SMT Solving for AI Planning: Theory, Tools and Applications

Erika Ábrahám
RWTH Aachen University, Germany

Francesco Leofante
RWTH Aachen University, Germany
University of Genoa, Italy

ICAPS 2018
Delft, The Netherlands
25 June 2018
About us

Erika
Full professor

Francesco
Ph.D. Student

Theory of Hybrid Systems @ RWTH
https://ths.rwth-aachen.de

Resources for this tutorial
https://ths.rwth-aachen.de/research/talks/smt4planning
What is this tutorial about?

What is satisfiability checking?

\[ \neg a \land b \lor c \]

\[ x^2 + x_2 \lor \sqrt{\varphi} \]
What is satisfiability checking?

How does SMT solving work?
What is this tutorial about?

Planning problem

What is satisfiability checking?

How does SMT solving work?

How to use it for planning?
Outline

SMT solving

I  Historical notes

II  SAT and SMT solving

III  Some applications outside planning

SMT solving for planning

IV  SMT and planning

V  Application: optimal planning with OMT

Concluding remarks
Outline

SMT solving

I Historical notes

II SAT and SMT solving

III Some applications outside planning

SMT solving for planning

IV SMT and planning

V Application: optimal planning with OMT

Concluding remarks
## The satisfiability problem

### Propositional logic

<table>
<thead>
<tr>
<th>Formula:</th>
<th>$(a \lor \neg b) \land (\neg a \lor b \lor c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Satisfying assignment:</td>
<td>$a = true, \quad b = false, \quad c = true$</td>
</tr>
</tbody>
</table>

It is perhaps the most well-known NP-complete problem [Cook, 1971].
The satisfiability problem

Propositional logic

Formula: \( (a \lor \neg b) \land (\neg a \lor b \lor c) \)

Satisfying assignment: \( a = \text{true}, \quad b = \text{false}, \quad c = \text{true} \)

It is perhaps the most well-known NP-complete problem [Cook, 1971].

Non-linear real algebra (NRA)

Formula: \( (x - 2y > 0 \lor x^2 - 2 = 0) \land x^4y + 2x^2 - 4 > 0 \)

Satisfying assignment: \( x = \sqrt{2}, \quad y = 2 \)

There are some hard problem classes... non-linear integer arithmetic is even undecidable.
"We have success stories of using zChaff to solve problems with more than one million variables and 10 million clauses. (Of course, it can't solve every such problem!)."
We have success stories of using zChaff to solve problems with more than one million variables and 10 million clauses. (Of course, it can't solve every such problem!)

The efficiency of our programs allowed us to solve over one hundred open quasigroup problems in design theory.
“We have success stories of using zChaff to solve problems with more than one million variables and 10 million clauses. (Of course, it can’t solve every such problem!).” [zch, ]
“We have success stories of using zChaff to solve problems with more than one million variables and 10 million clauses. (Of course, it can’t solve every such problem!).” [zch, ]

“The efficiency of our programs allowed us to solve over one hundred open quasigroup problems in design theory.” [SATO webpage, ]
Success story: SAT-solving

- Practical problems with millions of variables are solvable.
- A wide range of applications, e.g., verification, synthesis, combinatorial optimization, etc.
Success story: SAT-solving

- Practical problems with millions of variables are solvable.
- A wide range of applications, e.g., verification, synthesis, combinatorial optimization, etc.

Community support:

- Standard input language.
- Large benchmark library.
- Competitions since 2002.
  - 2017: 6 tracks, 28 solvers in the main track.
- SAT Live! forum as community platform, dedicated conferences, journals, etc.
An impression of the SAT solver development

Results of the SAT competition/race winners on the SAT 2009 application benchmarks, 20mn timeout

Source: The International SAT Solver Competitions [Järvisalo et al., 2012]
Computer algebra systems

SAT solvers

We have success stories of using zChaff to solve problems with more than one million variables and 10 million clauses. (Of course, it can’t solve every such problem!).
We have success stories of using zChaff to solve problems with more than one million variables and 10 million clauses. (Of course, it can't solve every such problem!).
Satisfiability modulo theories (SMT) solving:

- Propositional logic is sometimes too weak for modeling.
- Increase expressiveness: quantifier-free (QF) fragments of first-order logic over various theories.
Satisfiability modulo theories (SMT) solving:

- Propositional logic is sometimes too weak for modeling.
- Increase expressiveness: quantifier-free (QF) fragments of first-order logic over various theories.

Community support:

- **SMT-LIB**: standard input language since 2004.
- Large (~ 250,000) **benchmark library**.
- **Competitions** since 2005.
  - **2017**: 26 solvers in the main track.
SMT-LIB logics

Source: http://smtlib.cs.uiowa.edu/logics.shtml
SMT-LIB logics

Quantifier-free equality logic with uninterpreted functions

\[(a = c \land b = d) \rightarrow f(a, b) = f(c, d)\]

Source: http://smtlib.cs.uiowa.edu/logics.shtml
Quantifier-free bit-vector arithmetic

\[(a|b) \leq (a \& b)\]
SMT-LIB logics

Quantifier-free equality logic with uninterpreted functions

Quantifier-free bit-vector arithmetic

Quantifier-free array theory

Quantifier-free integer/rational difference logic

Quantifier-free integer/rational linear arithmetic

(Quantifier-free) real/integer linear arithmetic

(Quantifier-free) real/integer non-linear arithmetic

Combined theories

Source: http://smtlib.cs.uiowa.edu/logics.shtml

E. Ábrahám, F. Leofante

ICAPS’18

June 25, 2018
SMT-LIB logics

- Quantifier-free equality logic with uninterpreted functions
  \[(a = c \land b = d) \rightarrow f(a, b) = f(c, d)\]

- Quantifier-free bit-vector arithmetic
  \[(a | b) \leq (a \& b)\]

- Quantifier-free array theory
  \[i = j \rightarrow \text{read(write(a, i, v), j)} = v\]

- Quantifier-free integer/rational difference logic
  \[x - y \sim 0, \sim \in \{<, \leq, =, \geq, >\}\]

- (Quantifier-free) real/integer linear arithmetic
  \[3x + 7y = 8\]

- (Quantifier-free) real/integer non-linear arithmetic
  \[x^2 + 2xy + y^2 \geq 0\]

