AN OVERVIEW OF SATISFIABILITY MODULO THEORIES AND ITS APPLICATIONS

ETAPS 2019

Cesare Tinelli

April 8, 2019
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.
OUTLINE

Introduction

SMT Solver Functionality

Background Theories

Applications
  Model Checking
  Software Verification
  Synthesis

Misc

References
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 procedure 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

**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
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**

$$n > 3 \times m + 1 \land (f(n) \leq \text{head}(l_1) \lor l_2 = f(n) :: l_1)$$
The Basic SMT Problem

Determining the satisfiability of a logical formula \textit{wrt} some combination \( T \) of background theories

**Example**

\[
n > 3 \times m + 1 \land ( f(n) \leq \text{head}(l_1) \lor l_2 = f(n) :: l_1 )
\]
Determining the **satisfiability** of a logical formula **wrt** some combination $T$ of **background theories**

**Example**

$$n > 3 \times m + 1 \land (f(n) \leq \text{head}(l_1) \lor l_2 = f(n) :: l_1)$$

**Linear Arithm.**

(LIA)
Determining the **satisfiability** of a logical formula **wrt** some combination \( T \) of **background theories**

**Example**

\[
n > 3 \times m + 1 \quad \land \quad \left( f(n) \leq \text{head}(l_1) \quad \lor \quad l_2 = f(n) :: l_1 \right)
\]

- Linear Arithm. (LIA)
- Equality (EUF)
Determining the **satisfiability** of a logical formula wrt some combination $T$ of background theories

**Example**

\[ n > 3 \times m + 1 \land (f(n) \leq \text{head}(l_1) \lor l_2 = f(n) :: l_1) \]

- Linear Arithm. (LIA)
- Equality (EUF)
- Lists (ADT)
Determining the satisfiability of a logical formula \( \text{wrt} \) some combination \( T \) of background theories

**Example**

\[
n > 3 \times m + 1 \land (f(n) \leq \text{head}(l_1) \lor l_2 = f(n) :: l_1)
\]

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
THE EXPLOSION OF SMT

“Satisfiability Modulo Theories” OR “SMT Solver”
### Popular SMT Solvers

<table>
<thead>
<tr>
<th>Solver</th>
<th>Citations</th>
<th>Google Scholar Hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z3</td>
<td>5,068&lt;sup&gt;1&lt;/sup&gt;</td>
<td>7,870</td>
</tr>
<tr>
<td>CVC Lite, CVC 3, 4</td>
<td>1,560&lt;sup&gt;2&lt;/sup&gt;</td>
<td>2,030</td>
</tr>
<tr>
<td>Yices 1, 2</td>
<td>972&lt;sup&gt;3&lt;/sup&gt;</td>
<td>2,430</td>
</tr>
<tr>
<td>MathSat 3, 4, 5</td>
<td>628&lt;sup&gt;4&lt;/sup&gt;</td>
<td>1,010</td>
</tr>
</tbody>
</table>

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

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

program verification
verification in separation logic
(non-)termination
loop invariant generation
procedure summaries
race analysis
concurrency errors detection
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
EVEN MORE APPLICATIONS OF SMT

Planning
- motion planning
- nonlinear PDDL planning

Machine Learning
- verification of deep NNs

Business
- verification of business rules
- spreadsheet debugging
MORE INFORMATION ON SMT

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

Online

· SMT-LIB at http://smt-lib.org
· SMT-COMP at http://smt-comp.org
\( \nu \) value — i.e., distinguished variable-free term

\( \varphi[x] \) formula with free vars from \( x = (x_1, \ldots, x_n) \)

\( \varphi[x \mapsto v] \) formula obtained by replacing free occurrences of variables from \( x \) in \( \varphi \) with corresponding values from \( v = (v_1, \ldots, v_n) \)

\( x = v \) \( x_1 = v_1 \land \cdots \land x_n = v_n \)

\( z \subseteq x \) every element of \( z \) occurs in \( x \)

\( M \models \varphi \) model \( M \) satisfies formula \( \varphi \)

\( \varphi \models_T \psi \) formula \( \varphi \) entails formula \( \psi \) in theory \( T \)
Background theory $T$

\[ \varphi_1, \ldots, \varphi_n \rightarrow \text{SMT Solver} \]
Background theory $T$

