18.40759 - 230.6364t - 121.2736t^2 - 649.67999t^3$$
$$T_2^z(t) = 40280.15999 - 10828t + 24061.9816t^2 - 32484t^3$$

$H = 5\,nm \qquad V = 1000\,ft \qquad 0 \leq t \leq \frac{1}{20}h$

$$|T_1^z(t) - T_2^z(t)| \leq V$$
$$(T_1^x(t) - T_2^x(t))^2 + (T_1^y(t) - T_2^y(t))^2 \leq H^2$$

Aircraft Trajectory Conflict Detection



SMT solver solution

$t \mapsto \frac{319}{16384} \approx 0.019470215$

$+ 433.12t^2 - 324.83999t^3$

$t + 121.2736t^2 - 649.67999t^3$

$T_1^z(t) = 38980.8 + 5414t - 21656t^2 + 32484t^3$

$T_2^x(t) = 1.0828 - 135.35t + 234.9676t^2 2 + 3248.4t^3$

$T_2^y(t) = 18.40759 - 230.6364t - 121.2736t^2 - 649.67999t^3$

$T_2^z(t) = 40280.15999 - 10828t + 24061.9816t^2 - 32484t^3$

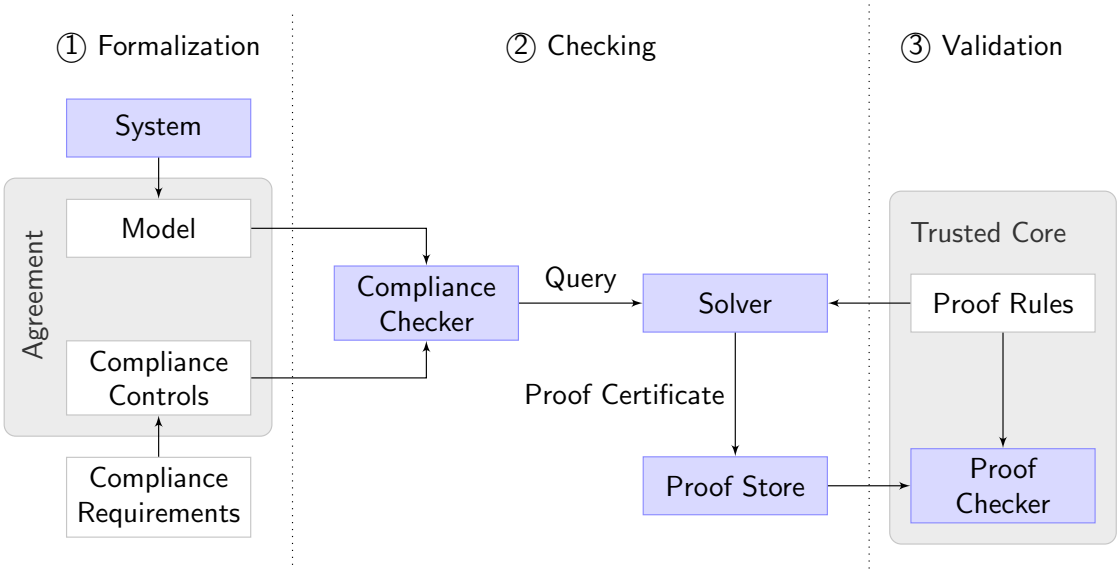
$H = 5\,nm \qquad V = 1000\,ft \qquad 0 \leq t \leq \frac{1}{20}h$

$$|T_1^z(t) - T_2^z(t)| \leq V$$

$$(T_1^x(t) - T_2^x(t))^2 + (T_1^y(t) - T_2^y(t))^2 \leq H^2$$

Automated Compliance

(Barbosa et al. 2023)



Producing and checking proof certificates

SMT solvers can be hard to trust

- ▷ Code bases are large and complex (300K LOC in CVC5)
- ▷ Despite the best effort of developers, bugs remain
- ▷ Every year SMT-COMP has numerous disagreements between solvers
- ▷ Fuzzing tools often find bugs in solvers

Why don't we just certify/qualify the solvers?

- Large, complex code bases are too costly to certify
- A (simpler) certified system can be too slow (Fleury 2019; Fleury et al. 2018)
- Certifying/qualifying a system freezes it, hindering improvements
  - Working around adding new features is slow and costly (Burdy and Déharbe 2018)

A viable alternative: certifying solvers

- Produce a *proof certificate* for every proof
- A proof certificate can be checked independently of the solver
  - Using a small trusted checker
  - And (if done properly) fast (relative to solving time)
- Confidence in results is decoupled from the solver's implementation

A viable alternative: certifying solvers

- Produce a *proof certificate* for every proof
- A proof certificate can be checked independently of the solver
  - Using a small trusted checker
  - And (if done properly) fast (relative to solving time)
- Confidence in results is decoupled from the solver's implementation

So why isn't proof production commonplace in SMT?

