AN OVERVIEW OF SATISFIABILITY MODULO THEORIES AND ITS APPLICATIONS

ETAPS 2019

Cesare Tinelli

April 8, 2019

The University of Iowa

ACKNOWLEDGMENTS

Many thanks to **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.

INTRODUCTION

Historically:

Automated logical reasoning achieved through uniform theorem-proving procedures for First Order Logic

(e.g., resolution, superposition, and tableaux calculi)

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

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

Two successful examples of this trend:

SAT: propositional formalization, Boolean reasoning

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

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

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

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

Example

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

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

$$n > 3 * m + 1 \;\wedge\; \big(f(n) \le \mathit{head}(l_1) \;\vee\; l_2 = f(n) :: l_1\big)$$

Linear Arithm.
(LIA)

THE BASIC SMT PROBLEM

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

Example

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

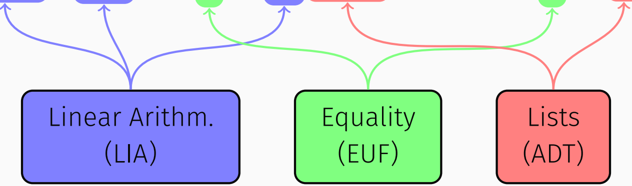
Linear Arithm. (LIA)

Equality (EUF)

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

Example

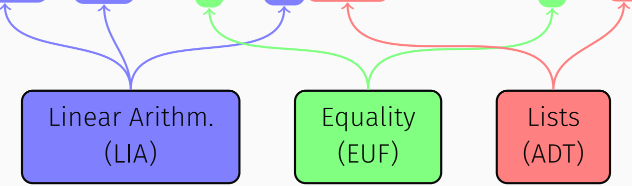
$$n > 3 * m + 1 \;\wedge\; (f(n) \leq head(l_1) \;\vee\; l_2 = f(n) :: l_1)$$



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

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

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:

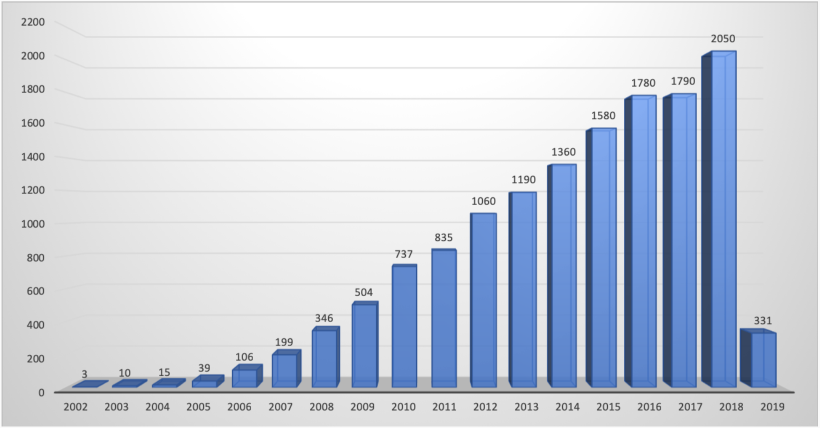
- instead of building a special-purpose tool
- translate problem into a logical formula
- use an SMT solver as backend reasoner

Not only easier, often better

THE EXPLOSION OF SMT

Google Scholar

“Satisfiability Modulo Theories” OR “SMT Solver”



	Citations	Google Scholar Hits
Z3	5,068[1]	7,870
CVC Lite, CVC 3, 4	1,560[2]	2,030
Yices 1, 2	972[3]	2,430
MathSat 3, 4, 5	628[4]	1,010

[1] [DMB08], 2018 ETAPS Test of Time Award to Z3 developers
[2] [BT07, BT07, BCD+11]
[3] [DdM06b, Dut14]
[4] [BBC+05b, BCF+08, CGSS13]

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

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

Planning

- motion planning
- nonlinear PDDL planning

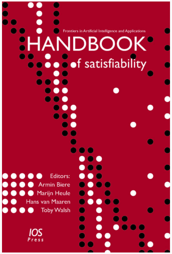
Machine Learning

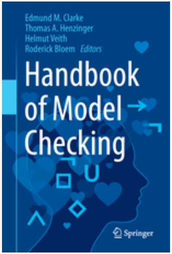
- verification of deep NNs

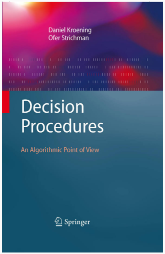
Business

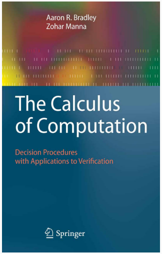
- verification of business rules
- spreadsheet debugging

Handbook chapters and books [BSST09, BT18, BM07, KS08]









Online

- SMT-LIB at `http://smt-lib.org`
- SMT-COMP at `http://smt-comp.org`

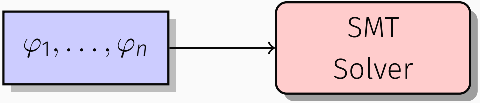
FUNCTIONALITY

LEGEND

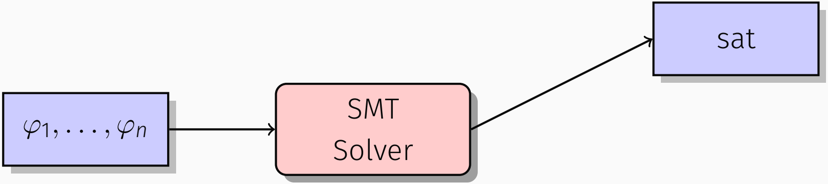
v	value — i.e., distinguished variable-free term
$\varphi[\mathbf{x}]$	formula with free vars from $\mathbf{x} = (x_1, \ldots, x_n)$
$\varphi[\mathbf{x} \mapsto \mathbf{v}]$	formula obtained by replacing free occurrences of variables from $\mathbf{x}$ in φ with corresponding values from $\mathbf{v} =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