$\varphi_1, \ldots, \varphi_n$ → SMT Solver → sat
Background theory $T$

\[ \varphi_1, \ldots, \varphi_n \rightarrow \text{SMT Solver} \rightarrow \text{sat} \]

\[ \varphi_1, \ldots, \varphi_n \rightarrow \text{SMT Solver} \rightarrow \text{unsat} \]
Background theory $T$

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

$$M \models \varphi_1 \land \cdots \land \varphi_n$$
Background theory $T$

$\varphi_1, \ldots, \varphi_n$ → SMT Solver →

- sat
- unknown
- unsat

**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 $T$

$\varphi[x] \rightarrow \text{SMT Solver} \rightarrow \text{sat} \rightarrow \alpha$

$\alpha$ is a satisfying assignment for $x = (x_1, \ldots, x_n)$:
Background theory $T$

$\varphi[x]$ \rightarrow \text{SMT Solver} \rightarrow \text{sat} \rightarrow \alpha$

$\alpha$ is a **satisfying assignment** for $x = (x_1, \ldots, x_n)$:

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

$\varphi[x]$ \rightarrow \text{SMT Solver} \rightarrow \text{sat} \rightarrow \alpha$

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

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

Note.

$x$ may consist of first- \text{and} second-order variables (aka, \textit{uninterpreted constants and function symbols})
Background theory $T$

$\varphi[x]$ → SMT Solver → sat → $Z_1 = v_1$

$\vdots$

$Z_m = v_m$

$z = v$ is a backbone for $\varphi$: 
Background theory $T$

$z = v$ is a **backbone** for $\varphi$:

1. $z \subseteq x$
2. $\varphi \models_T z = v$
3. $z$ is maximal (or largish)
Background theory $T$

$\varphi[x] \xrightarrow{\text{SMT Solver}} \text{sat} \xrightarrow{\text{z = v is a sat core for } \varphi:} \begin{align*} z_1 &= v_1 \\
&\vdots \\
z_m &= v_m \end{align*}$
Background theory $T$

$z = v$ is a sat core for $\varphi$:

1. $z \subseteq x$
2. $y = x \setminus z$
3. $\forall y (\varphi \land z = v)$ is satisfiable in $T$
4. $z$ is minimal (or smallish)
Background theory $T$

\[ \varphi_1, \ldots, \varphi_n \xrightarrow{\text{unsat}} \text{SMT Solver} \xrightarrow{\text{unsat}} \psi_1, \ldots, \psi_m \]

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

$\varphi_1, \ldots, \varphi_n$ 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 $T$

$\varphi_1, \ldots, \varphi_n \xrightarrow{\text{unsat}} \text{SMT Solver} \xrightarrow{\pi}$

$\pi$ is a checkable proof object for $\{\varphi_1, \ldots, \varphi_n\}$:
Background theory $T$

$\varphi_1, \ldots, \varphi_n$ $\rightarrow$ SMT Solver $\rightarrow$ unsat $\rightarrow$ $\pi$

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

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

$\varphi_1[x_1], \varphi_2[x_2] \rightarrow \text{SMT Solver} \rightarrow \text{unsat} \rightarrow \psi[x]$

$\psi$ is a logical interpolant of $\varphi_1$ and $\varphi_2$: 

$\varphi_1 \land \neg \psi \land \varphi_2$
Background theory $T$

\[ \varphi_1[x_1], \quad \varphi_2[x_2] \rightarrow \text{SMT Solver} \rightarrow \text{unsat} \rightarrow \psi[x] \]

$\psi$ is a logical interpolant of $\varphi_1$ and $\varphi_2$:

1. $\varphi_1 \models_T \psi$ and $\psi \models_T \neg \varphi_2$
2. $x = x_1 \cap x_2$
Background theory $T$

$\varphi[x]$ \rightarrow \text{SMT Solver} \rightarrow \text{sat} \rightarrow \psi[x]

$\psi$ is a prime implicate of $\varphi$: 
Background theory $T$

$\psi$ is a prime implicate of $\varphi$:

1. $\psi$ is a disjunction of literals
2. $\varphi \models_T \psi$
3. there is no disjunction of literals $\psi' \not\in \{\varphi, \psi\}$ such that $\varphi \models_T \psi'$ and $\psi' \models_T \psi$
Background theory $T$

\[ \Gamma, \neg \varphi \rightarrow_{\text{sat}} \psi \]

\( \psi \) is an \textit{abduction hypothesis} for \( \varphi \) wrt \( \Gamma \):
Background theory $T$

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

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

$\Gamma[x], \varphi[x, y]$ → SMT Solver → $\psi[x]$
Background theory $T$

$\Gamma[x], \varphi[x,y] \rightarrow \text{SMT Solver} \rightarrow \psi[x]$

$\psi$ is a projection of $\varphi$ over $y$ with respect to $\Gamma$:
Background theory $T$

$\Gamma[x], \varphi[x, y] \rightarrow \text{SMT Solver} \rightarrow \psi[x]$

$\psi$ is a projection of $\varphi$ over $y$ with respect to $Γ$:

1. $Γ \models_T ψ \iff \exists y \varphi$
Background theory $T$

$\alpha$ is a \textit{an optimal assignment} for $\varphi$: 

$\varphi[x], \
o \ = \ t[x]$
Background theory $T$

$\varphi[x], \ o = t[x]$ → SMT Solver → $\alpha$

$\alpha$ is a *an optimal assignment* for $\varphi$:

1. $\alpha = \{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\}$ for some values $v_1, \ldots, v_n$
2. $M \models \varphi[x \mapsto v]$ for some model $M$ of $T$
3. $\alpha$ 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 \[ 4 \cdot (x \gg 2) = x \& \sim 3 \]

Strings and RegExs \[ x = y \cdot z \land z \in ab^* \Rightarrow |x| > |y| \]

Arrays \[ i = j \Rightarrow \text{store}(a, i, x)[j] = x \]

Algebraic Data Types \[ x \neq \text{Leaf} \Rightarrow \exists l, r : \text{Tree}(\alpha). \exists a : \alpha. x = \text{Node}(l, a, r) \]

Finite Sets \[ e_1 \in x \land e_2 \in x \setminus e_1 \Rightarrow \exists y, z : \text{Set}(\alpha). |y| = |z| \land x = y \cup z \land y \neq \emptyset \]

Finite Relations \[ (x, y) \in r \land (y, z) \in r \Rightarrow (x, z) \in r \bowtie s \]
Simplest first-order theory with equality, applications of uninterpreted functions, and variables of uninterpreted sorts

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

- **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**: $\forall x, y : \sigma. x = y \Rightarrow f(x) = f(y)$

**Example**

\[
\begin{align*}
  f(f(f(a))) &= b \\
  g(f(a), b) &= a \\
  f(a) &\neq a
\end{align*}
\]
Operates over sorts $\text{Array}(\sigma_i, \sigma_e)$, $\sigma_i$, $\sigma_e$ and function symbols

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

For any index sort $\sigma_i$ and element sort $\sigma_e$

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

**Example**

\[ \text{store}(\text{store}(a, i, a[j]), j, a[i]) = \text{store}(\text{store}(a, j, a[i]), i, a[j]) \]
Restricted fragments, over the reals or the integers, support efficient methods:

- **Bounds**: $x \bowtie k$ with $\bowtie \in \{<, >, \le, \ge, =\}$ [BBC+05a]

- **Difference constraints**: $x - y \bowtie k$, with $\bowtie \in \{<, >, \le, \ge, =\}$ [NO05, WIGG05, CM06]

- **UTVPI**: $\pm x \pm y \bowtie k$, with $\bowtie \in \{<, >, \le, \ge, =\}$ [LM05]

- **Linear arithmetic**, e.g: $2x - 3y + 4z \le 5$ [DdM06a]

- **Non-linear arithmetic**, e.g: $2xy + 4xz^2 - 5y \le 10$ [BLNM+09, ZM10, JdM12]
Family of user-definable theories

Example

Color : red | green | blue
List(α) : nil | (head : α) :: (tail : List(α))

Distinctiveness: ∀h, t. nil ≠ h :: t
Exhaustiveness: ∀l. l = nil ∨ ∃h, t. h :: t
Injectivity: ∀h₁, h₂, t₁, t₂.

\[ h₁ :: t₁ = h₂ :: t₂ \Rightarrow h₁ = h₂ \wedge t₁ = t₂ \]