Combined theories

Source: [http://smtlib.cs.uiowa.edu/logics.shtml](http://smtlib.cs.uiowa.edu/logics.shtml)

E. Ábrahám, F. Leofante

ICAPS’18

June 25, 2018 13 / 74
SMT-LIB logics

Quantifier-free equality logic with uninterpreted functions

Quantifier-free bit-vector arithmetic

Quantifier-free array theory

Quantifier-free integer/rational difference logic

(Quantifier-free) real/integer linear arithmetic

$3x + 7y = 8$
SMT-LIB logics

(Quantifier-free) real/integer non-linear arithmetic

$$x^2 + 2xy + y^2 \geq 0$$
SMT-LIB logics

Quantifier-free equality logic with uninterpreted functions

\[(a = c \land b = d) \rightarrow f(a, b) = f(c, d)\]

Quantifier-free bit-vector arithmetic

\[(a | b) \leq (a \& b)\]

Quantifier-free array theory

\[i = j \rightarrow \text{read}(\text{write}(a, i, v), j) = v\]

Quantifier-free integer/rational difference logic

\[x - y \sim 0, \sim \in \{<, \leq, =, \geq, >\}\]

(Quantifier-free) real/integer linear arithmetic

\[3x + 7y = 8\]

(Quantifier-free) real/integer non-linear arithmetic

\[x^2 + 2xy + y^2 \geq 0\]

Combined theories

\[2f(x) + 5y > 0\]

Source: http://smtlib.cs.uiowa.edu/logics.shtml
Outline

SMT solving

I  Historical notes

II  SAT and SMT solving

III Some applications outside planning

SMT solving for planning

IV  SMT and planning

V  Application: optimal planning with OMT

Concluding remarks
Strategic combinations of decision procedures

+ = ?
Strategic combinations of decision procedures
Strategic combinations of decision procedures
Assumption: propositional logic formula in conjunctive normal form (CNF)

\[ c_1 : ( \neg a \lor b ) \land \]
\[ c_2 : ( \neg b \lor \neg c ) \land \]
\[ c_3 : ( \neg b \lor c ) \land \]

\ldots
Assumption: propositional logic formula in conjunctive normal form (CNF)

Ingredients: Enumeration

\[
c_1 : ( \neg a \lor b ) \land \\
c_2 : ( \neg b \lor \neg c ) \land \\
c_3 : ( \neg b \lor c ) \land \\
\ldots
\]
DPLL SAT solving with conflict-directed clause learning

Assumption: propositional logic formula in conjunctive normal form (CNF)

Ingredients: Enumeration

\[ c_1 : ( \neg a \lor b ) \land \]
\[ c_2 : ( \neg b \lor \neg c ) \land \]
\[ c_3 : ( \neg b \lor c ) \land \]

\[
\begin{array}{c}
\text{...}
\end{array}
\]
Assumption: propositional logic formula in conjunctive normal form (CNF)

Ingredients: Enumeration

\[
\begin{align*}
c_1 & : ( \neg a \lor b ) \land \\
c_2 & : ( \neg b \lor \neg c ) \land \\
c_3 & : ( \neg b \lor c ) \land \\
\end{align*}
\]

\[ \ldots \]
DPLL SAT solving with conflict-directed clause learning

Assumption: propositional logic formula in conjunctive normal form (CNF)

Ingredients: Enumeration

\[ c_1 : ( \neg a \lor b ) \land \]
\[ c_2 : ( \neg b \lor \neg c ) \land \]
\[ c_3 : ( \neg b \lor c ) \land \]

...
DPLL SAT solving with conflict-directed clause learning

Assumption: propositional logic formula in conjunctive normal form (CNF)

Ingredients: Enumeration

\[ c_1 : ( \neg a \lor b ) \land \]
\[ c_2 : ( \neg b \lor \neg c ) \land \]
\[ c_3 : ( \neg b \lor c ) \land \]

...
DPLL SAT solving with conflict-directed clause learning

Assumption: propositional logic formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation

\[ c_1 : ( \neg a \lor b ) \land \]
\[ c_2 : ( \neg b \lor \neg c ) \land \]
\[ c_3 : ( \neg b \lor c ) \land \]

...
DPLL SAT solving with conflict-directed clause learning

Assumption: propositional logic formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation

\[ c_1 : ( \neg a \lor b ) \land \]
\[ c_2 : ( \neg b \lor \neg c ) \land \]
\[ c_3 : ( \neg b \lor c ) \land \]

...
DPLL SAT solving with conflict-directed clause learning

Assumption: propositional logic formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation

\[ c_1 : ( \neg a \lor b ) \land \]
\[ c_2 : ( \neg b \lor \neg c ) \land \]
\[ c_3 : ( \neg b \lor c ) \land \]

\[ \ldots \]
DPLL SAT solving with conflict-directed clause learning

Assumption: propositional logic formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation

\[ c_1 : ( \neg a \lor b ) \land \]
\[ c_2 : ( \neg b \lor \neg c ) \land \]
\[ c_3 : ( \neg b \lor c ) \land \]

...
DPLL SAT solving with conflict-directed clause learning

Assumption: propositional logic formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation + Resolution

\[ c_1 : ( \neg a \lor b ) \land \]
\[ c_2 : ( \neg b \lor \neg c ) \land \]
\[ c_3 : ( \neg b \lor c ) \land \]

\[ \ldots \]
Assumption: propositional logic formula in conjunctive normal form (CNF)
Resolution

Assumption: propositional logic formula in conjunctive normal form (CNF)

Derivation rule form:

\[
\begin{array}{c}
\text{antecedent}_1 \quad \ldots \quad \text{antecedent}_n \\
\text{consequent}
\end{array}
\]

\text{Rule\_name}
Assumption: propositional logic formula in conjunctive normal form (CNF)

Derivation rule form:

\[
\frac{\text{antecedent}_1 \ldots \text{antecedent}_n \quad \text{consequent}}{\text{Rule\_name}}
\]

\[
\frac{(l_1 \lor \ldots \lor l_n \lor x) \quad (l'_1 \lor \ldots \lor l'_m \lor \neg x)}{(l_1 \lor \ldots \lor l_n \lor l'_1 \lor \ldots \lor l'_m)} \quad \text{Rule\_res}
\]
Resolution