SMT SOLVER BASIC FUNCTIONALITY

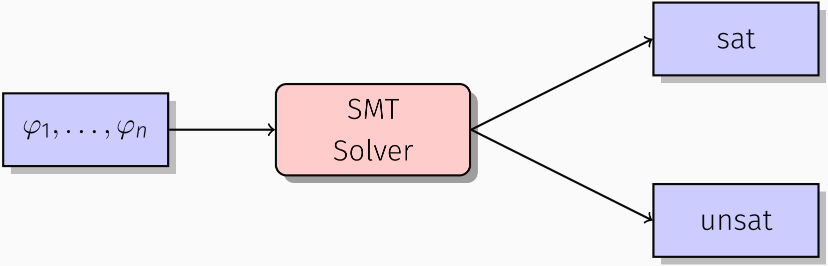
Background theory T



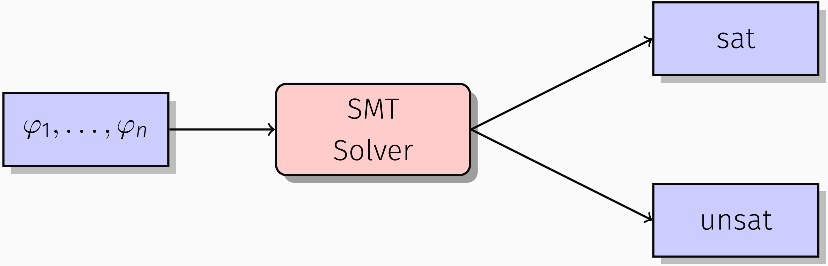
Background theory T



Background theory T



Background theory T

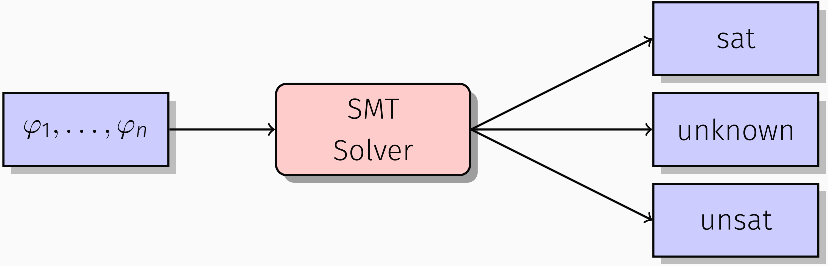


sat/unsat: there is a/no model M of T such that

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

SMT SOLVER BASIC FUNCTIONALITY

Background theory T

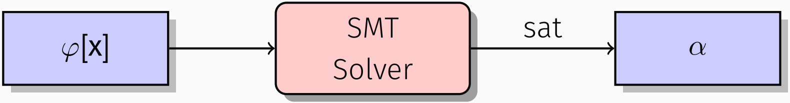


sat/unsat: there is a/no model M of T such that

$$M \models \varphi_1 \land \cdots \land \varphi_n$$

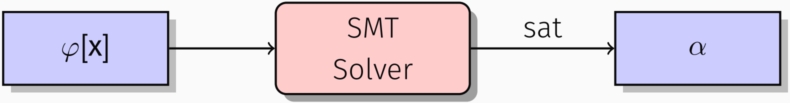
unknown: inconclusive — because of resource limits or incompleteness

Background theory T



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

Background theory T

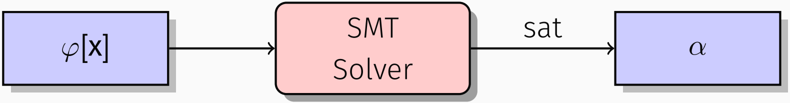


α is a *satisfying assignment* for $\mathbf{x} = (x_1, \ldots, x_n)$:

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

SMT SOLVER OUTPUT: SATISFYING ASSIGNMENTS

Background theory T

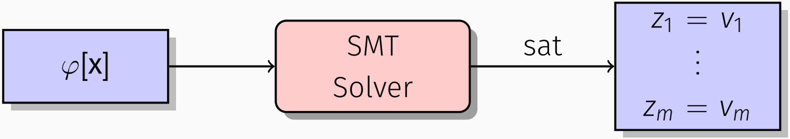


α is a *satisfying assignment* for $\mathbf{x} = (x_1, \ldots, x_n)$:

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

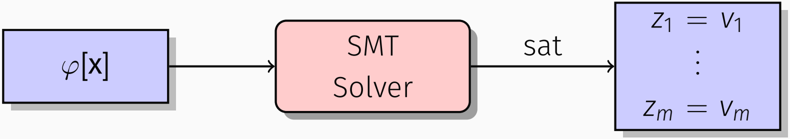
Note.
$\mathbf{x}$ may consist of first- and second-order variables
(aka, *uninterpreted constants and function symbols*)

Background theory T



$\mathbf{z} = \mathbf{v}$ is a *backbone* for φ:

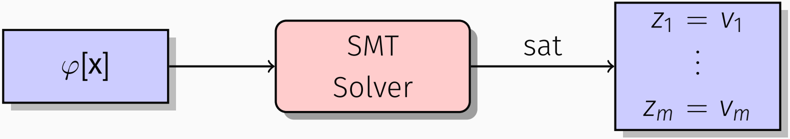
Background theory T



$\mathbf{z} = \mathbf{v}$ is a *backbone* for φ:

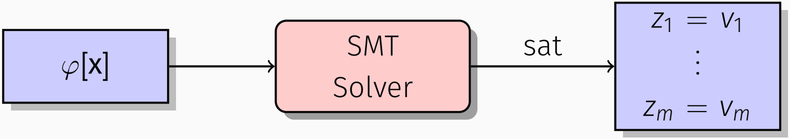
1. $\mathbf{z} \subseteq \mathbf{x}$
2. $\varphi \models_T \mathbf{z} = \mathbf{v}$
3. $\mathbf{z}$ is maximal (or largish)

Background theory T



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

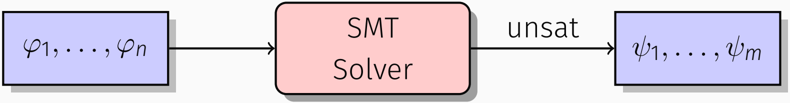
Background theory T



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

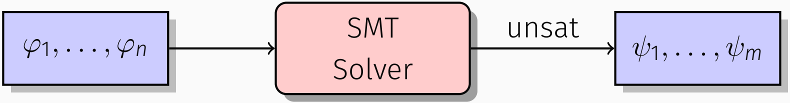
1. $\mathbf{z} \subseteq \mathbf{x}$
2. $\mathbf{y} = \mathbf{x} \setminus \mathbf{z}$
3. $\forall \mathbf{y}\,(\varphi \wedge \mathbf{z} = \mathbf{v})$ is satisfiable in T
4. $\mathbf{z}$ is minimal (or smallish)

Background theory T



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

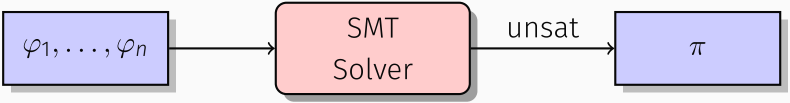
Background theory T



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

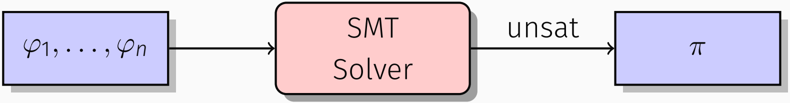
1. $\{\psi_1, \ldots, \psi_m\} \subseteq \{\varphi_1, \ldots, \varphi_n\}$
2. $\{\psi_1, \ldots, \psi_m\}$ is unsat in T
3. $\{\psi_1, \ldots, \psi_m\}$ is minimal (or smallish)

Background theory T



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

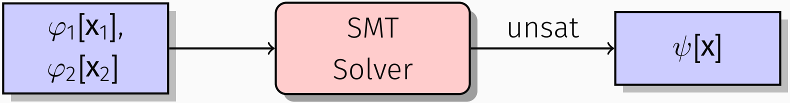
Background theory T



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

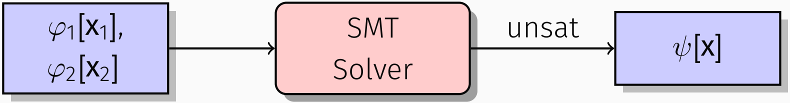
1. π is a proof term in some formal proof system
2. π expresses a refutation of $\{\varphi_1, \ldots, \varphi_n\}$
3. π can be efficiently checked by an external proof checker

Background theory T



ψ is a logical *interpolant* of φ_1 and φ_2:

Background theory T

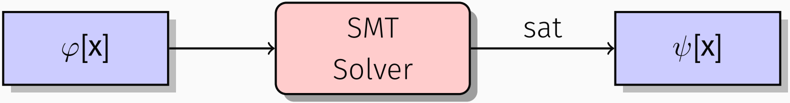


ψ is a logical *interpolant* of φ_1 and φ_2:

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

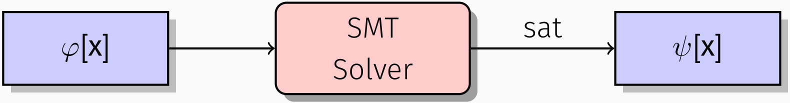
EXTENDED FUNCTIONALITY: PRIME IMPLICATE COMPUTATION

Background theory T



ψ is a *prime implicate* of φ:

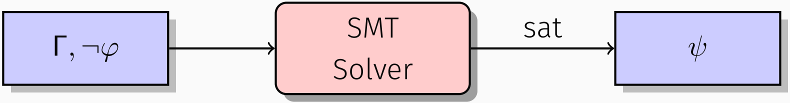
Background theory T



ψ is a *prime implicate* of φ:

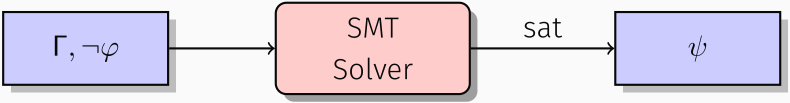
1. ψ is a disjunction of literals
2. $\varphi \models_T \psi$
3. there is no disjunction of literals $\psi' \notin \{\varphi, \psi\}$ s.t. $\varphi \models_T \psi'$ and $\psi' \models_T \psi$

Background theory T



ψ is an *abduction hypothesis* for φ wrt Γ:

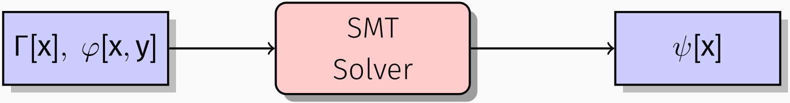
Background theory T



ψ is an *abduction hypothesis* for φ wrt Γ:

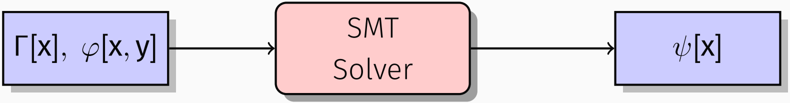
1. Γ, ψ is satisfiable in T
2. $\Gamma, \psi \models_T \varphi$
3. ψ is maximal, e.g., with respect to $\models_T$
   (if ψ' satisfies 1 and 2 and $\psi \models_T \psi'$ then $\psi' \models_T \psi$)

Background theory T



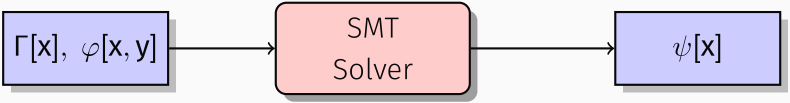
EXTENDED FUNCTIONALITY: QUANTIFIER ELIMINATION

Background theory T



ψ is a *projection* of φ over **y** with respect to Γ:

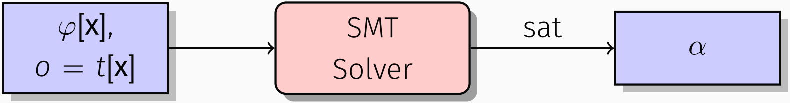
Background theory T



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

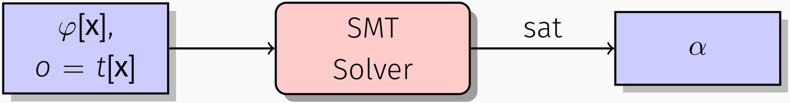
1. $\Gamma \models_T \psi \Leftrightarrow \exists \mathbf{y}\, \varphi$

Background theory T



α is a *an optimal assignment* for φ:

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. $M \models \varphi[\mathbf{x} \mapsto \mathbf{v}]$ for some model M of T
3. α minimizes/maximizes *objective* o

BACKGROUND THEORIES

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

EQUALITY AND UNINTERPRETED FUNCTIONS (EUF) [NO80, NO07]

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

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

$$\begin{aligned}
\text{Reflexivity:} &\quad \forall x : \sigma.\ x = x \\
\text{Symmetry:} &\quad \forall x : \sigma.\ x = y \Rightarrow y = x \\
\text{Transitivity:} &\quad \forall x, y : \sigma.\ x = y \land y = z \Rightarrow x = z \\
\text{Congruence:} &\quad \forall \mathbf{x}, \mathbf{y} : \sigma.\ \mathbf{x} = \mathbf{y} \Rightarrow f(\mathbf{x}) = f(\mathbf{y})
\end{aligned}$$

Example

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

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: $\forall a, b, i.\ a \neq b \Rightarrow \exists i.\ a[i] \neq b[i]$

Example

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

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

- Bounds: $x \bowtie k$ with $\bowtie \in \{<, >, \leq, \geq, =\}$ [BBC+05a]
- Difference constraints: $x - y \bowtie k$, with $\bowtie \in \{<, >, \leq, \geq, =\}$ [NO05, WIGG05, CM06]
- UTVPI: $\pm x \pm y \bowtie k$, with $\bowtie \in \{<, >, \leq, \geq, =\}$ [LM05]
- Linear arithmetic, e.g: $2x - 3y + 4z \leq 5$ [DdM06a]
- Non-linear arithmetic, e.g: $2xy + 4xz^2 - 5y \leq 10$ [BLNM+09, ZM10, JdM12]

Family of user-definable theories

Example

$$
\begin{aligned}
\text{Color} &:= \text{red} \mid \text{green} \mid \text{blue} \\
\text{List}(\alpha) &:= \text{nil} \mid (\text{head} : \alpha) :: (\text{tail} : \text{List}(\alpha))
\end{aligned}
$$

$$
\begin{aligned}
\text{Distinctiveness:}\quad & \forall h, t.\ \text{nil} \neq h :: t \\
\text{Exhaustiveness:}\quad & \forall l.\ l = \text{nil} \vee \exists h, t.\ h :: t \\
\text{Injectivity:}\quad & \forall h_1, h_2, t_1, t_2. \\
& \qquad h_1 :: t_1 = h_2 :: t_2 \Rightarrow h_1 = h_2 \wedge t_1 = t_2 \\
\text{Selectors:}\quad & \forall h, t.\ \text{head}(h :: t) = h \wedge \text{tail}(h :: t) = t \\
\text{(Non-circularity:}\quad & \forall l, x_1, \ldots, x_n.\ l \neq x_1 :: \cdots :: x_n :: l)
\end{aligned}
$$

- Strings and regular expressions [KGG+09, LRT+14]
- Floating point arithmetic [BDG+14, ZWR14]
- Finite sets with cardinality [BRBT16]
- Finite relations [MRTB17]
- Transcendental Functions [GKC13]
- Ordinary differential equations [GKC13]
- ...

