SMT solving for fun and profit

Haniel Barbosa

UFMG

BIRS Workshop 26w5626
Theory and Practice of SAT and Combinatorial Solving

Jan 13, 2026

Acknowledgments

Many thanks to Abdalrhman M Mohamed, Cesare Tinelli, Andres Nötzli, Andrew Reynolds, Clark Barrett, Alberto Griggio, Liana Hadarean, Dejan Jovanovic, and Albert Oliveras for contributing, directly or transitively, 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

Introduction

Automated Reasoning for Formal Methods

Two successful examples:

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

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

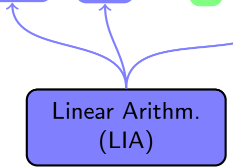
$$n > 3 * m + 1 \;\wedge\; (f(n) \leq head(l_1) \;\vee\; l_2 = f(n) :: l_1)$$

The Basic SMT Problem

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

Example

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

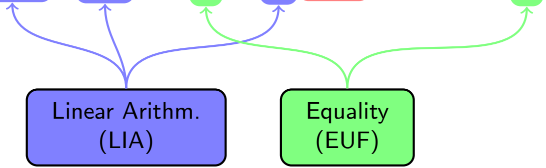


The Basic SMT Problem

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

Example

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

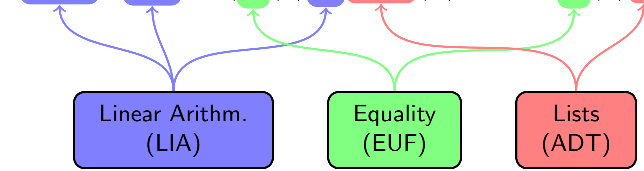


The Basic SMT Problem

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

Example

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

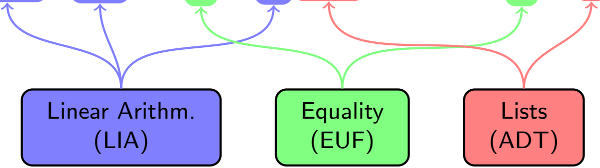


The Basic SMT Problem

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

Example

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



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

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

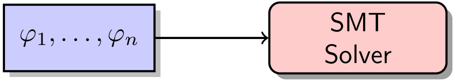
Billions SMT queries a day via Zelkova[a]

[a] Backes et al. 2018; Rungta 2022

SMT solver functionality

SMT Solver Basic Functionality

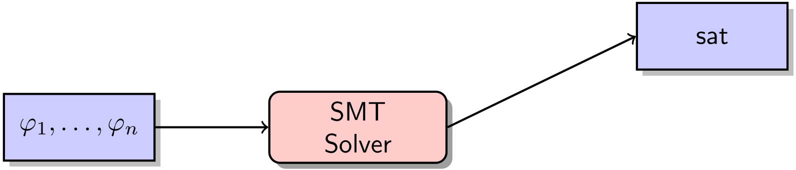
Background theory T



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	$4 \cdot (x \gg 2) = x \mathbin{\&} {\sim}3$
Strings and RegExs	$x = y \cdot z \wedge z \in ab^* \Rightarrow \lvert x \rvert > \lvert y \rvert$
Arrays	$i = j \Rightarrow \mathsf{store}(a, i, x)[j] = x$