Assumption: propositional logic formula in conjunctive normal form (CNF)

Derivation rule form:

\[
\begin{array}{c}
\text{antecedent}_1 \quad \ldots \quad \text{antecedent}_n \\
\text{consequent}
\end{array}
\]

\text{Rule\_name}

\[
\frac{(l_1 \lor \ldots \lor l_n \lor x) \quad (l'_1 \lor \ldots \lor l'_m \lor \neg x)}{(l_1 \lor \ldots \lor l_n \lor l'_1 \lor \ldots \lor l'_m)}
\]

\text{Rule\_res}

\exists x. C_x \land C_{\neg x} \land C \iff \text{Resolvents} (C_x, C_{\neg x}) \land C
DPLL SAT solving with conflict-directed clause learning

Assumption: propositional logic formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation + Resolution

\[ c_1 : (\neg a \lor b) \land \]
\[ c_2 : (\neg b \lor \neg c) \land \]
\[ c_3 : (\neg b \lor c) \land \]

\ldots
DPLL SAT solving with conflict-directed clause learning

Assumption: propositional logic formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation + Resolution

\[
c_1 : ( \neg a \lor b ) \land \\
c_2 : ( \neg b \lor \neg c ) \land \\
c_3 : ( \neg b \lor c ) \land \\
\ldots
\]

Resolution

\[
c_3 : (\neg b \lor c) \quad c_2 : (\neg b \lor \neg c) \\
\hline \\
c_4 : (\neg b)
\]
DPLL SAT solving with conflict-directed clause learning

Assumption: propositional logic formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation + Resolution

\[ c_1 : (\neg a \lor b) \land \]
\[ c_2 : (\neg b \lor \neg c) \land \]
\[ c_3 : (\neg b \lor c) \land \]
\[ c_4 : (\neg b) \land \]

\[ \frac{c_3 : (\neg b \lor c)}{c_2 : (\neg b \lor \neg c)} \]
\[ \frac{c_2 : (\neg b \lor \neg c)}{c_4 : (\neg b)} \]

Resolution
(Full/less) lazy SMT solving

- quantifier-free FO formula $\varphi$
- propositional logic formula in CNF $\varphi'$
- SAT solver
- Theory solver(s)
- SAT or UNSAT
- SAT or UNSAT + lemmas

Boolean abstraction
Tseitin’s transformation
Less lazy SMT solving
\[(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)\]
(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)

\downarrow

(a \lor b) \land (c \lor d)
Less lazy SMT solving

\[(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)\]

\[(a \lor b) \land (c \lor d)\]

SAT solver

Theory solver(s)
Less lazy SMT solving

\[(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)\]

\[
(a \lor b) \land (c \lor d)
\]

SAT solver

Theory solver(s)

\[\neg a\]
Less lazy SMT solving

\[(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)\]

\[(a \lor b) \land (c \lor d)\]

\[\neg a, b\]
(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)

\downarrow

(a \lor b) \land (c \lor d)

---SAT solver---

\neg a, \ b

---Theory solver(s)---

x \geq 0, \ x > 2
(x < 0 ∨ x > 2) ∧ (x^2 = 1 ∨ x^2 < 0)

SAT solver

¬a, b

x ≥ 0, x > 2

Theory solver(s)
(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)

(a \lor b) \land (c \lor d)

\neg a, b, \neg c

x \geq 0, x > 2
Less lazy SMT solving

\[(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)\]

\[(a \lor b) \land (c \lor d)\]

\(x \geq 0, x > 2\)

SAT solver

\(-a, b, -c, d\)

Theory solver(s)
Less lazy SMT solving

\[(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)\]

\[(a \lor b) \land (c \lor d)\]

$x \geq 0, x > 2, x^2 \neq 1, x^2 < 0$
(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)

\(\neg a, b, \neg c, d\)

SAT solver

Theory solver(s)

\(x \geq 0, x > 2, x^2 \neq 1, x^2 < 0\)

UNSAT: \(\neg (x^2 < 0)\)
Less lazy SMT solving

\[ (x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0) \]

\[ \downarrow \]

\[ (a \lor b) \land (c \lor d) \land (\neg d) \]

\[ \rightarrow \text{SAT solver} \rightarrow \neg a, b, \neg c, d \]

\[ x \geq 0, x > 2, x^2 \neq 1, x^2 < 0 \]

\[ \text{UNSAT: } \neg(x^2 < 0) \]

\[ \rightarrow \text{Theory solver(s)} \rightarrow \]
Less lazy SMT solving

\((x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)\)

\((a \lor b) \land (c \lor d) \land (\neg d)\)

SAT solver

Theory solver(s)
Less lazy SMT solving

\[(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)\]

\[\downarrow\]

\[(a \lor b) \land (c \lor d) \land (\neg d)\]

\[\downarrow\]

SAT solver

\[\neg d\]

Theory solver(s)
Less lazy SMT solving

\[(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)\]

\[(a \lor b) \land (c \lor d) \land (\neg d)\]

SAT solver

\[\neg d, c\]

Theory solver(s)
Less lazy SMT solving

\[(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)\]

\[(a \lor b) \land (c \lor d) \land (\neg d)\]

\[x^2 \geq 0, \ x^2 = 1\]
Less lazy SMT solving

\[(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)\]

\[
\downarrow
\]

\[
(a \lor b) \land (c \lor d) \land (\neg d)
\]

SAT solver

\[
\neg d, c
\]

\[
x^2 \geq 0, x^2 = 1
\]

SAT

Theory solver(s)
Less lazy SMT solving

\[(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)\]

\[
\left( a \lor b \right) \land \left( c \lor d \right) \land \neg d
\]

\[x^2 \geq 0, \ x^2 = 1\]
Model constructing satisfiability calculus (MCSAT)

B-decision
B-propagation
B-conflict resolution
Model constructing satisfiability calculus (MCSAT)

- \( \mathbb{B} \)-decision
- \( \mathbb{B} \)-propagation
- \( \mathbb{B} \)-conflict resolution

- \( \mathbb{T} \)-decision
- \( \mathbb{T} \)-propagation
- \( \mathbb{T} \)-conflict resolution
Optimization modulo theories (full lazy case)

\[ \varphi + \text{objective } f \]