Selectors: ∀h, t. head(h :: t) = h \wedge tail(h :: t) = t
(Non-circularity: ∀l, x₁, ..., xₙ. l ≠ x₁ :: ... :: xₙ :: l)
OTHER INTERESTING THEORIES

- Strings and regular expressions \([\text{KGG}^+09, \text{LRT}^+14]\)
- Floating point arithmetic \([\text{BDG}^+14, \text{ZWR}14]\)
- Finite sets with cardinality \([\text{BRBT}16]\)
- Finite relations \([\text{MRTB}17]\)
- Transcendental Functions \([\text{GKC}13]\)
- Ordinary differential equations \([\text{GKC}13]\)
- ...
APPLICATIONS
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 $x$ of state vars
3. Encode system $M$ as $T$-formulas $(I[x], R[x, x'])$
   ∙ $I$ encodes $M$'s initial state condition
   ∙ $R$ encodes $M$'s transition relation
4. Encode $S$ as a $T$-formula $B[x]
5. Find a $k$ such that $I[x_0] \land R[x_0, x_1] \land \cdots \land R[x_{k-1}, x_k] \land B[x_k]$ is satisfiable in $T$
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 $x$ of state vars
3. Encode system $M$ as $T$-formulas $(I[x], R[x, x'])$
   where
   - $I$ encodes $M$’s initial state condition and
   - $R$ encodes $M$’s transition relation
4. Encode $S$ as a $T$-formula $B[x]$
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 $x$ of state vars
3. Encode system $M$ as $T$-formulas $(I[x], R[x, x'])$
   where
   - $I$ encodes $M$’s initial state condition and
   - $R$ encodes $M$’s transition relation
4. Encode $S$ as a $T$-formula $B[x]$
5. Find a $k$ such that $I[x_0] \land R[x_0, x_1] \land \cdots \land R[x_{k-1}, x_k] \land B[x_k]$ 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 $x$ of state vars

3. Encode system $M$ as $T$-formulas $(I[x], R[x, x'])$ where
   - $I$ encodes $M$’s initial state condition and
   - $R$ encodes $M$’s transition relation

4. Encode $S$ as a $T$-formula $P[x]$
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 $x$ of state vars

3. Encode system $M$ as $T$-formulas $(I[x], R[x, x'])$ where
   - $I$ encodes $M$’s initial state condition and
   - $R$ encodes $M$’s transition relation

4. Encode $S$ as a $T$-formula $P[x]$

5. Prove that $P[x]$ holds in all reachable states of $(I[x], R[x, x'])$
### Example (Parametric Resettable Counter)

**System**

**Vars**
- input pos int, \( n_0 \)
- input bool \( r \)
- int \( c, n \)

**Initialization**
- \( c := 1 \)
- \( n := n_0 \)

**Transitions**
- \( n' := n \)
- \( c' := \text{if} (r' \text{ or } c = n) \)
  - then 1
  - else \( c + 1 \)

**Property**
- \( c \leq n \)
Example (Parametric Resettable Counter)

**System**

**Vars**
- input pos int, \( n_0 \)
- input bool \( r \)
- int \( c, n \)

**Initialization**
- \( c := 1 \)
- \( n := n_0 \)

**Transitions**
- \( n' := n \)
- \( c' := \text{if } (r' \text{ or } c = n) \text{ then } 1 \text{ else } c + 1 \)

**Property**
- \( c \leq n \)

The transition relation contains infinitely many instances of the schema above, one for each \( n_0 > 0 \).
Example (Parametric Resettable Counter)