APPLICATIONS

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

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)

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 $\mathbf{x}$ of *state vars*

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 $\mathbf{x}$ of *state vars*
3. Encode system M as T-formulas $(I[\mathbf{x}], R[\mathbf{x}, \mathbf{x}'])$
   where
   - I encodes M's initial state condition and
   - R encodes M's transition relation

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 $\mathbf{x}$ of *state vars*
3. Encode system M as T-formulas $(I[\mathbf{x}], R[\mathbf{x}, \mathbf{x}'])$ where
   - I encodes M's initial state condition and
   - R encodes M's transition relation
4. Encode S as a T-formula $B[\mathbf{x}]$

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 $\mathbf{x}$ of *state vars*
3. Encode system M as T-formulas $(I[\mathbf{x}], R[\mathbf{x}, \mathbf{x}'])$ where
   - I encodes M's initial state condition and
   - R encodes M's transition relation
4. Encode S as a T-formula $B[\mathbf{x}]$
5. Find a k such that $I[\mathbf{x}_0] \wedge R[\mathbf{x}_0, \mathbf{x}_1] \wedge \cdots \wedge R[\mathbf{x}_{k-1}, \mathbf{x}_k] \wedge B[\mathbf{x}_k]$ is satisfiable in T

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 $\mathbf{x}$ of *state vars*
3. Encode system M as T-formulas $(I[\mathbf{x}], R[\mathbf{x}, \mathbf{x}'])$
   where
   - I encodes M's initial state condition and
   - R encodes M's transition relation
4. Encode S as a T-formula $P[\mathbf{x}]$

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 $\mathbf{x}$ of *state vars*
3. Encode system M as T-formulas $(I[\mathbf{x}], R[\mathbf{x}, \mathbf{x}'])$ where
   - I encodes M's initial state condition and
   - R encodes M's transition relation
4. Encode S as a T-formula $P[\mathbf{x}]$
5. Prove that $P[\mathbf{x}]$ holds in all reachable states of $(I[\mathbf{x}], R[\mathbf{x}, \mathbf{x}'])$

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 <= n

Example (Parametric Resettable Counter)

System

Vars

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

Initialization

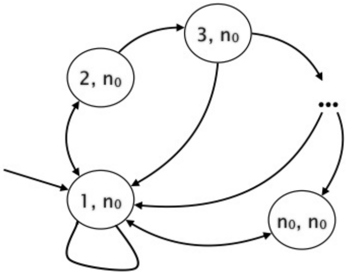
c := 1
n := n_0

Transitions

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

Property

c <= n



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

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 <= n

Encoding in T = LIA

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

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

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

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

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

$M = ($

To pro
to sho
i.e.,

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 $(c, n, r) = (4, 3, \mathit{false})$ and $(c, n, r)' = (5, 3, \mathit{false})$

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

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

Various approaches:

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

Various approaches:

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

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

Various approaches:

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

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

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

Various approaches:

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

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

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

Example

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

Check if the swap is correct:

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

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

Example

```c
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

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

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

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 [dMB10]

Example (Binary Search)