Boolean abstraction
Tseitin’s transformation

quantifier-free FO formula

propositional logic formula in CNF

SAT solver

(SAT + \( \mu_{opt} \)) or UNSAT

theory constraints + \( f \)

SAT + \( \mu_{opt} \):
\[ \varphi := \varphi \land f \sim \mu_{opt} \]

or

UNSAT + lemmas

Theory solver(s)
Some theory solver candidates for arithmetic theories

Linear real arithmetic:
- Simplex
- Ellipsoid method
- Fourier-Motzkin variable elimination (mostly preprocessing)
- Interval constraint propagation (incomplete)

Linear integer arithmetic:
- Cutting planes, Gomory cuts
- Branch-and-bound (incomplete)
- Bit-blasting (eager)
- Interval constraint propagation (incomplete)

Non-linear real arithmetic:
- Cylindrical algebraic decomposition
- Gröbner bases (mostly preprocessing/simplification)
- Virtual substitution (focus on low degrees)
- Interval constraint propagation (incomplete)

Non-linear integer arithmetic:
- Generalised branch-and-bound (incomplete)
- Bit-blasting (eager, incomplete)
- Interval constraint propagation (incomplete)
Can we simply plug in available implementations of such methods as theory solvers into an SMT solver?
Problem solved?

Can we simply plug in available implementations of such methods as theory solvers into an SMT solver?

Theory solvers should be **SMT-compliant**, i.e., they should

- work **incrementally**,
- generate **lemmas** explaining inconsistencies, and
- be able to **backtrack**.
Can we simply plug in available implementations of such methods as theory solvers into an SMT solver?

Theory solvers should be **SMT-compliant**, i.e., they should
- work **incrementally**,
- generate **lemmas** explaining inconsistencies, and
- be able to **backtrack**.

Originally, the mentioned methods are **not SMT-compliant**.

SMT-adaptations can be tricky, but can lead to beautiful novel algorithms.
Satisfiability checking and symbolic computation
Bridging two communities to solve real problems

http://www.sc-square.org/CSA/welcome.html

SC²
Satisfiability Checking and Symbolic Computation

SUMMARY

This project is funded (subject to contract) as project H2020-FETOPEN-2015-CSA_712689 of the European Union. It is the start of the general push to create a real SC² community.

Background

The use of advanced methods to solve practical and industrially relevant problems by computers has a long history. Whereas Symbolic Computation is concerned with the algorithmic determination of exact solutions to complex mathematical problems, more recent developments in the area of Satisfiability Checking tackle similar problems but with different algorithmic and technological solutions. Though both communities have made remarkable progress in the last decades, they still need to be strengthened to tackle practical problems of rapidly increasing size and complexity. Their separate tools (computer algebra systems and SMT solvers) are urgently needed to examine prevailing problems with a direct effect to our society. For example, Satisfiability Checking is an essential backend for assuring the security and the safety of computer systems. In various scientific areas, Symbolic Computation enables dealing with large mathematical problems out of reach of pencil and paper developments. Currently the two communities are largely disjoint and unaware of the achievements of each other, despite strong reasons for them to discuss and collaborate, as they share many central interests. However, researchers from these two communities rarely interact, and also their tools lack common, mutual interfaces for unifying their strengths. Bridges between the communities in the form of common platforms and roadmaps are necessary to initiate an exchange, and to support and to direct their interaction. These are the main objectives of this CSA. We will initiate a wide range of activities to bring the two communities together, identify common challenges, offer global events and bilateral visits, propose standards, and so on. We believe that these activities will
Some popular SMT solvers (incomplete!)

- **AProVE (RWTH Aachen University, Germany)** [Giesl et al., 2004]
- **CVC4 (New York and Iowa, USA)** [Deters et al., 2014]
- **MathSAT 5 (FBK, Italy)** [Cimatti et al., 2013]
- **MiniSmt (University of Innsbruck, Austria)** [Zankl and Middeldorp, 2010]
- **Boolector (JKU, Austria)** [Niemetz et al., 2014]
- **SMT-RAT (RWTH Aachen University, Germany)** [Corzilius et al., 2012]
- **Z3 (NYU, Microsoft Research, USA)** [de Moura and Bjørner, 2008]
- **Yices 2 (SRI International, USA)** [Dutertre, 2014]
- ...
Our SMT-RAT library [Corzilius et al., 2012, Corzilius et al., 2015]

- MIT licensed source code: github.com/smtrat/smtrat
- Documentation: smtrat.github.io
Strategic composition of solver modules in SMT-RAT

- Strategy: directed graph over modules with guarded edges
- Guard: may talk about the formula forwarded to backends
- Backend-calls: passed to all enabled successors $\rightarrow$ parallelism
## SMT-RAT modules

### Module

**Implements**

- `add(Formula)`
- `remove(Formula)`
- `check()`
- `updateModel()`
## SMT-RAT modules

<table>
<thead>
<tr>
<th>Module</th>
<th>Implements</th>
<th>check() may</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>add(Formula)</td>
<td>forward (sub-)problems to backend modules</td>
</tr>
<tr>
<td></td>
<td>remove(Formula)</td>
<td>return sat or unsat</td>
</tr>
<tr>
<td></td>
<td>check()</td>
<td>return a lemma or split</td>
</tr>
<tr>
<td></td>
<td>updateModel()</td>
<td>return unknown</td>
</tr>
</tbody>
</table>
### Solver modules in SMT-RAT  
[Corzilius et al., 2012, Corzilius et al., 2015]

**CArL library** for basic arithmetic datatypes and computations  
[NFM’11, CAI’11, Sapientia’18]

#### Basic modules
- SAT solver
- CNF converter
- Preprocessing/simplifying modules

#### Non-algebraic decision procedures
- Bit-vectors
- Bit-blasting
- Equalities and uninterpreted functions
- Pseudo-Boolean formulas
- Interval constraint propagation

#### Algebraic decision procedures
- Fourier-Motzkin variable elimination
- Simplex
- Subtropical satisfiability
- Gröbner bases  
  - [CAI’13]
- MCSAT (FM, VS, CAD)
- Cylindrical algebraic decomposition  
  - [CADE-24, SC²’17, PhD Loup, PhD Kremer]
- Virtual substitution  
  - [FCT’11, SC²’17, PhD Corzilius]
- Generalized branch-and-bound  
  - [CASC’16]
- Cube tests
class myStrategy: public Manager {
    myStrategy(): Manager() {
        setStrategy(
            addBackend<SATModule<SATSettings>>(
                addBackend<CADModule<CADSettings>>()
            )
        );
    }
};
SMT-RAT strategies