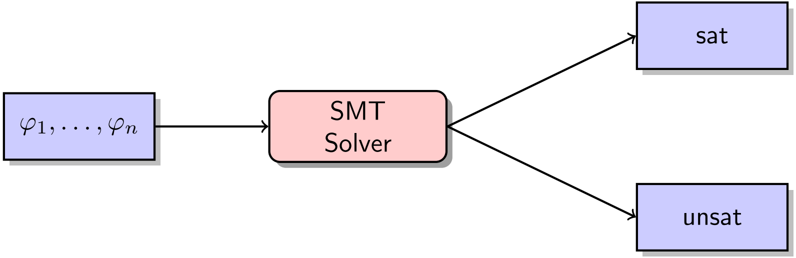
SMT Solver Basic Functionality

Background theory T



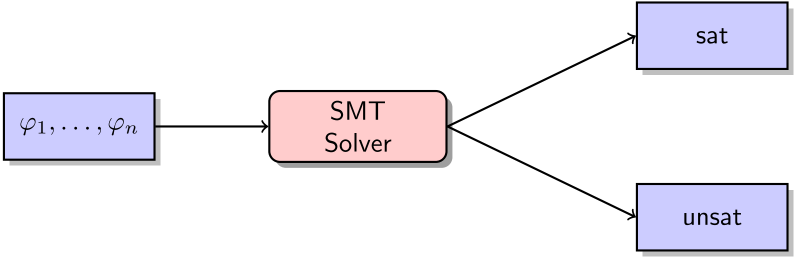
SMT Solver Basic Functionality

Background theory T



SMT Solver Basic Functionality

Background theory T

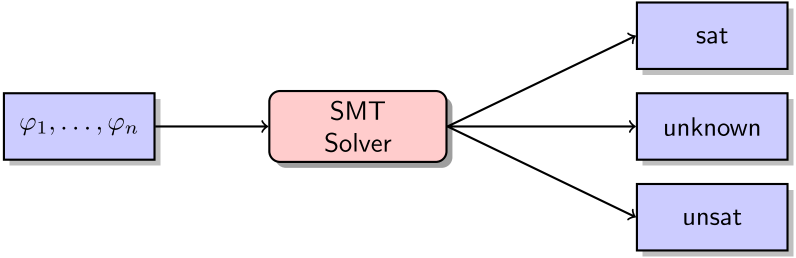


sat/unsat: there is a/no model M of T such that

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

SMT Solver Basic Functionality

Background theory T



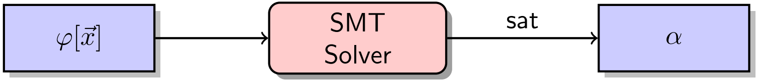
sat/unsat: there is a/no model M of T such that

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

unknown: inconclusive — because of resource limits or incompleteness

SMT Solver Output: Satisfying Assignments

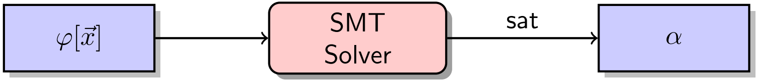
Background theory T



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

SMT Solver Output: Satisfying Assignments

Background theory T

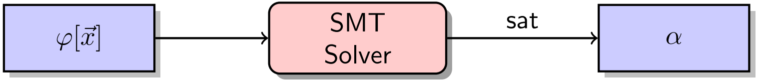


α is a satisfying assignment for $\vec{x} = (x_1, \ldots, x_n)$:

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 $\vec{x} = (x_1, \ldots, x_n)$:

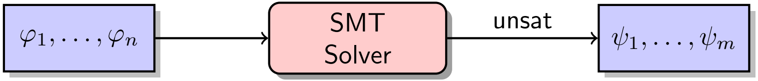
1. $\alpha = \{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\}$ for some values $\vec{v} = (v_1, \ldots, v_n)$
2. $M \models \varphi[\vec{x} \mapsto \vec{v}]$ for some model M of T

Note.

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

SMT Solver Output: Unsat Cores

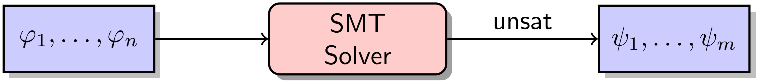
Background theory T



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

SMT Solver Output: Unsat Cores

Background theory T

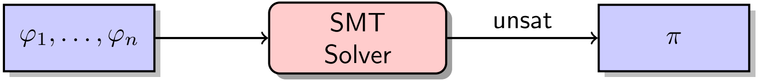


$\psi_1, \ldots, \psi_m$ is a unsat core of $\{\varphi_1, \ldots, \varphi_n\}$:

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

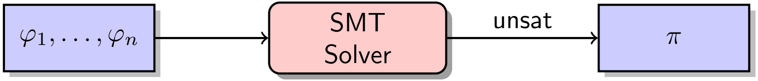
Background theory T



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

SMT Solver Output: Proofs

Background theory T

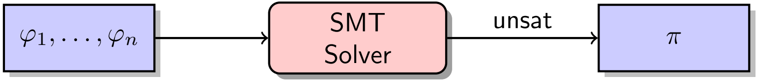


π is a checkable proof object for $\{\varphi_1, \ldots, \varphi_n\}$:

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

SMT Solver Output: Proofs

Background theory T

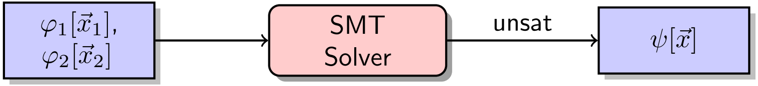


π is a checkable proof object for $\{\varphi_1, \ldots, \varphi_n\}$:

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
   - The "efficiently" there is actually a highly debatable point...

Extended Functionality: Interpolation

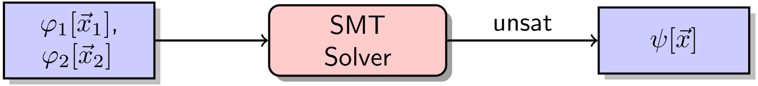
Background theory T



ψ is a logical interpolant of φ_1 and φ_2:

Extended Functionality: Interpolation

Background theory T

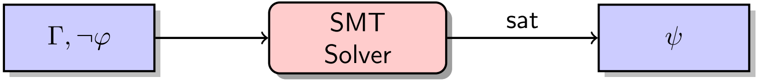


ψ is a logical interpolant of φ_1 and φ_2:

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

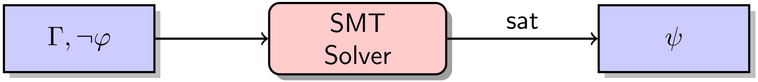
Background theory T



ψ is an abduction hypothesis for φ wrt Γ:

Extended Functionality: Abduction

Background theory T

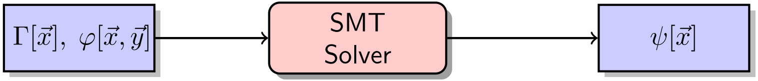


ψ is an abduction hypothesis for φ wrt Γ:

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 $\psi' \models_T \psi$)

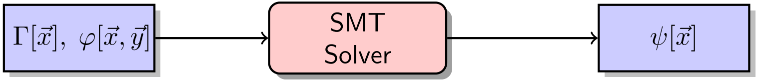
Extended Functionality: Quantifier Elimination

Background theory T



Extended Functionality: Quantifier Elimination

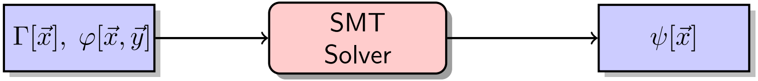
Background theory T



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

Extended Functionality: Quantifier Elimination

Background theory T

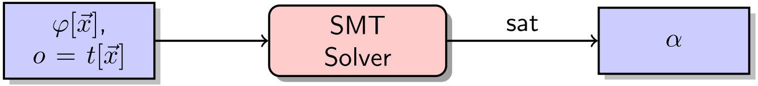


ψ is a projection of φ over $\vec{y}$ with respect to Γ:

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

Extended Functionality: Optimization

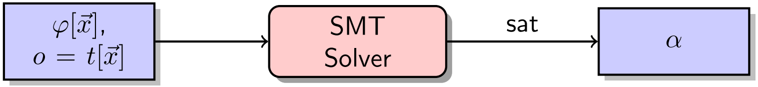
Background theory T



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

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	$x \neq \mathit{Leaf} \Rightarrow \exists\, l, r : \mathit{Tree}(\alpha).\ \exists\, a : \alpha.\ x = \mathit{Node}(l, a, r)$
Finite Sets	$e_1 \in x \wedge e_2 \in x \setminus e_1 \Rightarrow \exists y, z : \mathit{Set}(\alpha).\ \lvert y \rvert = \lvert z \rvert \wedge x = y \cup z \wedge y \neq \emptyset$
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: $\forall x, y : \sigma.\ x = y \land y = z \Rightarrow x = z$
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. 2005)
- 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

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

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

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

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

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

(**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 "BIRS"?