<table>
<thead>
<tr>
<th>System</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Vars</strong></td>
<td><strong>c &lt;= n</strong></td>
</tr>
<tr>
<td>input pos int, $n_0$</td>
<td>Encoding in $T = \text{LIA}$</td>
</tr>
<tr>
<td>input bool $r$</td>
<td>$x := (c, n, r, n_0)$</td>
</tr>
<tr>
<td>int $c$, $n$</td>
<td>$l[x] := c = 1$</td>
</tr>
<tr>
<td><strong>Initialization</strong></td>
<td>$\land n = n_0$</td>
</tr>
<tr>
<td>$c := 1$</td>
<td>$R[x, x'] := n' = n$</td>
</tr>
<tr>
<td>$n := n_0$</td>
<td>$\land (\neg r' \land c \neq n \lor c' = 1)$</td>
</tr>
<tr>
<td><strong>Transitions</strong></td>
<td>$\land (r' \lor c = n \lor c' = c + 1)$</td>
</tr>
<tr>
<td>$n' := n$</td>
<td>$P[x] := c \leq n$</td>
</tr>
<tr>
<td>$c' := \text{if } (r' \text{ or } c = n) \text{ then } 1$</td>
<td></td>
</tr>
<tr>
<td>else $c + 1$</td>
<td></td>
</tr>
</tbody>
</table>
$M = (l[x], r[x, x'])$
$M = (I[x], R[x, x'])$

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

(1) $I[x] \vdash_T P[x]$ (base case)

and

(2) $P[x] \land R[x, x'] \vdash_T P[x']$ (inductive step)
**INDUCTIVE REASONING**

\[ M = (I, R) \]

To prove \( P(X) \) invariant for \( M \) it suffices to show that it is inductive for \( M \), i.e.,

1. \( I(X) \models T P(X) \) (base case)

2. \( P(X) \wedge R(X, X') \models T P(X') \) (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 \((c, n, r)' = (5, 3, false)\)
Various approaches:

(1) $l[x] \models_{T} P[x]$ 

(2) $P[x] \wedge R[x, x'] \models_{T} P[x']$
(1) \( I[x] \models_T P[x] \)  

Various approaches:

**Strengthen** \( P \): find a property \( Q \) such that \( Q[x] \models_T P[x] \) and prove \( Q \) inductive  
(ex., interpolation-based MC, IC3, CHC)
STRENGTHENING INDUCTIVE REASONING

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

Various approaches:

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

**Strengthen** \( R \): find an auxiliary invariant \( Q[x] \) and use \( Q[x] \land R[x, x'] \land Q[x'] \) instead of \( R[x, x'] \)
(ex., Houdini, invariant sifting)
Various approaches:

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

**Strengthen $R$:** find an auxiliary invariant $Q[x]$ and use $Q[x] \land R[x, x'] \land Q[x']$ instead of $R[x, x']$  
(ex., Houdini, invariant sifting)

**Lengthen $R$:** Consider increasingly longer $R$-paths  
$R[x_0, x_1] \land \cdots \land R[x_{k-1}, x_k] \land R[x_k, x_{k+1}]$  
(ex., $k$-induction)
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 $a^* = \text{old}(b^*)$ and $b^* = \text{old}(a^*)$
Example

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

Check if the swap is correct:

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

$h_1 = \text{store}(h_0, a, h_0[a] + 32 \ h_0[b])$

$h_2 = \text{store}(h_1, b, h_1[a] - 32 \ h_1[b])$

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

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

```c
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)

```c
int BinarySearch (int [ ] a, int n, int k) {
    int l = 0;
    int h = n;
    while (l < h) {
        // Find middle value
        int m = l + (h - l) / 2;
        int v = a[m];
        if (k < v) { l = m + 1; }
        else if (v < k) { h = m; }
        else { return m; }
    }
    return -1;
}
```

Main approach
1. Compile source and annotations to program in Dijkstra’s core language:

```plaintext
S, T ::= x = t | havoc x | assert \( \varphi \) | assume \( \varphi \) | 
          S; T | S [] T
```

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