Preprocessing

SAT

nonlinear real

CAD
SMT-RAT strategies

- Preprocessing
- Bit-blasting
- SAT
  - nonlinear real
- CAD
SMT-RAT strategies

Preprocessing

Bit-blasting

SAT

nonlinear real

VS

CAD
SMT-RAT strategies

Preprocessing

Bit-blasting

SAT

nonlinear real

ICP

VS

CAD
SMT-RAT strategies

```
class myStrategy: public Manager
{
    myStrategy(): Manager()
    {
        setStrategy(addBackend<SATModule<SATSettings>>(
            addBackend<CADModule<CADSettings>>()
        ));
    }
};
```
SMT-RAT strategies

Preprocessing

Bit-blasting

SAT

Branch and bound

Simplex

Linear integer

ICP

Linear real

Simplex

Linear integer

VS

Nonlinear real

CAD
1. Download and build CArL & SMT-RAT
   http://smtrat.github.io/carl/getting_started.html

2. Optionally: Extend it with custom modules and strategies

3. Select a strategy
   $ cmake -D SMTRAT_Strategy=CADOnly ..

4. Build SMT-RAT
   $ make smtrat

5. Run it
   $ ./smtrat input.smt2
Outline

SMT solving

I Historical notes

II SAT and SMT solving

III Some applications outside planning

SMT solving for planning

IV SMT and planning

V Application: optimal planning with OMT

Concluding remarks
SMT applications

- model checking
- termination analysis
- runtime verification
- test case generation
- controller synthesis
- predicate abstraction
- equivalence checking
- scheduling
- planning
- deployment optimisation on the cloud
- product design automation
- ...
Embedding SAT/SMT solvers

Environment

Software engine

Problem

Logical problem specification

SAT/SMT solver

Solution
Embedding SAT/SMT solvers

Environment

Software engine

Problem

Logical problem specification

SAT/SMT solver

Solution

Encoding: SAT/SMT-LIB standard
elaborate encoding is extremely important!
Embedding SAT/SMT solvers

Environment

Software engine

Problem

Logical problem specification

SAT/SMT solver

Solution

Encoding: SAT/SMT-LIB standard
elaborate encoding is extremely important!

standard input syntax → free solver choice
Embedding SAT/SMT solvers

Encoding: SAT/SMT-LIB standard
elaborate encoding is extremely important!

standard input syntax → free solver choice

In the following: applications of SMT solvers
Bounded model checking for C/C++ [Kroening and Tautschnig, 2014]

CBMC is a Bounded Model Checker for C and C++ programs. It supports C89, C99, most of C11 and most compiler extensions provided by gcc and Visual Studio. It also supports SystemC using Scoot. We have recently added experimental support for Java Bytecode.

CBMC verifies array bounds (buffer overflows), pointer safety, exceptions and user-specified assertions. Furthermore, it can check C and C++ for consistency with other languages, such as Verilog. The verification is performed by unwinding the loops in the program and passing the resulting equation to a decision procedure.

While CBMC is aimed for embedded software, it also supports dynamic memory allocation using malloc and new. For questions about CBMC, contact Daniel Kroening.

CBMC is available for most flavours of Linux (pre-packaged on Debian, Ubuntu and Fedora), Solaris 11, Windows and MacOS X. You should also read the CBMC license.

CBMC comes with a built-in solver for bit-vector formulas that is based on MiniSat. As an alternative, CBMC has featured support for external SMT solvers since version 3.3. The solvers we recommend are (in no particular order) Boolector, MathSAT, Yices 2 and Z3. Note that these solvers need to be installed separately and have different licensing conditions.

Logical encoding of finite paths

CBMC is a Bounded Model Checker for C and C++ programs. It supports C89, C99, most of C11 and most compiler extensions provided by gcc and Visual Studio. It also supports SystemC using Scoot. We have recently added experimental support for Java Bytecode.

CBMC verifies array bounds (buffer overflows), pointer safety, exceptions and user-specified assertions. Furthermore, it can check C and C++ for consistency with other languages, such as Verilog. The verification is performed by unwinding the loops in the program and passing the resulting equation to a decision procedure.

While CBMC is aimed for embedded software, it also supports dynamic memory allocation using malloc and new. For questions about CBMC, contact Daniel Kroening.

CBMC is available for most flavours of Linux (pre-packaged on Debian, Ubuntu and Fedora), Solaris 11, Windows and MacOS X. You should also read the CBMC license.

CBMC comes with a built-in solver for bit-vector formulas that is based on MiniSat. As an alternative, CBMC has featured support for external SMT solvers since version 3.3. The solvers we recommend are (in no particular order) Boolector, MathSAT, Yices 2 and Z3. Note that these solvers need to be installed separately and have different licensing conditions.

Logical encoding of finite paths

Encoding idea: $Init(s_0) \land Trans(s_0, s_1) \land \ldots \land Trans(s_{k-1}, s_k) \land Bad(s_0, \ldots, s_k)$

Logical encoding of finite paths

**Encoding idea:** $\text{Init}(s_0) \land \text{Trans}(s_0, s_1) \land \ldots \land \text{Trans}(s_{k-1}, s_k) \land \text{Bad}(s_0, \ldots, s_k)$

**Application examples:**
- Error localisation and explanation
- Equivalence checking
- Test case generation
- Worst-case execution time

Source: D. Kroening. [CBMC home page](http://www.cprover.org/cbmc/)
Hybrid systems reachability analysis [Kong et al., 2015]

**dReach** is a tool for safety verification of hybrid systems.

It answers questions of the type: Can a hybrid system run into an unsafe region of its state space? This question can be encoded to SMT formulas, and answered by our SMT solver. **dReach** is able to handle general hybrid systems with nonlinear differential equations and complex discrete mode-changes.


**dReach home page.** [http://dreal.github.io/dReach/](http://dreal.github.io/dReach/)
Termination analysis for programs [Ströder et al., 2015]


**AProVE: Termination and memory safety of C programs (competition contribution).**

In Proc. TACAS’15.
Termination analysis for programs [Ströder et al., 2015]

AProve: Termination and memory safety of C programs (competition contribution).