Other Interesting Theories

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

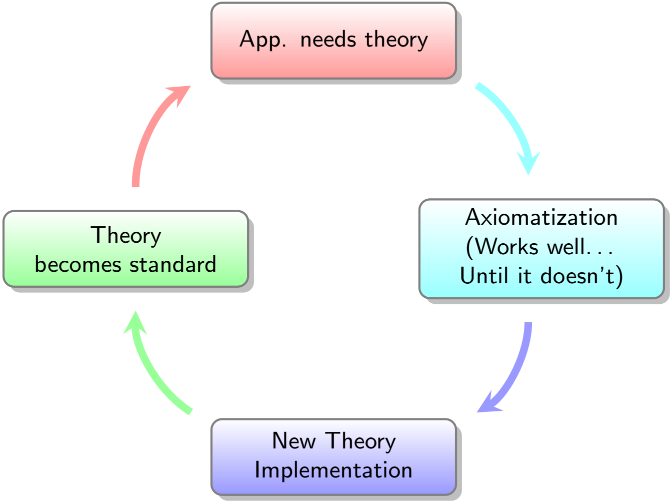
Some SMT solvers also allow you to axiomatize your own theory

- 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, veriT, 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



Application example: Software Verification

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*)

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

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

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

${}_{32}\ h_0[b])$

${}_{32}\ h_1[b])$

$h_3 = \text{store}(h_2, a, h_2[a] -_{32} h_2[b])$

$\neg(h_3[a] = h_0[b] \wedge h_3[b] = h_0[a])$

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

Example (Binary Search)

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

```
//@as
//
//@e
//
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;
}
```

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

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

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

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

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

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

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

$$inv = 0 \leq l \land l \leq h \land h \leq n \land n \leq |a| \land$$
$$\forall i : \mathsf{Int}\ \ 0 \leq i \land i \leq l - 1 \Rightarrow a[i] < k \land$$
$$\forall i : \mathsf{Int}\ \ h \leq i \land i$$

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

SMT solver answer

Unsatisfiable

What's next? What's hot?

Stability

- Seemingly irrelevant changes (e.g. variable renaming, assertion order) can have unpredictable performance impact.
- For the longest time SMT developers could get away with "these are NP-complete or undecidable problems, we must use heuristics, so this is to be expected."
- Users have been pushing back more and more so we had to accept we need to deal with this.
- Initial explorations:
  - Normalization of input (Amrollahi et al. 2025)
  - Identifying missing instances across mutated input (Zhou et al. 2025)
  - Better engineering (Cebeci et al. 2025)

Improving hard theories

- Bitvectors (BV) and floating-point arithmetic (FP)
  - Leveraging numerical methods in FP (Zhang et al. 2026)
  - Algebraic methods for BV (Kaufmann and Biere 2021; Rath et al. 2024)
  - CEGAR-based approaches for mitigating bit-blasting bottlenecks (Niemetz et al. 2024)
- Strings
  - Automata-based approaches in Z3Noodler (Chocholatý et al. 2025) and OSTRICH (Hague et al. 2025)
  - Symbolic derivatives in Z3 (Varatalu et al. 2025)
- Non-linear arithmetic
  - Cylindrical algebraic decomposition (CAD) based methods in SMT-RAT (Ábrahám et al. 2021; Promies et al. 2025)
  - MCSat-based methods in Yices (Lipparini et al. 2025)
  - CAD and incremental linearization methods in cvc5 (Cimatti et al. 2017a; Kremer et al. 2022)

Proofs

- Numerous applications (correctness, integration with other systems, interpretability)
  - Improving automation in Lean (Mohamed et al. 2025) and Isabelle (Lachnitt et al. 2025)
  - Huge investment from AWS aiming to automate compliance via cvc5 proofs (Barbosa et al. 2023)
  - cvc5 proofs may be going into the Linux kernel (Sun and Su 2025)

- Numerous challenges
  - Enormous effort to instrument solvers to produce *detailed* proofs (Barbosa et al. 2022)
  - No standard format or proof calculus, leading to lots of duplicated work (Hoenicke and Schindler 2022; Moura and Bjørner 2008; Schurr et al. 2021)
  - Proofs for complex theory solvers (e.g. CAD-based, automata-based) and rewriters (e.g. strings) are not as mature
  - Huge proofs that are costly to produce and costly to check
    - Lazy proof generation can help (Hitarth et al. 2024)
    - Shorter proofs via different proof calculi (Liew et al. 2020) or algorithms (Andreotti and Barbosa 2026)

Others

- Parallel solving
  - Initial attempts of lifting cube-and-conquer to SMT (Hyvärinen et al. 2021; Wilson et al. 2023)
  - Clause-sharing in portfolio solving (Barrett et al. 2024)
- Using a SAT solver with chronological backtracking
  - ...