```plaintext
\( \wp(x = t, \varphi) = \varphi\{x \mapsto t\} \) \quad \wp(\text{assert } \psi, \varphi) = \psi \land \varphi
\( \wp(\text{assume } \psi, \varphi) = \psi \Rightarrow \varphi \) \quad \wp(\text{havoc } x, \varphi) = \forall x \varphi
\( \wp(S; T, \varphi) = \wp(S, \wp(T, \varphi)) \)
\( \wp(S [] T, \varphi) = \wp(S, \varphi) \land \wp(T, \varphi) \)
```

Example adapted from [dMB10]
Example (Binary Search)

pre = 0 ≤ n ≤ |a| ∧ ∀i : Int 0 ≤ i ∧ i ≤ n − 2 ⇒ a[i] ≤ a[i + 1]

post = (0 ≤ res ⇒ a[res] = k) ∧
     (res < 0 ⇒ ∀i : Int 0 ≤ i ∧ i ≤ n − 1 ⇒ a[i] ≠ k)

inv = 0 ≤ l ∧ l ≤ h ∧ h ≤ n ∧ n ≤ |a| ∧
     ∀i : Int 0 ≤ i ∧ i ≤ l − 1 ⇒ a[i] < k ∧
     ∀i : Int h ≤ i ∧ i ≤ n − 1 ⇒ a[i] > k
Example (Binary Search)

**pre** = \(0 \leq n \leq |a| \land \forall i : \text{Int} \ 0 \leq i \land i \leq n - 2 \Rightarrow a[i] \leq a[i + 1]\)

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

**inv** = \(0 \leq l \land l \leq h \land h \leq n \land n \leq |a| \land\)
\(\forall i : \text{Int} \ 0 \leq i \land i \leq l - 1 \Rightarrow a[i] < k \land\)
\(\forall i : \text{Int} \ h \leq i \land i \leq n - 1 \Rightarrow a[i] > k\)

\(\text{pre} \land \neg\textbf{let} \ l = 0, \ h = n \ \textbf{in} \ \text{inv} \land \forall : \text{Int} \ l, h. \ \text{inv} \Rightarrow\)
\(\ (\neg(l < h) \Rightarrow \text{post}\{\text{res} \mapsto -1}\}) \land\)
\(\ (l < h \Rightarrow \textbf{let} \ m = l + (h - l)/2, \ v = a[m] \ \textbf{in} \)
\(\ (k < v \Rightarrow \text{inv}\{l \mapsto m + 1\}) \land\)
\(\ (\neg(k < v) \land v < k \Rightarrow \text{inv}\{n \mapsto m\}) \land\)
\(\ (\neg(k < v) \land \neg(v < k) \Rightarrow \text{post}\{\text{res} \mapsto m\}))\)
Example (Binary Search)

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

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

\[\text{inv} = 0 \leq l \land l \leq h \land h \leq n \land n \leq |a| \land \]
\[\quad \forall i : \text{Int } 0 \leq i \land i \leq n \land i \leq |a| \land \]
\[\quad \forall i : \text{Int } h \leq i \land i \leq n \land i \leq |a| \land \]

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

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

$$\exists f. \forall x. P[f, x]$$
Formalization in second-order logic

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

$$\exists f. \forall x. P[f, x]$$

**Example**

Maximum of 2 values

$$P[f, x] = f(x) \geq x_1 \land f(x) \geq x_2 \land (f(x) = x_1 \lor f(x) = x_2)$$
REFUTATION-BASED SYNTHESIS

Formalization in second-order logic
- Let $P[f, x]$ be a property (specification) for a function $f$ over some variables $x = (x_1, x_2)$
- The synthesis problem is to determine the satisfiability of $\exists f. \forall x. P[f, x]$

Example

Maximum of 2 values

$P[f, x] = f(x) \geq x_1 \land f(x) \geq x_2 \land (f(x) = x_1 \lor f(x) = x_2)$

Problem: SMT only understands first-order logic
Single-invocation properties

- Every occurrence of $f$ is of the form $f(x)$
  - Previous example is single-invocation
  - Not single-invocation: $\forall x. f(x_1, x_2) = f(x_2, x_1)$

- When the synthesis property is single-invocation, it can be written as $\exists f. \forall x. P[f(x), x]$
Single-invocation properties

- Every occurrence of $f$ is of the form $f(x)$
- Previous example is single-invocation
- Not single-invocation: $\forall x. f(x_1, x_2) = f(x_2, x_1)$

When the synthesis property is single-invocation, it can written as $\exists f. \forall x. P[f(x), x]$

Note that:

$$\exists f. \forall x. P[f(x), x]$$ (1)

is equivalent to

$$\forall x. \exists y P[y, x]$$ (2)

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