Challenges for SMT proofs

- Collecting and storing proofs efficiently
  many attempts, no panacea (Bouton et al. 2009; Hadarean et al. 2015; Katz et al. 2016; Kovács and Voronkov 2013; Moskal 2008; Moura and Bjørner 2008a; Schulz 2013; Sutcliffe et al. 2004; Weidenbach et al. 2009)

- Proofs for sophisticated preprocessing and rewriting techniques
  substantial initial progress but many challenges remain (Barbosa et al. 2020; Nötzli et al. 2022)

- Proofs for complex theory solvers (e.g., CAD, regular expressions)
  open problem

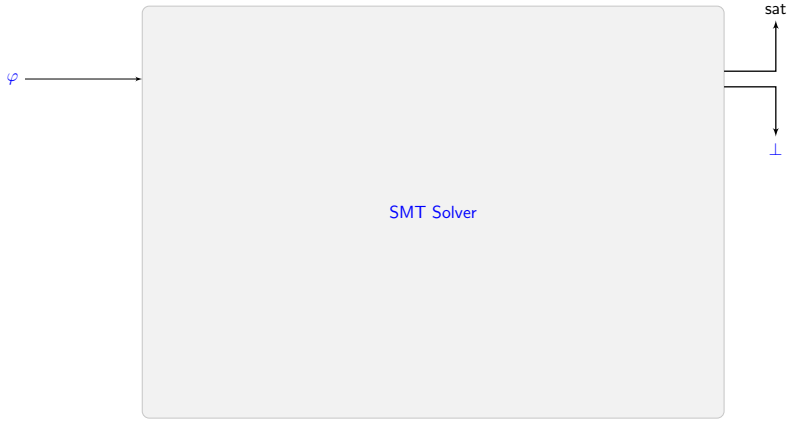
- Standardizing a proof format
  a couple of attempts, not much success

- Scalable, trustworthy checking
  many attempts, no panacea (Barbosa et al. 2020; Blanchette et al. 2013; Ekici et al. 2017; Schurr et al. 2021; Stump et al. 2013)

▷ Our goals:

- Minimize the impact of proof production on the solver's behavior and performance
  - Incorporate (almost) all relevant optimizations
  - Achieve an acceptable performance overhead
- An internal proof checker, part of the cvc5 code base, for every proof rule
- Modular infrastructure allowing fine-grained error localization
- Allow custom eager/lazy generation of proofs
- Support different proof formats (and different external proof checkers)

Proof module architecture for CDCL($\mathcal{T}$)

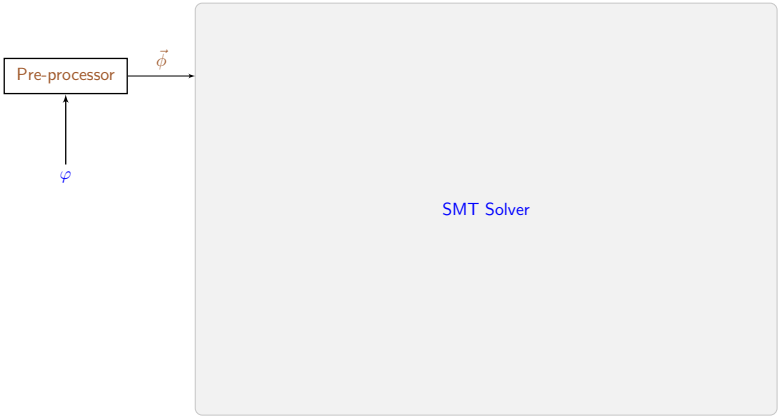


Proof module architecture for CDCL($\mathcal{T}$)

Proof module architecture for CDCL($\mathcal{T}$)

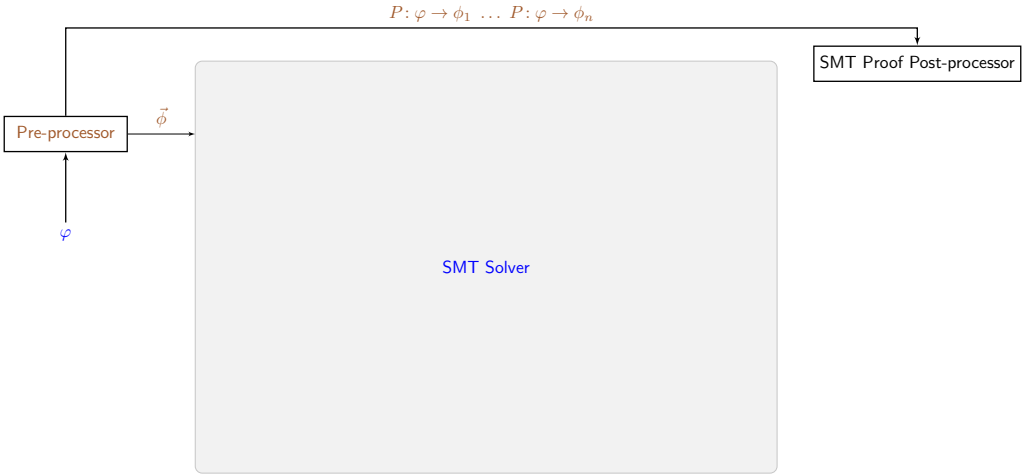
▷

Proof module architecture for CDCL($\mathcal{T}$)