In Proc. TACAS’15.
Termination analysis for programs [Ströder et al., 2015]

Term rewrite system

Automated Program Verification Environment

Term rewrite system

Dependency pairs

Chains

Logical encoding for well-founded orders.


AProve: Termination and memory safety of C programs (competition contribution).

In Proc. TACAS’15.
JUnit<sub>RV</sub>: Runtime verification of multi-threaded, object-oriented systems [Decker et al., 2016]

**Properties:** linear temporal logics enriched with first-order theories
**Method:** SMT solving + classical monitoring

![Diagram of monitoring approach](image)

**Fig. 1** Schematic overview of the monitoring approach


**Monitoring modulo theories.**

Figure 1: An example of RCPSP (Liess and Michelon 2008)

Source: C. Ansótegui, M. Bofill, M. Palahí, J. Suy, M. Villaret.

**Satisfiability modulo theories: An efficient approach for the resource-constrained project scheduling problem.**

Proc. of SARA’11.
Deployment optimisation on the cloud [Ábrahám et al., 2016]

Source: E. Ábrahám, F. Corzilius, E. Broch Johnsen, G. Kremer, J. Mauro.

Zephyrus2: On the fly deployment optimization using SMT and CP technologies.
SETTA’16.
Parameter synthesis for probabilistic systems [Dehnert et al., 2015]


PROPhESY: A probabilistic parameter synthesis tool.

In Proc. of CAV’15.
Outline

SMT solving

I  Historical notes

II  SAT and SMT solving

III  Some applications outside planning

SMT solving for planning

IV  SMT and planning

V  Application: optimal planning with OMT

Concluding remarks
From planning to satisfiability checking

Classical planning
Classical planning

restrict search for a plan to paths with (predetermined) bound
Classical planning

- restrict search for a plan to paths with (predetermined) bound

Reductions of planning to SAT
- linear encodings [Kautz and Selman, 1992]
From planning to satisfiability checking

Original work by Kautz and Selman was later extended with, e.g.,

- parallel plans [Kautz et al., 1996, Rintanen et al., 2006]
- metric constraints [Wolfman and Weld, 1999]
- non-deterministic domains [Giunchiglia, 2000]
- time constraints [Shin and Davis, 2005] (when SMT was not yet known as such)
- preferences [Giunchiglia and Maratea, 2007]

* Have a look at, e.g., [Rintanen, 2009] for more on planning and SAT.
From planning to satisfiability checking

Original work by Kautz and Selman was later extended with, e.g.,

- parallel plans [Kautz et al., 1996, Rintanen et al., 2006]
- metric constraints [Wolfman and Weld, 1999]
- non-deterministic domains [Giunchiglia, 2000]
- time constraints [Shin and Davis, 2005] (when SMT was not yet known as such)
- preferences [Giunchiglia and Maratea, 2007]

Then SMT came... and new solutions followed, e.g.,

- numeric planning [Scala et al., 2016]
- temporal planning [Rintanen, 2015, Rintanen, 2017]
- planning in hybrid domains [Cashmore et al., 2016]
- optimal temporal planning (with OMT) [Leofante et al., 2018]

*Have a look at, e.g., [Rintanen, 2009] for more on planning and SAT.
Planning problem

Let $F$ and $A$ be the sets of *fluents* and *actions*.

Let $X = F \cup A$ and $X' = \{ x' : x \in X \}$ be its *next state* copy.

A planning problem is a triple of boolean formulae $\Pi = \langle I, T, G \rangle$ where

- $I(F)$ represents the set of *initial* states
- $T(X, X')$ describes how actions *affect* states
- $G(F)$ represents the set of *goal* states
For a given bound $k \in \mathbb{N}$, let $X_n = \{x_n : x \in X\}, \ n = 0, ..., k$.

Furthermore, let

- $I(X_n)$ (resp. $G(X_n)$) be the formula obtained from $I$ (resp. $G$) by replacing each $x \in X$ with the corresponding $x_n \in X_n$
- $T(X_n, X_{n+1})$ be the formula obtained from $T$ by replacing each $x \in X$ (resp. $x' \in X'$) with the corresponding $x_n \in X_n$ (resp. $x_{n+1} \in X_{n+1}$).
Encoding Π in SAT - renaming

For a given bound $k \in \mathbb{N}$, let $X_n = \{x_n : x \in X\}, n = 0, ..., k$.

Furthermore, let

- $I(X_n)$ (resp. $G(X_n)$) be the formula obtained from $I$ (resp. $G$) by replacing each $x \in X$ with the corresponding $x_n \in X_n$.

- $T(X_n, X_{n+1})$ be the formula obtained from $T$ by replacing each $x \in X$ (resp. $x' \in X'$) with the corresponding $x_n \in X_n$ (resp. $x_{n+1} \in X_{n+1}$).

Encoding Π in SAT - the formula

The planning problem $\Pi$ with makespan $k$ is the formula

$$\phi(\Pi, k) := I(X_0) \land \bigwedge_{i=0}^{k-1} T(X_i, X_{i+1}) \land G(X_k)$$
Encoding $\Pi$ in SAT

- $\varphi_{(\Pi,k)}$ is sat iff there exists a plan with length $k$
  - ✔ in that case, a plan can be extracted from the satisfying assignment
- in parallel encodings, two actions can be executed in parallel if they are non-mutex
- optimal plans minimize the number of steps:
  - ⏫ start with $k = 1$
  - ⊗ increase until $\varphi_{(\Pi,k)}$ becomes sat or upper bound on $k$ is reached.
We will now consider a simplified planning problem and show how it can be encoded as an SMT formula.
We will now consider a simplified planning problem and show how it can be encoded as an SMT formula.

What we will see:
- Brief problem description (also comes in PDDL 😊)
- SMT-LIB standard (basic syntax, advance features)
Encoding planning problems: how to?

We will now consider a simplified planning problem and show how it can be encoded as an SMT formula.