\[ \forall x. \exists y \, P[y, x] \]
Proving the validity of

$$\forall x. \exists y \ P[y, x]$$

is equivalent to proving the unsatisfiability of

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

or the unsatisfiability of

$$\forall y. \neg P[y, c]$$

for some fresh constants $c$
How does an SMT solver show that

\[ \forall y \neg P[y, c] \] is unsatisfiable?
How does an SMT solver determine that

\[ \forall y \lnot P[y, c] \text{ is unsatisfiable?} \]

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

\[ \{ \lnot P[t_1[c], c], \lnot P[t_2[c], c], \ldots, \lnot P[t_n[c], c] \} \]
How does an SMT solver determine that

$$\forall y \neg P[y, c]$$
is unsatisfiable?

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

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

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

$$f = \lambda x. \text{ite}(P[t_1[x], x], t_1[x], (\cdots \text{ite}(P[t_{n-1}[x], x], t_{n-1}[x], t_n[x]) \cdots ))$$
Example
Schedule $n$ jobs, each composed of $m$ consecutive tasks, on $m$ machines.

Schedule in 8 time slots

<table>
<thead>
<tr>
<th>$d_{i,j}$</th>
<th>Mach. 1</th>
<th>Mach. 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job 1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Job 2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Job 3</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Example
Schedule \( n \) jobs, each composed of \( m \) consecutive tasks, on \( m \) machines.

Schedule in 8 time slots

<table>
<thead>
<tr>
<th>( d_{i,j} )</th>
<th>Mach. 1</th>
<th>Mach. 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job 1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Job 2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Job 3</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

\[ (t_{1,1} \geq 0) \land (t_{1,2} \geq t_{1,1} + 2) \land (t_{1,2} + 1 \leq 8) \]
\[ (t_{2,1} \geq 0) \land (t_{2,2} \geq t_{2,1} + 3) \land (t_{2,2} + 1 \leq 8) \]
\[ (t_{3,1} \geq 0) \land (t_{3,2} \geq t_{3,1} + 2) \land (t_{3,2} + 3 \leq 8) \]
\[ ((t_{1,1} \geq t_{2,1} + 3) \lor (t_{2,1} \geq t_{1,1} + 2)) \]
\[ ((t_{1,1} \geq t_{3,1} + 2) \lor (t_{3,1} \geq t_{1,1} + 2)) \]
\[ ((t_{2,1} \geq t_{3,1} + 2) \lor (t_{3,1} \geq t_{2,1} + 3)) \]
\[ ((t_{1,2} \geq t_{2,2} + 1) \lor (t_{2,2} \geq t_{1,2} + 1)) \]
\[ ((t_{1,2} \geq t_{3,2} + 3) \lor (t_{3,2} \geq t_{1,2} + 1)) \]
\[ ((t_{2,2} \geq t_{3,2} + 3) \lor (t_{3,2} \geq t_{2,2} + 1)) \]
Example
Schedule $n$ jobs, each composed of $m$ consecutive tasks, on $m$ machines.

Schedule in 8 time slots

<table>
<thead>
<tr>
<th>$d_{i,j}$</th>
<th>Mach. 1</th>
<th>Mach. 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job 1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Job 2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Job 3</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

SMT solver solution

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

$((t_{1,2} \geq t_{3,2} + 3) \lor (t_{3,2} \geq t_{1,2} + 1))$
$((t_{2,2} \geq t_{3,2} + 3) \lor (t_{3,2} \geq t_{2,2} + 1))$
H = 5 \text{ nm} \quad V = 1000 \text{ ft} \quad 0 \leq t \leq \frac{1}{20} \text{ h}

\left| T^z_1(t) - T^z_2(t) \right| \leq V

\left( T^x_1(t) - T^x_2(t) \right)^2 + \left( T^y_1(t) - T^y_2(t) \right)^2 \leq H^2

T^x_1(t) = 3.2484 + 270.7t + 433.12t^2 - 324.83999t^3

T^y_1(t) = 15.1592 + 108.28t + 121.2736t^2 - 649.67999t^3

T^z_1(t) = 38980.8 + 5414t - 21656t^2 + 32484t^3

T^x_2(t) = 1.0828 - 135.35t + 234.9676t^2 + 3248.4t^3

T^y_2(t) = 18.40759 - 230.6364t - 121.2736t^2 - 649.67999t^3

T^z_2(t) = 40280.15999 - 10828t + 24061.9816t^2 - 32484t^3