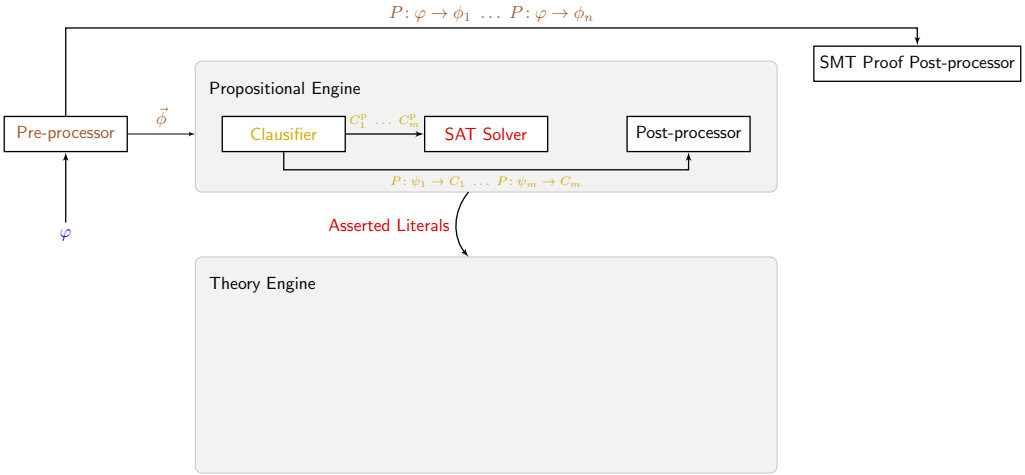
▷ Preprocessor simplifies formula globally: $x \simeq t \wedge F[x] \;\mapsto\; F[t]$ $\quad F[(ite\ P\ t_1\ t_2)] \;\mapsto\; F[t'] \wedge P \to t' \simeq t_1 \wedge \neg P \to t' \simeq t_2$

Proof module architecture for CDCL($\mathcal{T}$)



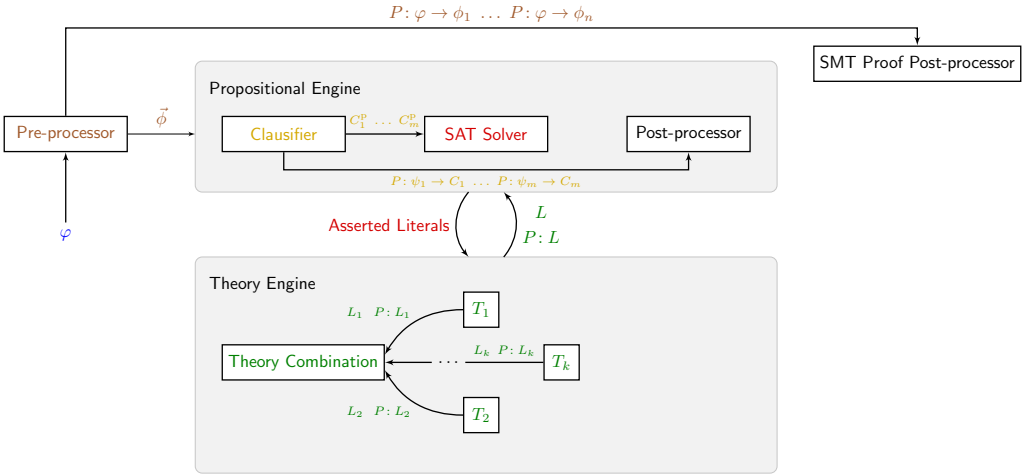
▷ Preprocessor simplifies formula globally: $x \simeq t \wedge F[x] \ \mapsto \ F[t] \qquad F[(ite\ P\ t_1\ t_2)] \ \mapsto \ F[t'] \wedge P \to t' \simeq t_1 \wedge \neg P \to t' \simeq t_2$

Proof module architecture for CDCL($\mathcal{T}$)



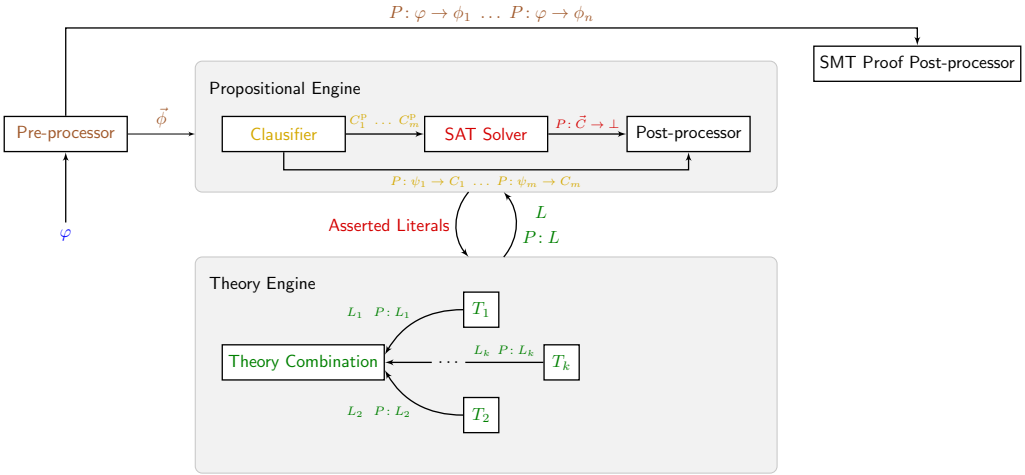
- Clausifier converts to Conjunctive Normal Form (CNF)
  SAT solver asserts literals that must hold based on Boolean abstraction

Proof module architecture for CDCL($\mathcal{T}$)