What we will see:

- Brief problem description (also comes in PDDL 😊)
- SMT-LIB standard (basic syntax, advance features)

Disclaimer

Planning problems can be encoded in many different (± efficient) ways. Given the introductory nature of this tutorial, we will use naive encodings only to introduce functionalities of SMT solvers.
Problem statement

- Set of locations: \( \ell_1, \ell_2 \)
- All locations must be visited
- Each location must be visited at most once

Simplifying assumptions:

- Graph fully connected, undirected, unweighted (weights kick in later)
Working example: TSP

PDDL Domain

```prolog
(define (domain tsp)
 (:requirements :negative-preconditions)
 (:predicates (at ?x) (visited ?x))
 (:action move
  :parameters (?x ?y)
  :precondition (and (at ?x) (not (visited ?y)))
  :effect (and (at ?y) (visited ?y) (not (at ?x))))
)
```

PDDL Problem

```prolog
(define (problem tsp-2)
 (:domain tsp)
 (:objects l1 l2 )
 (:init
  (at l1))
 (:goal
  (and (visited l1) (visited l2))))
```
SMT-LIB standard

Syntax of core theory

:sorts ((Bool 0))
:funs (  
  (true Bool)
  (false Bool)
  (not Bool Bool)
  (and Bool Bool Bool :left-assoc)
  ...
  (par (A) (= A A Bool :chainable))
  (par (A) (ite Bool A A A))
  ...

Syntax of arithmetic theories

:sorts ((Real 0))
:funs ( 
  ...
  (+ Real Real Real :left-assoc)
  (* Real Real Real :left-assoc)
  ...
  (< Real Real Bool :chainable)
  ...
)
Check the following link for more:
http://smtlib.cs.uiowa.edu/index.shtml
Propositional encoding - I

; SAT encoding for TSP
; benchmark generated from python API

; Declare variables

(declare-fun visited_1_0 () Bool)
(declare-fun visited_2_0 () Bool)
(declare-fun visited_1_1 () Bool)
(declare-fun visited_2_1 () Bool)

(declare-fun at_1_0 () Bool)
(declare-fun at_2_0 () Bool)
(declare-fun at_1_1 () Bool)
(declare-fun at_2_1 () Bool)
Boolean example

Propositional encoding - II

; Assert formula for initial state

(assert (and at_1_0 visited_1_0 (not visited_2_0) (not at_2_0)))

; Assert formula encoding unrolling of transition relation

(assert
(let (($x1 (and (and at_1_0 (not visited_2_0) at_2_1 visited_2_1 (not at_1_1)) (and (= visited_1_1 visited_1_0 )))))
(let (($x2 ... )))
(or $x1 $x2)))

; Assert formula for goal states

(assert (and visited_1_1 visited_2_1))
Boolean example

Propositional encoding - III

; Assert additional conditions
(assert (=> at_1_0 (not at_2_0)))
(assert (=> at_2_0 (not at_1_0)))
...

; Check whether the formula is satisfiable
(check-sat)

; If sat, retrieve model
(get-value ( at_1_0 visited_1_0 at_2_0 visited_2_0 at_1_1
visited_1_1 at_2_1 visited_2_1))

; (get-model) to retrieve the complete model
; Solver returns

sat
((at_1_0 true)
(visited_1_0 true)
(at_2_0 false)
(visited_2_0 false)
(at_1_1 false)
(visited_1_1 true)
(at_2_1 true)
(visited_2_1 true))
Unsat cores - intuition

Let’s assume our input formula $\varphi$ is unsat...we would like to know why!

*More here: [Liffiton and Sakallah, 2008]
Let's assume our input formula $\varphi$ is unsat...we would like to know why!

Recall: $\varphi$ is CNF

$$\varphi := \bigwedge_{i=1}^{n} C_i \quad \text{with} \quad C_i := \bigvee_{j=1}^{k_i} a_{ij}$$

*More here: [Liffiton and Sakallah, 2008]*
Let’s assume our input formula $\varphi$ is unsat...we would like to know why!

Recall: $\varphi$ is CNF

$$\varphi := \bigwedge_{i=1}^{n} C_i \quad \text{with} \quad C_i := \bigvee_{j=1}^{k_i} a_{ij}$$

Transform formula adding \textit{clause-selector} variables

$$C'_i := (\neg y_i \lor C_i) \quad \forall i = 1, \ldots, n$$

*More here: [Lifftiton and Sakallah, 2008]
Let’s assume our input formula \( \varphi \) is unsat...we would like to know why!

Recall: \( \varphi \) is CNF

\[
\varphi := \bigwedge_{i=1}^{n} C_i \quad \text{with} \quad C_i := \bigvee_{j=1}^{k_i} a_{ij}
\]

Transform formula adding *clause-selector* variables

\[
C'_i := (\neg y_i \lor C_i) \quad \forall i = 1, \ldots, n
\]

We can now enable and disable constraints by playing with \( y_i \)

👉 check satisfiability of subsets of original constraints

*More here: [Liffiton and Sakallah, 2008]*
Unsat cores

Propositional encoding - I

; SAT encoding for TSP
; benchmark generated from python API
(set-info :status unsat)

; Enable unsat core generation
(set-option :produce-unsat-cores true)

; Declare variables
...

; Assert formula for initial state

(assert (! at_1_0 :named I1))
(assert (! visited_1_0 :named I2))
(assert (! (not visited_2_0) :named I3))
(assert (! (not at_2_0) :named I4))
Propositional encoding - II

; Assert formula encoding unrolling of transition relation

(assert (!
(let (($x1 (and (and at_1_0 (not visited_2_0) at_2_1
    visited_2_1 (not at_1_1)) (and (= visited_1_1 visited_1_0))))))
(let (($x2 ... )))
(or $x1 $x2))) :named T))

; Assert formula for goal states

(assert (! (not visited_1_1) :named G1) )
(assert (! visited_2_1 :named G2))
... ; Check whether the formula is satisfiable
(check-sat)

; If unsat, produce unsat core
(get-unsat-core)

; Solver returns

; unsat
; (i2 t g1)
SMT encoding - I

; SMT encoding for TSP
...

