

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





Erika
Full professor



Francesco
Ph.D. Student

Theory of Hybrid Systems @ RWTH

<https://ths.rwth-aachen.de>

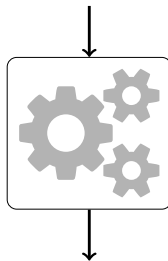
Resources for this tutorial

<https://ths.rwth-aachen.de/research/talks/smt4planning>

What is this tutorial about?

What is satisfiability checking?

$$\neg \mathbf{a} \wedge \mathbf{b} \vee \mathbf{c}$$
$$\mathbf{x}^2 + \mathbf{x}_2 \quad \sqrt{\varphi}$$

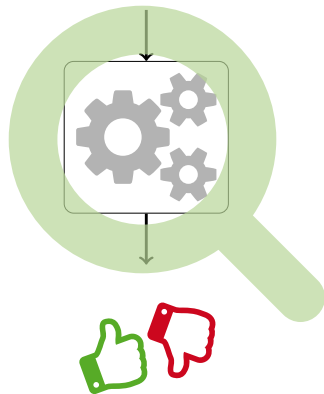


What is this tutorial about?

What is satisfiability checking?

How does SMT solving work?

$$\neg \mathbf{a} \wedge \mathbf{b} \vee \mathbf{c}$$
$$\mathbf{x}^2 + \mathbf{x}_2 \quad \sqrt{\varphi}$$



What is this tutorial about?

Planning problem

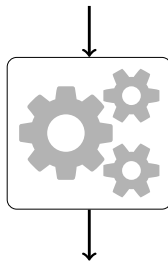


$$\neg \mathbf{a} \wedge \mathbf{b} \vee \mathbf{c}$$
$$\mathbf{x}^2 + \mathbf{x}_2 \quad \sqrt{\varphi}$$

What is satisfiability checking?

How does SMT solving work?

How to use it for planning?



Plan



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 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

Formula: $(a \vee \neg b) \wedge (\neg a \vee b \vee c)$

Satisfying assignment: $a = \text{true}, b = \text{false}, c = \text{true}$

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

The satisfiability problem

Propositional logic

Formula: $(a \vee \neg b) \wedge (\neg a \vee b \vee c)$

Satisfying assignment: $a = \text{true}, b = \text{false}, c = \text{true}$

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

Non-linear real algebra (NRA)

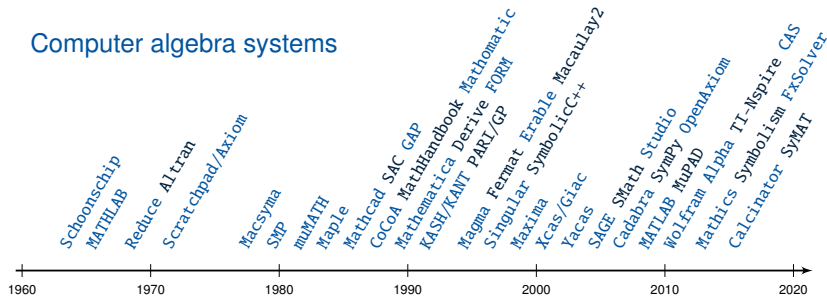
Formula: $(x - 2y > 0 \vee x^2 - 2 = 0) \wedge x^4 y + 2x^2 - 4 > 0$