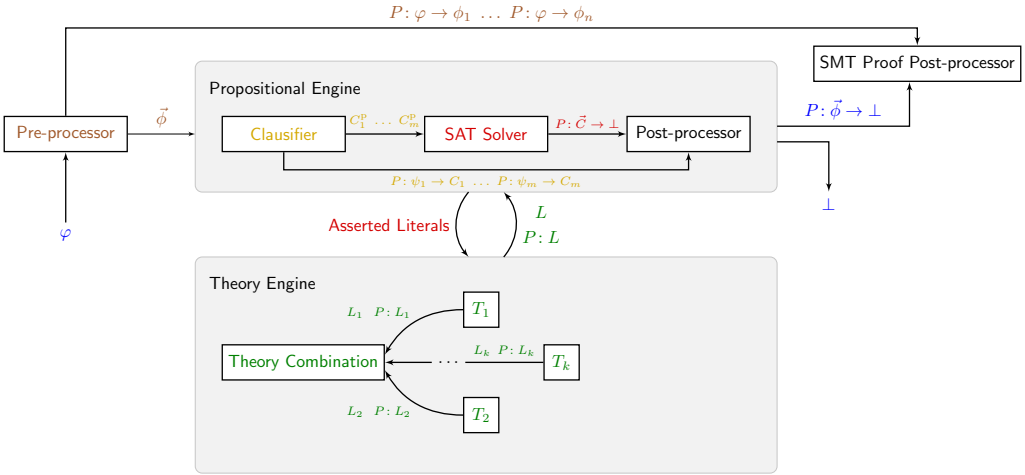
- ▷ Theory solvers check consistency in the theory

Proof module architecture for CDCL($\mathcal{T}$)



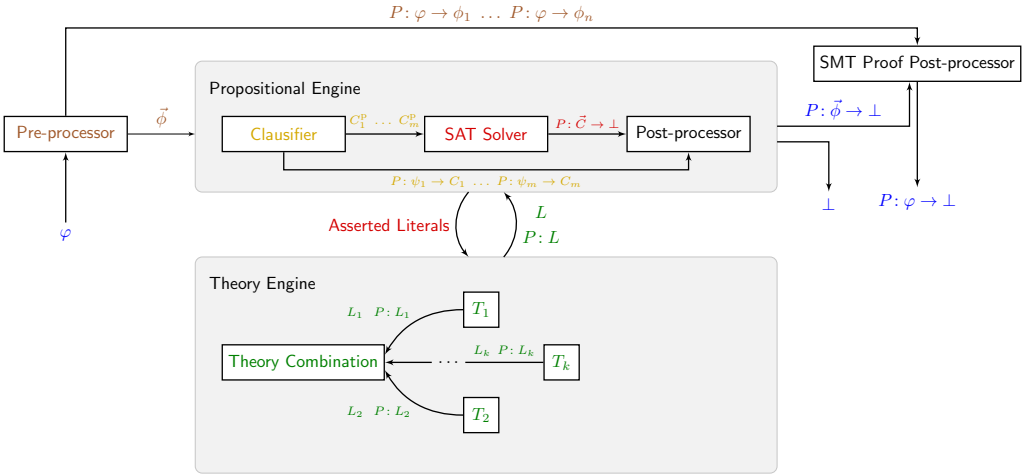
- ▷ Theory solvers check consistency in the theory

Proof module architecture for CDCL($\mathcal{T}$)



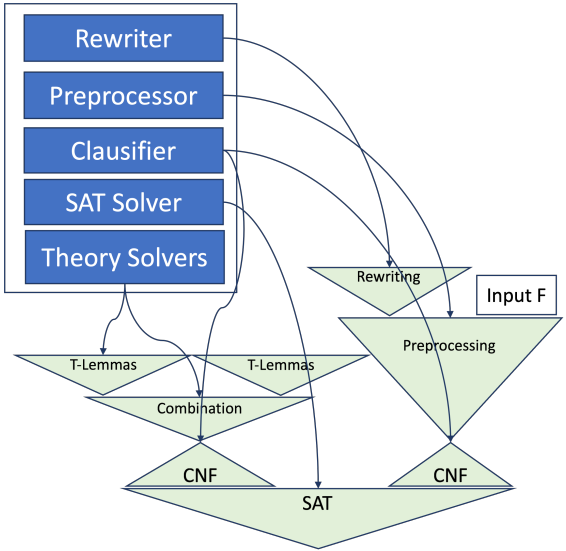
- ▷ Theory solvers check consistency in the theory

Proof module architecture for CDCL($\mathcal{T}$)



- ▷ Theory solvers check consistency in the theory

Resulting proofs



- Preprocessing
- Clausification
- Propositional reasoning
- Theory reasoning
  (UF, LIRA, Strings, . . .)
  and
  quantifier instantiation
- Theory combination
- Rewriting

Proof certificates in various formats

Consider the following unsatisfiable SMT problem in SMT-LIB format

```
(set-logic QF_UF)

(declare-sort U 0)
(declare-const p1 Bool) (declare-const p2 Bool) (declare-const p3 Bool)
(declare-const a U)     (declare-const b U)
(declare-fun f (U) U)

(assert (= a b))
(assert (and p1 true))
(assert (or (not p1) (and p2 p3)))
(assert (or (not p3) (not (= (f a) (f b)))))

(check-sat)
```