Thanks!

SMT solving for fun and profit

Haniel Barbosa

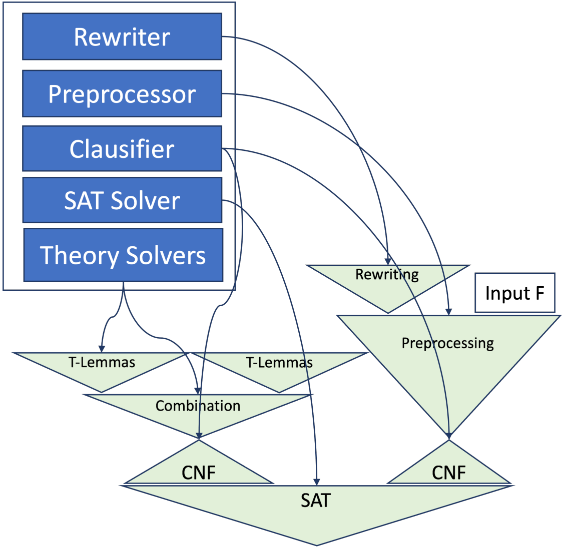
UFMG

BIRS Workshop 26w5626

Theory and Practice of SAT and Combinatorial Solving

Jan 13, 2026

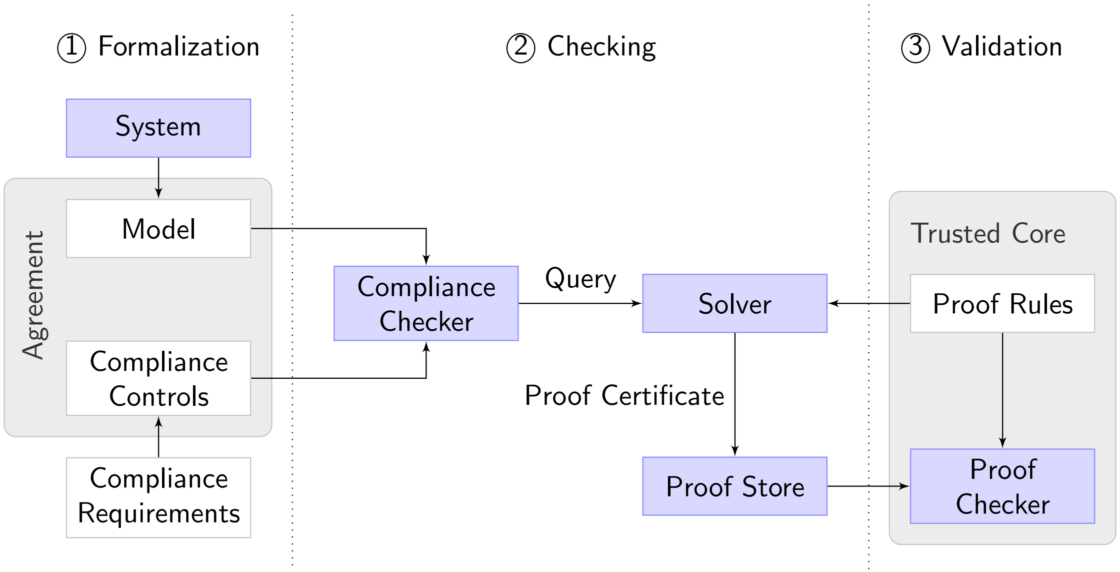
Resulting proofs



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

Automated Compliance

(Barbosa et al. 2023)