```
//
//
//
//
i
}
```

Main approach

1. Compile source and annotations to program in Dijkstra's core language:

$$\begin{aligned} S, T \quad ::= \quad & x = t \mid \textbf{havoc}\ x \mid \textbf{assert}\ \varphi \mid \textbf{assume}\ \varphi \mid \\ & S; T \mid S \ [] \ T \end{aligned}$$

2. Convert core program to SMT using the weakest liberal precondition transformer wp:

$$wp(x = t, \varphi) = \varphi\{x \mapsto t\} \qquad wp(\textbf{assert}\ \psi, \varphi) = \psi \wedge \varphi$$
$$wp(\textbf{assume}\ \psi, \varphi) = \psi \Rightarrow \varphi \qquad wp(\textbf{havoc}\ x, \varphi) = \forall x\ \varphi$$
$$wp(S; T, \varphi) = wp(S, wp(T, \varphi))$$
$$wp(S \ [] \ T, \varphi) = wp(S, \varphi) \wedge wp(T, \varphi)$$

Example adapted from [dMB10]

Example (Binary Search)

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

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

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

Example (Binary Search)

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

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

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

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

Example (Binary Search)

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

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

$$inv = 0 \le l \wedge l \le h \wedge h \le n \wedge n \le |a| \wedge$$

$\forall i : \mathrm{Int}\ \ 0 \le$

$\forall i : \mathrm{Int}\ \ h \le$

SMT solver answer: Unsatisfiable

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

Synthesis

- Synthesize a function that satisfies a given high-level specification
- Already used extensively for hardware systems
- Particularly challenging for software

Synthesis

- Synthesize a function that satisfies a given high-level specification
- Already used extensively for hardware systems
- Particularly challenging for software

Recent interest

- Major new efforts by several research groups
- New syntax-guided synthesis (SyGuS) format
- SyGuS competition started in 2014
- New technique: Refutation-Based Synthesis in SMT [RDK+15]

Formalization in second-order logic

- Let $P[f, \mathbf{x}]$ be a property (specification) for a function f over some variables $\mathbf{x} = (x_1, x_2)$
- The synthesis problem is to determine the satisfiability of

$$\exists f.\, \forall \mathbf{x}.\, P[f, \mathbf{x}]$$

Formalization in second-order logic

- Let $P[f, \boldsymbol{x}]$ be a property (specification) for a function f over some variables $\boldsymbol{x} = (x_1, x_2)$
- The synthesis problem is to determine the satisfiability of

$$\exists f.\, \forall \boldsymbol{x}.\, P[f, \boldsymbol{x}]$$

Example

Maximum of 2 values

$$P[f, \boldsymbol{x}] = f(\boldsymbol{x}) \geq x_1 \wedge f(\boldsymbol{x}) \geq x_2 \wedge (f(\boldsymbol{x}) = x_1 \vee f(\boldsymbol{x}) = x_2)$$

Formalization in second-order logic

- Let $P[f, \mathbf{x}]$ be a property (specification) for a function f over some variables $\mathbf{x} = (x_1, x_2)$
- The synthesis problem is to determine the satisfiability of

$$\exists f. \forall \mathbf{x}. P[f, \mathbf{x}]$$

Example

Maximum of 2 values

$$P[f, \mathbf{x}] = f(\mathbf{x}) \geq x_1 \wedge f(\mathbf{x}) \geq x_2 \wedge (f(\mathbf{x}) = x_1 \vee f(\mathbf{x}) = x_2)$$

Problem: SMT only understands first-order logic

Single-invocation properties

- Every occurrence of f is of the form $f(\mathbf{x})$
  - Previous example is single-invocation
  - Not single-invocation: $\forall \mathbf{x}.\, f(x_1, x_2) = f(x_2, x_1)$
- When the synthesis property is single-invocation, it can written as $\exists f.\, \forall \mathbf{x}.\, P[f(\mathbf{x}), \mathbf{x}]$

Single-invocation properties

- Every occurrence of f is of the form $f(\mathbf{x})$
  - Previous example is single-invocation
  - Not single-invocation: $\forall \mathbf{x}.\, f(x_1, x_2) = f(x_2, x_1)$
- When the synthesis property is single-invocation, it can written as $\exists f.\, \forall \mathbf{x}.\, P[f(\mathbf{x}), \mathbf{x}]$

Note that:

$$\exists f.\, \forall \mathbf{x}.\, P[f(\mathbf{x}), \mathbf{x}] \tag{1}$$

is equivalent to

$$\forall \mathbf{x}.\, \exists y\, P[y, \mathbf{x}] \tag{2}$$

because (1) is the Skolemization of (2) which is first-order!

Proving the validity of

$$\forall \boldsymbol{x}.\, \exists y\, P[y, \boldsymbol{x}]$$

Proving the validity of

$$\forall \boldsymbol{x}.\, \exists y\, P[y, \boldsymbol{x}]$$

is equivalent to proving the unsatisfiability of

$$\exists \boldsymbol{x}.\, \forall y\, \neg P[y, \boldsymbol{x}]$$

or the unsatisfiability of

$$\forall y\, \neg P[y, \boldsymbol{c}]$$

for some fresh constants $\boldsymbol{c}$

How does an SMT solver show that

$$\forall y\, \neg P[y, \mathbf{c}] \text{ is unsatisfiable?}$$

How does an SMT solver determine that

$$\forall y\, \neg P[y, \mathbf{c}] \text{ is unsatisfiable?}$$

SMT solvers use heuristic instantiation [GBT07, GdM09, RTGK13] to produce a set of unsatisfiable quantifier-free formulas:

$$\{\neg P[t_1[\mathbf{c}], \mathbf{c}], \neg P[t_2[\mathbf{c}], \mathbf{c}], \ldots, \neg P[t_n[\mathbf{c}], \mathbf{c}]\}$$

How does an SMT solver determine that

$$\forall y\, \neg P[y, \mathbf{c}] \text{ is unsatisfiable?}$$

SMT solvers use heuristic instantiation [GBT07, GdM09, RTGK13] to produce a set of unsatisfiable quantifier-free formulas:

$$\{\neg P[t_1[\mathbf{c}], \mathbf{c}], \neg P[t_2[\mathbf{c}], \mathbf{c}], \ldots, \neg P[t_n[\mathbf{c}], \mathbf{c}]\}$$

This also gives a constructive solution to the original synthesis problem:

$$f = \lambda \mathbf{x}.\, ite(P[t_1[\mathbf{x}], \mathbf{x}], t_1[\mathbf{x}], (\cdots ite(P[t_{n-1}[\mathbf{x}], \mathbf{x}], t_{n-1}[\mathbf{x}], t_n[\mathbf{x}]) \cdots))$$

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