Ongoing work

- Conversions to different proof formats
  - Alethe
    - proof reconstruction in Isabelle/HOL via Sledgehammer
    - proof reconstruction in Coq via SMTCoq
    - proof checking in Carcara, a custom checker
  - CPC
    - proof checking with Ethos, a checker parameterized by a specification of CPC in Eunoia
    - proof reconstruction in Lean 4 via cvc5's proof API
  - Dot
    - proof visualization

Conclusion

- Fine-grained proofs and comprehensive proofs are now available for SMT problems
  - Proofs for the strings solver in CVC5 has been a special milestone
- CVC5 has now a proof API and support for multiple proof formats
- We have designed a new and improved proof framework for SMT and built a generic checker for it
- Integration of CVC5 into multiple interactive theorem provers is ongoing
  - Including the formalization of CVC5's proof system in Eunoia, Lean, and Isabelle/HOL
- We expect the high-quality proofs produced by CVC5 to enable many future research directions and applications

Thanks!

SMT Solving for Verification

(or rather an overview of SMT and its applications)

Haniel Barbosa

UFMG

CVC5

ATVA 2024

Oct 21, 2024, Kyoto

References


Abdulla, ParoshAziz et al. (2015). "Norn: An SMT Solver for String Constraints". English. In: Computer Aided Verification. Ed. by Daniel Kroening and Corina S. Păsăreanu. Vol. 9206. Lecture Notes in Computer Science. Springer International Publishing, pp. 462–469.


Ábrahám, Erika et al. (2021). "Deciding the consistency of non-linear real arithmetic constraints with a conflict driven search using cylindrical algebraic coverings". In: J. Log. Algebraic Methods Program. 119, p. 100633.


Backes, John et al. (2018). "Semantic-based Automated Reasoning for AWS Access Policies using SMT". In: Formal Methods In Computer-Aided Design (FMCAD). Ed. by Nikolaj Bjørner and Arie Gurfinkel. IEEE, pp. 1–9.


Bansal, Kshitij et al. (June 2016). "A New Decision Procedure for Finite Sets and Cardinality Constraints in SMT". In: International Joint Conference on Automated Reasoning (IJCAR). Coimbra, Portugal, to appear.


Barbosa, Haniel et al. (2020). "Scalable Fine-Grained Proofs for Formula Processing". In: Journal of Automated Reasoning 64.3, pp. 485–510.


Barbosa, Haniel et al. (2022a). "cvc5: A Versatile and Industrial-Strength SMT Solver". In: Tools and Algorithms for Construction and Analysis of Systems (TACAS), Part I. Ed. by Dana Fisman and Grigore Rosu. Vol. 13243. Lecture Notes in Computer Science. Springer, pp. 415–442.


Barbosa, Haniel et al. (2022b). "Flexible Proof Production in an Industrial-Strength SMT Solver". In: International Joint Conference on Automated Reasoning (IJCAR). Ed. by Jasmin Blanchette, Laura Kovács, and Dirk Pattinson. Vol. 13385. Lecture Notes in Computer Science. Springer, pp. 15–35.

Barbosa, Haniel et al. (2023). "Generating and Exploiting Automated Reasoning Proof Certificates". In: Commun. ACM 66.10, pp. 86–95.


Barrett, Clark, Igor Shikanian, and Cesare Tinelli (2007). "An Abstract Decision Procedure for a Theory of Inductive Data Types". In: JSAT 3.1-2, pp. 21–46.

References


Barrett, Clark et al. (2009). "Satisfiability Modulo Theories". In: Handbook of Satisfiability. Ed. by Armin Biere et al. Vol. 185. Frontiers in Artificial Intelligence and Applications. IOS Press. Chap. 26, pp. 825–885.


Barrett, Clark et al. (2011). "CVC4". In: Computer Aided Verification (CAV). Ed. by Ganesh Gopalakrishnan and Shaz Qadeer. Springer, pp. 171–177.


Barrett, Clark W. and Sergey Berezin (2004). "CVC Lite: A New Implementation of the Cooperating Validity Checker Category B". In: Computer Aided Verification (CAV). Ed. by Rajeev Alur and Doron A. Peled. Vol. 3114. Lecture Notes in Computer Science. Springer, pp. 515–518.


Barrett, Clark W. and Cesare Tinelli (2007). "CVC3". In: Computer Aided Verification (CAV). Ed. by Werner Damm and Holger Hermanns. Vol. 4590. Lecture Notes in Computer Science. Springer, pp. 298–302.


— (2018). "Satisfiability Modulo Theories". In: Handbook of Model Checking. Ed. by Edmund M. Clarke et al. Springer, pp. 305–343.


Bjørner, Nikolaj S. and Lev Nachmanson (2024). "Arithmetic Solving in Z3". In: Computer Aided Verification (CAV), Part I. Ed. by Arie Gurfinkel and Vijay Ganesh. Vol. 14681. Lecture Notes in Computer Science. Springer, pp. 26–41.


Blanchette, Jasmin Christian, Sascha Böhme, and Lawrence C. Paulson (2013). "Extending Sledgehammer with SMT Solvers". In: Journal of Automated Reasoning 51.1, pp. 109–128.


Bofill, M. et al. (2008). "A Write-Based Solver for SAT Modulo the Theory of Arrays". In: Formal Methods in Computer-Aided Design, FMCAD, pp. 1–8.