Example from [NM12]
$H = 5\,nm \quad V = 1000\,ft \quad 0 \leq t \leq \frac{1}{20}\,h$

\[
|T^z_1(t) - T^z_2(t)| \leq V
\]

\[
(T^x_1(t) - T^x_2(t))^2 + (T^y_1(t) - T^y_2(t))^2 \leq H^2
\]

$T^x_1(t) = 3.2164 \cdot 39000 - 236.71t + 422.41t^2 - 234.6989t^3$

$T^y_1(t) = 15.1592 + 108.28t + 121.2736t^2 - 649.67999t^3$

$T^z_1(t) = 38980.8 + 5414t - 21656t^2 + 324.63284t^3$

$T^x_2(t) = 1.0828 - 135.35t + 234.9676t^2 - 3248.4t^3$

$T^y_2(t) = 18.40759 - 230.6364t - 121.2736t^2 - 649.67999t^3$

$T^z_2(t) = 40280.15999 - 10828t + 24061.9816t^2 - 32484t^3$

Example from [NM12]


M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junntila, P. van Rossum, S. Schulz, and R. Sebastiani.
An incremental and layered procedure for the satisfiability of linear arithmetic logic.

Marco Bozzano, Roberto Bruttomesso, Alessandro Cimatti, Tommi A. Junntila, Peter van Rossum, Stephan Schulz, and Roberto Sebastiani.
The MathSAT 3 system.
Clark Barrett, Christopher Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli.

CVC4.

Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, and Roberto Sebastiani.

The MathSAT 4 SMT solver.

Martin Brain, Vijay D’Silva, Alberto Griggio, Leopold Haller, and Daniel Kroening.

Deciding floating-point logic with abstract conflict driven clause learning.

C. Borralleras, S. Lucas, R. Navarro-Marset, E. Rodríguez-Carbonell, and A. Rubio.

Solving Non-linear Polynomial Arithmetic via SAT Modulo Linear Arithmetic.


Clark Barrett, Igor Shikanian, and Cesare Tinelli. 
An abstract decision procedure for satisfiability in the theory of recursive data types. 

Clark Barrett and Cesare Tinelli. 
CVC3. 

Clark Barrett and Cesare Tinelli. 
Satisfiability modulo theories. 

A. Cimatti, A. Griggio, B. Schaafsma, and R. Sebastiani. 
The MathSAT5 SMT solver. 
S. Cotton and O. Maler.
Fast and Flexible Difference Constraint Propagation for DPLL(T).

Bruno Dutertre and Leonardo de Moura.
A Fast Linear-Arithmetic Solver for DPLL(T).

Bruno Dutertre and Leonardo de Moura.
The YICES SMT solver.

Leonardo De Moura and Nikolaj Bjørner.
Z3: an efficient smt solver.
L. de Moura and N. Bjørner.
Generalized, efficient array decision procedures.

Leonardo de Moura and Nikolaj Bjørner.
Bugs, moles and skeletons: Symbolic reasoning for software development.

Leonardo De Moura and Nikolaj Bjørner.
Satisfiability modulo theories: introduction and applications.

Bruno Dutertre.
Yices 2.2.
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

Yeting Ge and Leonardo de Moura.
Complete instantiation for quantified formulas in satisfiability modulo theories.
In Ahmed Bouajjani and Oded Maler, editors, Proceedings of the 21st
International Conference on Computer Aided Verification, volume 5643 of Lecture

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


Robert Nieuwenhuis and Albert Oliveras.  
Fast congruence closure and extensions.  

Andrew Reynolds and Jasmin Christian Blanchette.  
A decision procedure for (co)datatypes in SMT solvers.  

Andrew Reynolds, Morgan Deters, Viktor Kuncak, Cesare Tinelli, and Clark W. Barrett.  
Counterexample-guided quantifier instantiation for synthesis in SMT.  

Andrew Reynolds, Cesare Tinelli, Amit Goel, and Sava Krstić.  
Finite model finding in SMT.  
A Decision Procedure for an Extensional Theory of Arrays.  

Deciding Separation Logic Formulae by SAT and Incremental Negative Cycle Elimination.  

Harald Zankl and Aart Middeldorp.  
Satisfiability of Non-linear (Ir)rational Arithmetic.  

Aleksandar Zeljić, Christoph M Wintersteiger, and Philipp Rümmer.  
Approximations for model construction.  