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

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

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

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

SMT solver solution

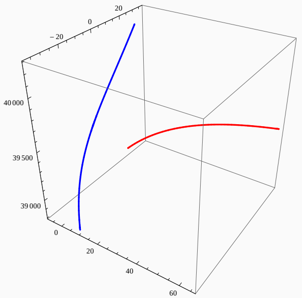
$t_{1,1} \mapsto 5, \quad t_{1,2} \mapsto 7$

$t_{2,1} \mapsto 2, \quad t_{2,2} \mapsto 6$

$t_{3,1} \mapsto 0, \quad t_{3,2} \mapsto 3$

$((t_{1,2} \geq t_{3,2} + 3) \vee (t_{3,2} \geq t_{1,2} + 1))$

$((t_{2,2} \geq t_{3,2} + 3) \vee (t_{3,2} \geq t_{2,2} + 1))$



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

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

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

$$T_1^x(t) = 3.2484 + 270.7t + 433.12t^2 - 324.83999t^3$$

$$T_1^y(t) = 15.1592 + 108.28t + 121.2736t^2 - 649.67999t^3$$

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

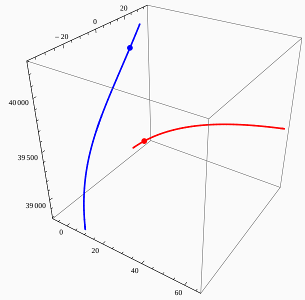
$$T_2^x(t) = 1.0828 - 135.35t + 234.9676t^2 2 + 3248.4t^3$$

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

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

Example from [NM12]

AIRCRAFT TRAJECTORY CONFLICT DETECTION



$$T_1^x(t) \ldots$$
$$T_1^y(t \ldots$$
$$T_1^z(t \ldots$$

SMT solver solution

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

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

$$H = 5\,nm \qquad V = 1000\,ft \qquad 0 \leq t \leq \frac{1}{20}h$$
$$|T_1^z(t) - T_2^z(t)| \leq V$$
$$(T_1^x(t) - T_2^x(t))^2 + (T_1^y(t) - T_2^y(t))^2 \leq H^2$$

Example from [NM12]

REFERENCES

1. C. Barrett and C. Tinelli. **Satisfiability Modulo Theories**. In Handbook of Model Checking. Springer, 2018.
2. L. de Moura and N. Bjørner. **Satisfiability modulo theories: introduction and applications**. Communications of the ACM, 54(9):69-77, 2011.
3. C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli. **Satisfiability Modulo Theories**. In Handbook of Satisfiability. IOS Press, 2009.
4. R. Sebastiani. **Lazy Satisfiability Modulo Theories**. Journal on Satisfiability, Boolean Modeling and Computation 3:141-224, 2007.

References

M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Schulz, and R. Sebastiani.
An incremental and layered procedure for the satisfiability of linear arithmetic logic.
In *Tools and Algorithms for the Construction and Analysis of Systems, 11th Int. Conf., (TACAS)*, volume 3440 of *Lecture Notes in Computer Science*, pages 317–333, 2005.

Marco Bozzano, Roberto Bruttomesso, Alessandro Cimatti, Tommi A. Junttila, Peter van Rossum, Stephan Schulz, and Roberto Sebastiani.
The MathSAT 3 system.
In Robert Nieuwenhuis, editor, *Proceedings of the 20th International Conference on Automated Deduction (CADE-20)*, volume 3632 of *Lecture Notes in Computer Science*, pages 315–321. Springer, 2005.

Clark Barrett, Christopher Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli.
CVC4.
In G. Gopalakrishnan and S. Qadeer, editors, *23rd International Conference on Computer Aided Verification (CAV'11), Snowbird, Utah*, volume 6806 of *Lecture Notes in Computer Science*, pages 171–177. Springer, 2011.

Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, and Roberto Sebastiani.
The MathSAT 4 SMT solver.
In Aarti Gupta and Sharad Malik, editors, *Proceedings of the 20th international conference on Computer Aided Verification (CAV'08)*, volume 5123 of *Lecture Notes in Computer Science*, pages 299–303. Springer, 2008.

Martin Brain, Vijay D'Silva, Alberto Griggio, Leopold Haller, and Daniel Kroening.
Deciding floating-point logic with abstract conflict driven clause learning.
Formal Methods in System Design, 45(2):213–245, 2014.

C. Borralleras, S. Lucas, R. Navarro-Marset, E. Rodríguez-Carbonell, and A. Rubio.
Solving Non-linear Polynomial Arithmetic via SAT Modulo Linear Arithmetic.
In R. A. Schmidt, editor, *22nd International Conference on Automated Deduction , CADE-22*, volume 5663 of *Lecture Notes in Computer Science*, pages 294–305. Springer, 2009.

Aaron R Bradley and Zohar Manna.
The calculus of computation: decision procedures with applications to verification.
Springer Science & Business Media, 2007.

M. Bofill, R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio.
A Write-Based Solver for SAT Modulo the Theory of Arrays.
In *Formal Methods in Computer-Aided Design, FMCAD*, pages 1–8, 2008.

Kshitij Bansal, Andrew Reynolds, Clark Barrett, and Cesare Tinelli.
A new decision procedure for finite sets and cardinality constraints in SMT.
In Nicola Olivetti and Ashish Tiwari, editors, *Proceedings of the 8th International Joint Conference on Automated Reasoning, Coimbra, Portugal*, volume 9706 of *Lecture Notes in Computer Science*, pages 82–98. Springer International Publishing, 2016.

Clark W Barrett, Roberto Sebastiani, Sanjit A Seshia, and Cesare Tinelli.
Satisfiability modulo theories.
Handbook of satisfiability, 185:825–885, 2009.

Clark Barrett, Igor Shikanian, and Cesare Tinelli.
An abstract decision procedure for satisfiability in the theory of recursive data types.
Electronic Notes in Theoretical Computer Science, 174(8):23–37, 2007.

Clark Barrett and Cesare Tinelli.
CVC3.
In W. Damm and H. Hermanns, editors, *Proceedings of the 19th International Conference on Computer Aided Verification (CAV'07), Berlin, Germany*, Lecture Notes in Computer Science. Springer, 2007.

Clark Barrett and Cesare Tinelli.
Satisfiability modulo theories.
In Edmund Clarke, Tom Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*. Springer, 2018.

A. Cimatti, A. Griggio, B. Schaafsma, and R. Sebastiani.
The MathSAT5 SMT solver.
In Nir Piterman and Scott A. Smolka, editors, *Proceedings of TACAS*, volume 7795 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2013.

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

Bruno Dutertre and Leonardo de Moura.
A Fast Linear-Arithmetic Solver for DPLL(T).