Borralleras, C. et al. (2009). "Solving Non-linear Polynomial Arithmetic via SAT Modulo Linear Arithmetic". In: 22nd International Conference on Automated Deduction , CADE-22. Ed. by R. A. Schmidt. Vol. 5663. Lecture Notes in Computer Science. Springer, pp. 294–305.

References


Bouton, Thomas et al. (2009). "veriT: An Open, Trustable and Efficient SMT-Solver". In: Proc. Conference on Automated Deduction (CADE). Ed. by Renate A. Schmidt. Vol. 5663. Lecture Notes in Computer Science. Springer, pp. 151–156.


Bozzano, Marco et al. (2005a). "An Incremental and Layered Procedure for the Satisfiability of Linear Arithmetic Logic". English. In: Tools and Algorithms for the Construction and Analysis of Systems. Ed. by Nicolas Halbwachs and LenoreD. Zuck. Vol. 3440. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 317–333.


Bozzano, Marco et al. (2005b). "The MathSAT 3 System". In: Proc. Conference on Automated Deduction (CADE). Ed. by Robert Nieuwenhuis. Vol. 3632. Lecture Notes in Computer Science. Springer, pp. 315–321.


Bradley, Aaron R. and Zohar Manna (2007). The calculus of computation - decision procedures with applications to verification. Springer.


Brain, Martin, Florian Schanda, and Youcheng Sun (2019). "Building Better Bit-Blasting for Floating-Point Problems". In: Tools and Algorithms for Construction and Analysis of Systems (TACAS), Part I. Ed. by Tomás Vojnar and Lijun Zhang. Vol. 11427. Lecture Notes in Computer Science. Springer, pp. 79–98.


Brain, Martin et al. (2014). "Deciding floating-point logic with abstract conflict driven clause learning". In: Formal Methods Syst. Des. 45.2, pp. 213–245.


Brummayer, Robert and Armin Biere (2009). "Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays". In: Tools and Algorithms for the Construction and Analysis of Systems: 15th International Conference, TACAS 2009, Held as Part of the Joint Europe Ed. by Stefan Kowalewski and Anna Philippou. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 174–177.


Bruttomesso, Roberto et al. (2008). "The MathSAT 4SMT Solver". In: Computer Aided Verification (CAV). Ed. by Aarti Gupta and Sharad Malik. Vol. 5123. Lecture Notes in Computer Science. Springer, pp. 299–303.

References


Burdy, Lilian and David Déharbe (2018). "Teaching an Old Dog New Tricks - The Drudges of the Interactive Prover in Atelier B". In: Abstract State Machines, Alloy, B, TLA, VDM, and Z - 6th International Conference, ABZ 2018, Southampton, UK, June 5-8, 2018, Proceedings. Ed. by Michael J. Butler et al. Vol. 10817. Lecture Notes in Computer Science. Springer, pp. 415–419.


Cimatti, Alessandro et al. (2013). "The MathSAT5 SMT Solver". In: Tools and Algorithms for Construction and Analysis of Systems (TACAS). Ed. by Nir Piterman and Scott A. Smolka. Vol. 7795. Lecture Notes in Computer Science. Springer, pp. 93–107.


Cimatti, Alessandro et al. (2017). "Satisfiability Modulo Transcendental Functions via Incremental Linearization". In: Proc. Conference on Automated Deduction (CADE). Ed. by Leonardo de Moura. Vol. 10395. Lecture Notes in Computer Science. Springer, pp. 95–113.


Conchon, Sylvain et al. (2017). "A Three-Tier Strategy for Reasoning About Floating-Point Numbers in SMT". In: Computer Aided Verification (CAV), Part II. Ed. by Rupak Majumdar and Viktor Kuncak. Vol. 10427. Lecture Notes in Computer Science. Springer, pp. 419–435.


Cotton, S. and O. Maler (2006). "Fast and Flexible Difference Constraint Propagation for DPLL(T)". In: 9th International Conference on Theory and Applications of Satisfiability Testing, SAT'06. Ed. by A. Biere and C. P. Gomes. Vol. 4121. Lecture Notes in Computer Science. Springer, pp. 170–183.


De Moura, Leonardo and Nikolaj Bjørner (2011). "Satisfiability modulo theories: introduction and applications". In: Communications of the ACM 54.9, pp. 69–77.


Dutertre, Bruno (2014). "Yices 2.2". English. In: Computer Aided Verification (CAV). Ed. by Armin Biere and Roderick Bloem. Vol. 8559. Lecture Notes in Computer Science. Springer International Publishing, pp. 737–744.

References


Dutertre, Bruno and Leonardo de Moura (2006). "A Fast Linear-Arithmetic Solver for DPLL(T)". English. In: Computer Aided Verification (CAV). Ed. by Thomas Ball and Robert B. Jones. Vol. 4144. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 81–94.


Ekici, Burak et al. (2017). "SMTCoq: A Plug-In for Integrating SMT Solvers into Coq". In: Computer Aided Verification (CAV). Ed. by Rupak Majumdar and Viktor Kuncak. Vol. 10427. Lecture Notes in Computer Science. Springer, pp. 126–133.


Fleury, Mathias (2019). "Optimizing a Verified SAT Solver". In: NASA Formal Methods - 11th International Symposium, NFM 2019, Houston, TX, USA, May 7-9, 2019, Proceedings. Ed. by Julia M. Badger and Kristin Yvonne Rozier. Vol. 11460. Lecture Notes in Computer Science. Springer, pp. 148–165.