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

$\ldots$

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

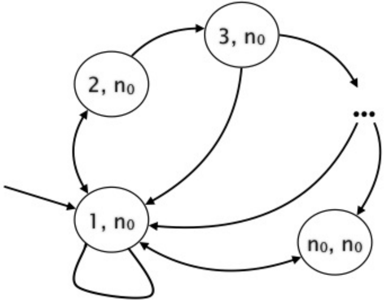
c := 1
n := n_0

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

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

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) $P[\vec{x}] \land R[\vec{x}, \vec{x}'] \models_T P[\vec{x}']$ (inductive step)

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)

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.
- Amrollahi, Daneshvar et al. (2025). “Towards SMT Solver Stability via Input Normalization”. In: Formal Methods In Computer-Aided Design (FMCAD). Ed. by Ahmed Irfan and Daniela Kaufmann. Vienna, Austria: TU Wien Academic Press, pp. 84–93.
- Andreotti, Bruno and Haniel Barbosa (2026). “Producing Shorter Congruence Closure Proofs in a State-of-the-Art SMT Solver”. In: Verification, Model Checking, and Abstract Interpretation (VMCAI). Ed. by Yu-Fang Chen, Thomas Jensen, and Ondřej Lengál. Cham: Springer Nature Switzerland, pp. 1–20.
- 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. (2022). “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.

References

- 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.
- Barrett, Clark W. et al. (2024). "SMT-D: New Strategies for Portfolio-Based SMT Solving". In: Formal Methods In Computer-Aided Design (FMCAD). Ed. by Nina Narodytska and Philipp Rümmer. IEEE, pp. 1–10.
- 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.
- 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.
- Bozzano, Marco et al. (2005). "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.
- 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.

References

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.

Cebeci, Can et al. (Aug. 2025). "A Conjecture Regarding SMT Instability". In: International Workshop on Satisfiability Modulo Theories (SMT). Vol. 4008. CEUR Workshop Proceedings. Edited by Jochen Hoenicke, Sophie Tourret. Glasgow, UK: CEUR-WS.org, pp. 136–147.

Chocholatý, David et al. (2025). "Z3-Noodler 1.3: Shepherding Decision Procedures for Strings with Model Generation". In: Tools and Algorithms for Construction and Analysis of Systems (TACAS), Part II. Ed. by Arie Gurfinkel and Marijn Heule. Vol. 15697. Lecture Notes in Computer Science. Springer, pp. 23–44.

Cimatti, Alessandro et al. (2017a). "Invariant Checking of NRA Transition Systems via Incremental Reduction to LRA with EUF". In: Tools and Algorithms for Construction and Analysis of Systems (TACAS). Ed. by Axel Legay and Tiziana Margaria. Vol. 10205. Lecture Notes in Computer Science, pp. 58–75.

— (2017b). "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.

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.
- 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.
- 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.
- Hague, Matthew et al. (2025). "OSTRICH2: Solver for Complex String Constraints". In: Formal Methods In Computer-Aided Design (FMCAD). Ed. by Ahmed Irfan and Daniela Kaufmann. Vienna, Austria: TU Wien Academic Press, pp. 145–158.
- Hitarth, S. et al. (2024). "Extending DRAT to SMT". In: Formal Methods In Computer-Aided Design (FMCAD). Ed. by Nina Narodytska and Philipp Rümmer. IEEE, pp. 1–11.
- Hoenicke, Jochen and Tanja Schindler (2022). "A Simple Proof Format for SMT". In: International Workshop on Satisfiability Modulo Theories (SMT). Ed. by David Déharbe and Antti E. J. Hyvärinen. Vol. 3185. CEUR Workshop Proceedings. CEUR-WS.org, pp. 54–70.
- Hyvärinen, Antti E. J., Matteo Marescotti, and Natasha Sharygina (2021). "Lookahead in Partitioning SMT". In: Formal Methods In Computer-Aided Design (FMCAD). IEEE, pp. 271–279.

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.

Kaufmann, Daniela and Armin Biere (2021). “AMulet 2.0 for Verifying Multiplier Circuits”. In: Tools and Algorithms for Construction and Analysis of Systems (TACAS), Part II. Ed. by Jan Friso Groote and Kim Guldstrand Larsen. Vol. 12652. Lecture Notes in Computer Science. Springer, pp. 357–364.

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.