; Declare variables
(declare-fun at_0 () Int)
(declare-fun at_1 () Int)

; Declare UF to encode predicate
(declare-fun visited (Int Int) Bool)

; Assert bounds on integers
(assert (and (>= at_0 1) (<= at_0 2)))
(assert (and (>= at_1 1) (<= at_1 2)))

; Assert formula for initial state
(assert (and (and (= at_0 1) (visited 1 0)) (not (visited 2 0)))))
Some theories in: QF_UFLIA

SMT encoding - II

; Assert unrolling of transition relation

(assert
(let (($x1 (and (= at_0 1) (and (not (visited 2 0))) (= at_1 2)
(visited 2 1) (and (= (visited 1 1) (visited 1 0))))))
(let (($x2 ... )))
(or $x1 $x2))))

; Assert formula for goal state

(assert (and (visited 1 1) (visited 2 1)))

; Check sat. If sat, retrieve model

(check-sat)
(get-value (at_0 (visited 1 0) (visited 2 0) at_1
(visited 1 1) (visited 2 1)))
Some theories in: QF_UFLIA

SMT encoding - II

; Solver returns

sat
((at_0 1)
((visited 1 0) true)
((visited 2 0) false)
(at_1 2)
((visited 1 1) true)
((visited 2 1) true))
; OMT encoding for TSP
; benchmark generated from python API
(set-info :status sat)

; Declare variables

(declare-fun at_0 () Int)
(declare-fun at_1 () Int)

; Cost variables

(declare-fun c_0 () Int)
(declare-fun c_1 () Int)

; Assert formula for initial state
(assert (and ... (= c_0 0) ...))
(assert
(let (((x1 (and (= at_0 1) (and (not (visited 2 0))) (= at_1 2)
(visited 2 1) (and (= (visited 1 1) (visited 1 0))) (= c_1 (+ c_0 3)))))
(let (((x2 ... ))))

; Define objective function
(minimize c_1)

(check-sat)

; Solver returns
; sat
;(objectives
; (c_1 3)
;)

;
Outline

SMT solving

I Historical notes

II SAT and SMT solving

III Some applications outside planning

SMT solving for planning

IV SMT and planning

V Application: optimal planning with OMT

Concluding remarks
Planning in the era of *Smart Factories*
Planning in the era of *Smart Factories*
Planning & Execution Competition for Logistics Robots in Simulation [Niemueller et al., 2015]

Source: [Zwilling et al., 2014].
Planning & Execution Competition for Logistics Robots in Simulation [Niemueller et al., 2015]

Source: [RCLL Technical Committee, 2017].
Temporal planning with OMT for the RCLL. What’s hard?

- time windows
- domain representation: over 250 configurations possible!
- combinatorics
- scalability

🪤 compact representations are needed to help solvers
The need for compact encodings

\[
\begin{align*}
S_0 & \xrightarrow{a_1} S_1 & \xrightarrow{a_2} S_2 & \xrightarrow{a_3} S_3 & \xrightarrow{a_4} S_4 
\end{align*}
\]
The need for compact encodings

\[ S_0 \xrightarrow{a_1} S_1 \xrightarrow{a_2} S_2 \xrightarrow{a_3} S_3 \xrightarrow{a_4} S_4 \]
The need for compact encodings

\[ S_0 \xrightarrow{a_1 \land a_2} S_1 \xrightarrow{a_3} S_2 \xrightarrow{a_4} S_3 \]
The need for compact encodings
The need for compact encodings

\[ S_0 \xrightarrow{a_1 \land a_2} S_1 \xrightarrow{a_3} S_2 \xrightarrow{a_4} S_3 \]
The need for compact encodings
A reduced encoding [Leofante et al., 2018]
A reduced encoding [Leofante et al., 2018]
A reduced encoding [Leofante et al., 2018]
A reduced encoding [Leofante et al., 2018]
A reduced encoding [Leofante et al., 2018]

\[ S_0 \quad S_1 \quad S_2 \quad S_3 \quad S_4 \]
A reduced encoding [Leofante et al., 2018]

\[ S_0 \quad S_1 \quad S_2 \quad S_3 \quad S_4 \]
A reduced encoding [Leofante et al., 2018]
A reduced encoding [Leofante et al., 2018]
Integrated synthesis and execution

Model \rightarrow \text{Synthesis} \rightarrow \text{Plan} \rightarrow \text{Executive} \rightarrow \text{Robot}

- Model
- Synthesis
- Plan
- Executive
- Robot
- Monitor
We implemented a domain-specific planner for the RCLL.
We implemented a domain-specific planner for the RCLL

“Cool but...How does it perform on other planning problems?”

We implemented a domain-specific planner for the RCLL

“Cool but...How does it perform on other planning problems?”


We’re working on a planner implementing our ideas, stay tuned!
Outline

SMT solving

I  Historical notes
II  SAT and SMT solving
III Some applications outside planning

SMT solving for planning

IV  SMT and planning
V  Application: optimal planning with OMT

Concluding remarks
Concluding remarks

- Satisfiability checking combines methods in innovative ways
Concluding remarks

- Satisfiability checking combines methods in innovative ways
  - emphasis on practical efficiency.
Concluding remarks

- Satisfiability checking combines methods in innovative ways
  - emphasis on practical efficiency.
- SAT and SMT solvers are powerful general-purpose tools.
Concluding remarks

- Satisfiability checking combines methods in innovative ways
  - emphasis on practical efficiency.
- SAT and SMT solvers are powerful general-purpose tools.
- SMT has a wide (and increasing) range of application areas
  - planning is one of them!
Concluding remarks

- Satisfiability checking combines methods in innovative ways
  - emphasis on practical efficiency.
- SAT and SMT solvers are powerful general-purpose tools.
- SMT has a wide (and increasing) range of application areas
  - planning is one of them!
- The SMT solving community is always looking for interesting problems...

Have one? Let’s talk!


Z3: an efficient SMT solver.  

Monitoring modulo theories.  

Prophesy: A probabilistic parameter synthesis tool.  

A tour of CVC4: how it works, and how to use it.  
References IV


Automated termination proofs with aprove.

Planning as satisfiability with expressive action languages: Concurrency, constraints and nondeterminism.

Planning as satisfiability with preferences.


CBMC - C bounded model checker - (competition contribution).

Integrated synthesis and execution of optimal plans for multi-robot systems in logistics.
*Information Systems Frontiers*.

Algorithms for computing minimal unsatisfiable subsets of constraints.

Boolector 2.0.

The RoboCup Logistics League as a benchmark for planning in robotics.
In *Proc. of PlanRob@ICAPS’15*.

RCLL Technical Committee (2017).


SATO webpage.
SATO solver.
http://homepage.divms.uiowa.edu/~hzhang/sato/.