Fleury, Mathias, Jasmin Christian Blanchette, and Peter Lammich (2018). "A verified SAT solver with watched literals using imperative HOL". In: Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2018, Los Angeles, CA, USA, January 8-9 Ed. by June Andronick and Amy P. Felty. ACM, pp. 158–171.


Gao, Sicun, Soonho Kong, and Edmund M Clarke (2013). "Satisfiability modulo ODEs". In: Formal Methods in Computer-Aided Design (FMCAD), 2013. IEEE, pp. 105–112.


Hadarean, Liana et al. (2015). "Fine Grained SMT Proofs for the Theory of Fixed-Width Bit-Vectors". In: Logic for Programming, Artificial Intelligence, and Reasoning (LPAR). Ed. by Martin Davis et al. Vol. 9450. Lecture Notes in Computer Science. Springer, pp. 340–355.


Hader, Thomas, Daniela Kaufmann, and Laura Kovács (2023). "SMT Solving over Finite Field Arithmetic". In: Logic for Programming, Artificial Intelligence, and Reasoning (LPAR). Ed. by Ruzica Piskac and Andrei Voronkov. Vol. 94. EPiC Series in Computing. EasyChair, pp. 238–256.

References


Jovanović, Dejan and Leonardo de Moura (2012). "Solving Non-linear Arithmetic". English. In: International Joint Conference on Automated Reasoning (IJCAR). Ed. by Bernhard Gramlich, Dale Miller, and Uli Sattler. Vol. 7364. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 339–354.


Katz, Guy et al. (2016). "Lazy proofs for DPLL(T)-based SMT solvers". In: Formal Methods In Computer-Aided Design (FMCAD). Ed. by Ruzica Piskac and Muralidhar Talupur. IEEE, pp. 93–100.


Kiezun, Adam et al. (2009). "HAMPI: a solver for string constraints". In: Proceedings of the eighteenth international symposium on Software testing and analysis. ACM, pp. 105–116.


Kovács, Laura and Andrei Voronkov (2013). "First-Order Theorem Proving and Vampire". English. In: Computer Aided Verification (CAV). Ed. by Natasha Sharygina and Helmut Veith. Vol. 8044. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 1–35.


Kroening, Daniel and Ofer Strichman (2008). Decision Procedures - An Algorithmic Point of View. Texts in Theoretical Computer Science. An EATCS Series. Springer.


Lahiri, Shuvendu K. and Madanlal Musuvathi (2005). "An Efficient Decision Procedure for UTVPI Constraints". In: 5th International Workshop on Frontiers of Combining Systems, FroCos'05. Ed. by B. Gramlich. Vol. 3717. Lecture Notes in Computer Science. Springer, pp. 168–183.


Liang, Tianyi et al. (2014). "A DPLL(T) Theory Solver for a Theory of Strings and Regular Expressions". In: Computer Aided Verification (CAV). Ed. by Armin Biere and Roderick Bloem. Vol. 8559. Lecture Notes in Computer Science. Springer, pp. 646–662.

McCarthy, John (1993). "Towards a mathematical science of computation". In: Program Verification. Springer, pp. 35–56.

References


Meng, Baoluo et al. (2017). "Relational Constraint Solving in SMT". In: Proceedings of the 26th International Conference on Automated Deduction. Ed. by Leonardo de Moura. Vol. 10395. Lecture Notes in Computer Science. Springer, pp. 148–165.


Moskal, Michał (2008). "Rocket-Fast Proof Checking for SMT Solvers". In: Tools and Algorithms for Construction and Analysis of Systems (TACAS). Ed. by C. R. Ramakrishnan and Jakob Rehof. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 486–500.


Moura, Leonardo Mendonça de and Nikolaj Bjørner (2008a). "Proofs and Refutations, and Z3". In: Logic for Programming, Artificial Intelligence, and Reasoning (LPAR) Workshops. Ed. by Piotr Rudnicki et al. Vol. 418. CEUR Workshop Proceedings. CEUR-WS.org.


— (2008b). "Z3: An Efficient SMT Solver". In: Tools and Algorithms for Construction and Analysis of Systems (TACAS). Ed. by C. R. Ramakrishnan and Jakob Rehof. Vol. 4963. Lecture Notes in Computer Science. Springer, pp. 337–340.


— (2009). "Generalized, efficient array decision procedures". In: Formal Methods In Computer-Aided Design (FMCAD). IEEE, pp. 45–52.


Moura, Leonardo Mendonça de and Nikolaj S. Bjørner (2010). "Bugs, Moles and Skeletons: Symbolic Reasoning for Software Development". In: International Joint Conference on Automated Reasoning (IJCAR). Ed. by Jürgen Giesl and Reiner Hähnle. Vol. 6173. Lecture Notes in Computer Science. Springer, pp. 400–411.


Narkawicz, Anthony and César A Munoz (2012). "Formal Verification of Conflict Detection Algorithms for Arbitrary Trajectories.". In: Reliable Computing 17.2, pp. 209–237.


Nelson, Greg and Derek C. Oppen (1980). "Fast Decision Procedures Based on Congruence Closure". In: J. ACM 27.2, pp. 356–364.