In T. Ball and R. B. Jones, editors, *18th International Conference on Computer Aided Verification, CAV'06*, volume 4144 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2006.

Bruno Dutertre and Leonardo de Moura.
The YICES SMT solver.
Technical report, SRI International, 2006.

Leonardo De Moura and Nikolaj Bjørner.
Z3: an efficient smt solver.
In *Proceedings of the Theory and practice of software, 14th international conference on Tools and algorithms for the construction and analysis of systems*, TACAS'08/ETAPS'08, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.

L. de Moura and N. Bjørner.
Generalized, efficient array decision procedures.
In *9th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2009*, pages 45–52. IEEE, 2009.

Leonardo de Moura and Nikolaj Bjørner.
Bugs, moles and skeletons: Symbolic reasoning for software development.
In *Proceedings pf the 5th International Joint Conference on Automated Reasoning (IJCAR 2010)*, volume 6173 of *Lecture Notes in Computer Science*, pages 400–411. Springer, 2010.

Leonardo De Moura and Nikolaj Bjørner.
Satisfiability modulo theories: introduction and applications.
Communications of the ACM, 54(9):69–77, 2011.

Bruno Dutertre.
Yices 2.2.
In Armin Biere and Roderick Bloem, editors, *Proceedings of the 26th International Conference on Computer Aided Verification (CAV 2014)*, volume 8559 of *Lecture Notes in Computer Science*, pages 737–744. Springer, 2014.

Yeting Ge, Clark Barrett, and Cesare Tinelli.
Solving quantified verification conditions using satisfiability modulo theories.
In F. Pfenning, editor, *Proceedings of the 21st International Conference on Automated Deduction (CADE-21), Bremen, Germany*, Lecture Notes in Computer Science. Springer, 2007.

Yeting Ge and Leonardo de Moura.
Complete instantiation for quantified formulas in satisfiabiliby modulo theories.
In Ahmed Bouajjani and Oded Maler, editors, *Proceedings of the 21st International Conference on Computer Aided Verification*, volume 5643 of *Lecture Notes in Computer Science*, pages 306–320. Springer, 2009.

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

Dejan Jovanović and Leonardo de Moura.
Solving Non-linear Arithmetic.
In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *6th International Joint Conference on Automated Reasoning (IJCAR '12)*, volume 7364 of *Lecture Notes in Computer Science*, pages 339–354. Springer, 2012.

Adam Kiezun, Vijay Ganesh, Philip J Guo, Pieter Hooimeijer, and Michael D Ernst.
HAMPI: a solver for string constraints.
In *Proceedings of the eighteenth international symposium on Software testing and analysis*, pages 105–116. ACM, 2009.

Daniel Kroening and Ofer Strichman.
Decision procedures: an algorithmic point of view.
Springer Science & Business Media, 2008.

Shuvendu K. Lahiri and Madanlal Musuvathi.
An Efficient Decision Procedure for UTVPI Constraints.
In B. Gramlich, editor, *5th International Workshop on Frontiers of Combining Systems, FroCos'05*, volume 3717 of *Lecture Notes in Computer Science*, pages 168–183. Springer, 2005.

Tianyi Liang, Andrew Reynolds, Cesare Tinelli, Clark Barrett, and Morgan Deters.
A DPLL(T) theory solver for a theory of strings and regular expressions.
In *International Conference on Computer Aided Verification*, pages 646–662. Springer, 2014.

John McCarthy.
Towards a mathematical science of computation.
In *Program Verification*, pages 35–56. Springer, 1993.

Baoluo Meng, Andrew Reynolds, Cesare Tinelli, and Clark Barrett.
Relational constraint solving in SMT.
In Leonardo de Moura, editor, *Proceedings of the 26th International Conference on Automated Deduction*, volume 10395 of *Lecture Notes in Computer Science*, pages 148–165. Springer, 2017.

Anthony Narkawicz and César A Munoz.
Formal verification of conflict detection algorithms for arbitrary trajectories.
Reliable Computing, 17(2):209–237, 2012.

Greg Nelson and Derek C. Oppen.
Fast decision procedures based on congruence closure.
Journal of the ACM, 27(2):356–364, 1980.

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

Robert Nieuwenhuis and Albert Oliveras.
Fast congruence closure and extensions.
Information and Computation, 205(4):557–580, 2007.

Andrew Reynolds and Jasmin Christian Blanchette.
A decision procedure for (co)datatypes in SMT solvers.
Journal of Automated Reasoning, 58(3):341–362, 2016.

Andrew Reynolds, Morgan Deters, Viktor Kuncak, Cesare Tinelli, and Clark W. Barrett.
Counterexample-guided quantifier instantiation for synthesis in SMT.
In Daniel Kroening and Corina S. Pasareanu, editors, *Proceedings of the 27th International Conference on Computer Aided Verification*, volume 9207 of *Lecture Notes in Computer Science*, pages 198–216. Springer, 2015.

Andrew Reynolds, Cesare Tinelli, Amit Goel, and Sava Krstić.
Finite model finding in SMT.
In *Proceedings of the 25th International Conference on Computer Aided Verification (St Petersburg, Russia)*, volume 8044 of *LNCS*, pages 640–655. Springer, 2013.

A. Stump, C. W. Barrett, D. L. Dill, and J. R. Levitt.
A Decision Procedure for an Extensional Theory of Arrays.
In *16th Annual IEEE Symposium on Logic in Computer Science, LICS'01*, pages 29–37. IEEE Computer Society, 2001.

C. Wang, F. Ivancic, M. K. Ganai, and A. Gupta.
Deciding Separation Logic Formulae by SAT and Incremental Negative Cycle Elimination.
In G. Sutcliffe and A. Voronkov, editors, *12h International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR'05*, volume 3835 of *Lecture Notes in Computer Science*, pages 322–336. Springer, 2005.

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

Aleksandar Zeljić, Christoph M Wintersteiger, and Philipp Rümmer.
Approximations for model construction.
In *International Joint Conference on Automated Reasoning*, pages 344–359. Springer, 2014.