Kremer, Gereon et al. (2022). “Cooperating Techniques for Solving Nonlinear Real Arithmetic in the cvc5 SMT Solver (System Description)”. 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. 95–105.

Lachnitt, Hanna et al. (2025). “Improving the SMT Proof Reconstruction Pipeline in Isabelle/HOL”. In: Interactive Theorem Proving (ITP). Ed. by Yannick Forster and Chantal Keller. Vol. 352. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 26:1–26:22.

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.

References

- Liew, Vincent et al. (2020). "Verifying Properties of Bit-vector Multiplication Using Cutting Planes Reasoning". In: Formal Methods In Computer-Aided Design (FMCAD). IEEE, pp. 194–204.
- Lipparini, Enrico et al. (2025). "Boosting MCSat Modulo Nonlinear Integer Arithmetic via Local Search". In: Proc. Conference on Automated Deduction (CADE). Ed. by Clark W. Barrett and Uwe Waldmann. Vol. 15943. Lecture Notes in Computer Science. Springer, pp. 95–115.
- McCarthy, John (1993). "Towards a mathematical science of computation". In: Program Verification. Springer, pp. 35–56.
- 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.
- Mohamed, Abdalrhman et al. (2025). "lean-smt: An SMT Tactic for Discharging Proof Goals in Lean". In: ed. by Ruzica Piskac and Zvonimir Rakamaric. Vol. 15933. Lecture Notes in Computer Science. Springer, pp. 197–212.
- Moura, Leonardo Mendonça de and Nikolaj Bjørner (2008). "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.
- — (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.
- 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.
- Niemetz, Aina, Mathias Preiner, and Yoni Zohar (2024). "Scalable Bit-Blasting with Abstractions". In: Computer Aided Verification (CAV), Part I. Ed. by Arie Gurfinkel and Vijay Ganesh. Vol. 14681. Lecture Notes in Computer Science. Springer, pp. 178–200.
- 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.
- 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.
- Promies, Valentin et al. (2025). "More is Less: Adding Polynomials for Faster Explanations in NLSAT". In: Proc. Conference on Automated Deduction (CADE). Ed. by Clark W. Barrett and Uwe Waldmann. Vol. 15943. Lecture Notes in Computer Science. Springer, pp. 116–135.
- Rath, Jakob et al. (2024). "PolySAT: Word-level Bit-vector Reasoning in Z3". In: Verified Software. Theories, Tools and Experiments - 16th International Conference, VSTTE 2024, Prague, Czech Republic, October 14-15, 2024, Ed. by Jonathan Protzenko and Azalea Raad. Vol. 15525. Lecture Notes in Computer Science. Springer, pp. 47–69.
- 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.

References

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.

Schurr, Hans-Jörg et al. (2021). "Alethe: Towards a Generic SMT Proof Format (extended abstract)". In: CoRR abs/2107.02354. arXiv: 2107.02354.

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.

Sun, Hao and Zhendong Su (2025). "Prove It to the Kernel: Precise Extension Analysis via Proof-Guided Abstraction Refinement". In: Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles. SOSP '25. Lotte Hotel World, Seoul, Republic of Korea: Association for Computing Machinery, 736–751.

Varatalu, Ian Erik et al. (2025). "Regex Decision Procedures in Extended RE#". In: ed. by Ruzica Piskac and Zvonimir Rakamaric. Vol. 15933. Lecture Notes in Computer Science. Springer, pp. 106–129.

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.

Wilson, Amalee et al. (2023). "Partitioning Strategies for Distributed SMT Solving". In: Formal Methods In Computer-Aided Design (FMCAD). Ed. by Alexander Nadel and Kristin Yvonne Rozier. IEEE, pp. 199–208.

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.

References

- Zhang, Yuanzhuo, Zhoulai Fu, and Binoy Ravindran (2026). Scalable Floating-Point Satisfiability via Staged Optimization. arXiv: 2601.04492 [cs.PL].
- Zhou, Yi et al. (2025). "Cazamariposas: Automated Instability Debugging in SMT-Based Program Verification". In: Proc. Conference on Automated Deduction (CADE). Ed. by Clark W. Barrett and Uwe Waldmann. Vol. 15943. Lecture Notes in Computer Science. Springer, pp. 75–94.