References


Niemetz, Aina and Mathias Preiner (2023). "Bitwuzla". In: Computer Aided Verification (CAV), Part II. Ed. by Constantin Enea and Akash Lal. Vol. 13965. Lecture Notes in Computer Science. Springer, pp. 3–17.


Nieuwenhuis, Robert and Albert Oliveras (July 2005). "DPLL(T) with Exhaustive Theory Propagation and its Application to Difference Logic". In: Proceedings of the 17th International Conference on Computer Aided Verification, CAV'05 (Edimburgh, Scotland). Ed. by Kousha Etessami and Sriram K. Rajamani. Vol. 3576. Lecture Notes in Computer Science. Springer, pp. 321–334.


— (2007). "Fast congruence closure and extensions". In: Information and Computation 205.4. Special Issue: 16th International Conference on Rewriting Techniques and Applications, pp. 557 –580.


Nötzli, Andres et al. (Oct. 2022). "Reconstructing Fine-Grained Proofs of Complex Rewrites Using a Domain-Specific Language". In: Proceedings of the 22^{nd} International Conference on Formal Methods In Computer-Aided Design (FMCAD '22). Ed. by Alberto Griggio and Neha Rungta. TU Wien Academic Press.


Ozdemir, Alex et al. (2023). "Satisfiability Modulo Finite Fields". In: Computer Aided Verification (CAV), Part II. Ed. by Constantin Enea and Akash Lal. Vol. 13965. Lecture Notes in Computer Science. Springer, pp. 163–186.


Reynolds, Andrew and Jasmin Christian Blanchette (2017). "A Decision Procedure for (Co)datatypes in SMT Solvers". In: J. Autom. Reasoning 58.3, pp. 341–362.


Reynolds, Andrew et al. (2017). "Refutation-based synthesis in SMT". In: Formal Methods in System Design.


Reynolds, Andrew et al. (2018). "Datatypes with Shared Selectors". In: International Joint Conference on Automated Reasoning (IJCAR). Ed. by Didier Galmiche, Stephan Schulz, and Roberto Sebastiani. Vol. 10900. Lecture Notes in Computer Science. Springer, pp. 591–608.


Rungta, Neha (2022). "A Billion SMT Queries a Day (Invited Paper)". In: Computer Aided Verification (CAV), Part I. Ed. by Sharon Shoham and Yakir Vizel. Vol. 13371. Lecture Notes in Computer Science. Springer, pp. 3–18.

References


Schulz, Stephan (2013). "System Description: E 1.8". English. In: Logic for Programming, Artificial Intelligence, and Reasoning (LPAR). Ed. by Ken McMillan, Aart Middeldorp, and Andrei Voronkov. Vol. 8312. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 735–743.


Schurr, Hans-Jörg, Mathias Fleury, and Martin Desharnais (2021). "Reliable Reconstruction of Fine-grained Proofs in a Proof Assistant". In: Proc. Conference on Automated Deduction (CADE). Ed. by André Platzer and Geoff Sutcliffe. Vol. 12699. Lecture Notes in Computer Science. Springer, pp. 450–467.


Solar-Lezama, Armando et al. (2006). "Combinatorial sketching for finite programs". In: Architectural Support for Programming Languages and Operating Systems (ASPLOS). Ed. by John Paul Shen and Margaret Martonosi. ACM, pp. 404–415.


Stump, Aaron, Clark W. Barrett, and David L. Dill (2002). "CVC: A Cooperating Validity Checker". In: Computer Aided Verification (CAV). Ed. by Ed Brinksma and Kim Guldstrand Larsen. Vol. 2404. Lecture Notes in Computer Science. Springer, pp. 500–504.


Stump, Aaron et al. (2001). "A Decision Procedure for an Extensional Theory of Arrays". In: Logic In Computer Science (LICS). IEEE Computer Society, pp. 29–37.


Stump, Aaron et al. (2013). "SMT proof checking using a logical framework". In: Formal Methods in System Design 42.1, pp. 91–118.


Sutcliffe, Geoff, Jürgen Zimmer, and Stephan Schulz (2004). "TSTP Data-Exchange Formats for Automated Theorem Proving Tools". In: Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems. Ed. by Weixiong Zhang and Volker Sorge. Vol. 112. Frontiers in Artificial Intelligence and Applications. IOS Press, pp. 201–215.


Udupa, Abhishek et al. (2013). "TRANSIT: specifying protocols with concolic snippets". In: Conference on Programming Language Design and Implementation (PLDI). Ed. by Hans-Juergen Boehm and Cormac Flanagan. ACM, pp. 287–296.

References


Wang, C. et al. (2005). "Deciding Separation Logic Formulae by SAT and Incremental Negative Cycle Elimination". In: 12h International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR'05. Ed. by G. Sutcliffe and A. Voronkov. Vol. 3835. Lecture Notes in Computer Science. Springer, pp. 322–336.


Weidenbach, Christoph et al. (2009). "SPASS Version 3.5". English. In: Proc. Conference on Automated Deduction (CADE). Ed. by RenateA. Schmidt. Vol. 5663. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 140–145.


Zankl, Harald and Aart Middeldorp (2010). "Satisfiability of Non-linear (Ir)rational Arithmetic". In: 16th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR'10. Ed. by Edmund M. Clarke and Andrei Voronkov. Vol. 6355. Lecture Notes in Computer Science. Springer, pp. 481–500.