Satisfying assignment: $x = \sqrt{2}, y = 2$

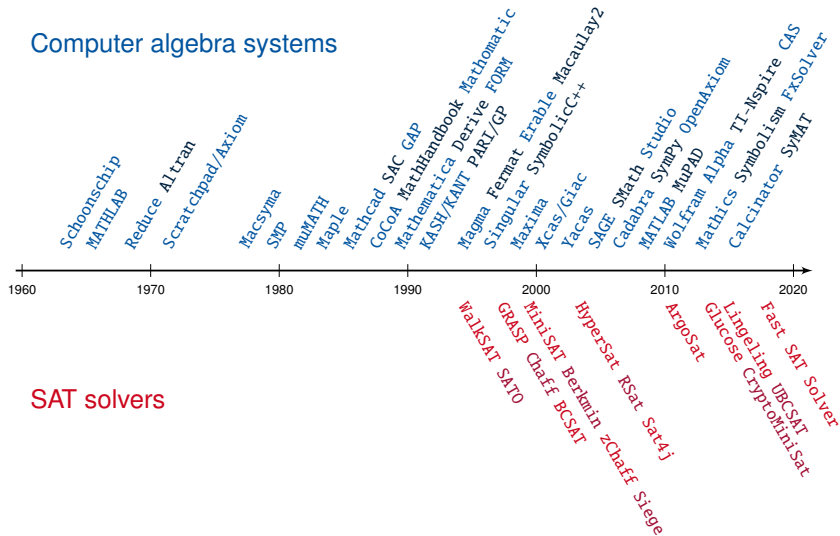
There are some hard problem classes... non-linear integer arithmetic is even undecidable.

Tool development (incomplete!)

Computer algebra systems

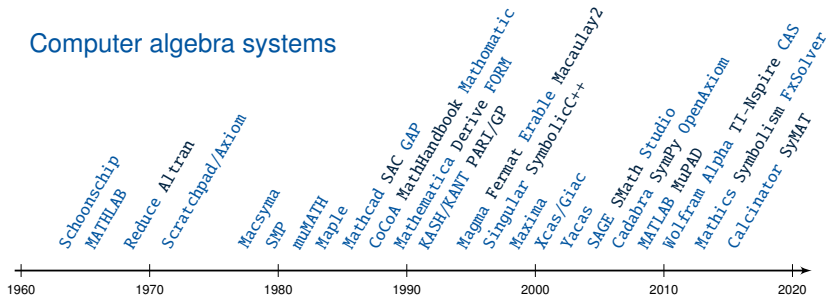


Tool development (incomplete!)



Tool development (incomplete!)

Computer algebra systems

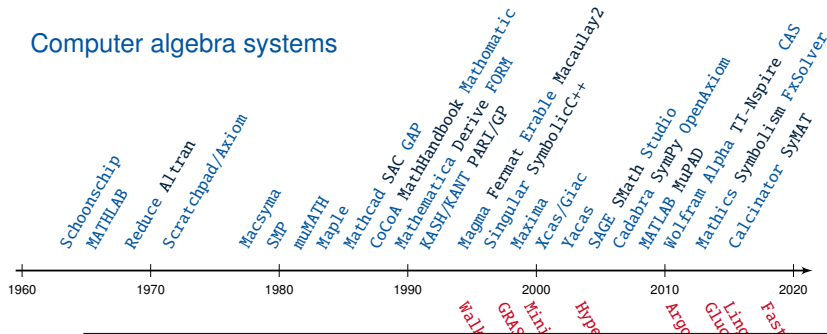


“We have success stories of using zChaff to solve problems with more than one million variables and 10 million clauses. (Of course, it can’t solve every such problem!).” [zch,]

SAT

Tool development (incomplete!)

Computer algebra systems



SAT

“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.

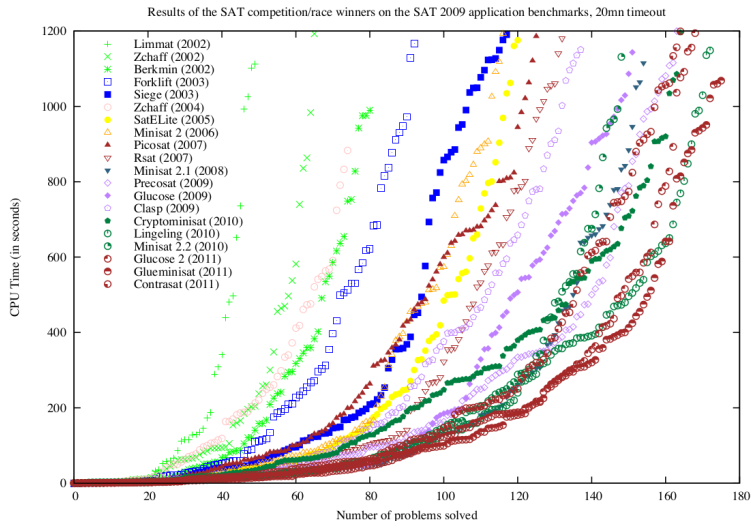
Success story: **SAT-solving**

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

Community support:

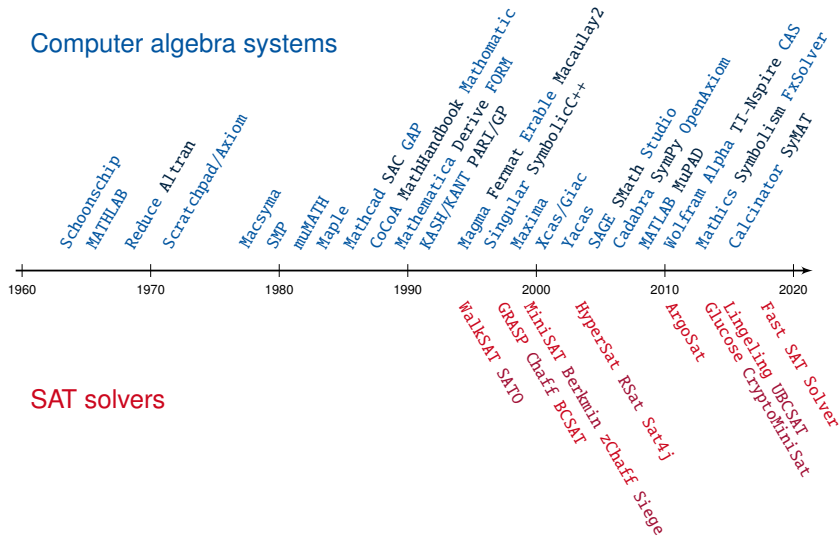
- **Standard input language.**
- Large **benchmark library.**
- **Competitions** since 2002.
2017: 6 tracks, 28 solvers in the main track.
- **SAT Live!** forum as community platform, dedicated conferences, journals, etc.

An impression of the SAT solver development



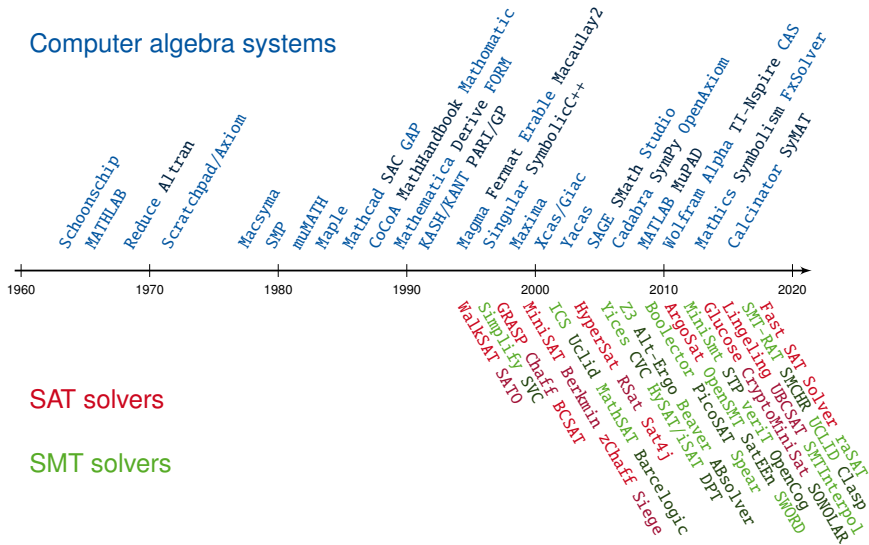
Source: The International SAT Solver Competitions [Järvisalo et al., 2012]

Tool development (incomplete!)

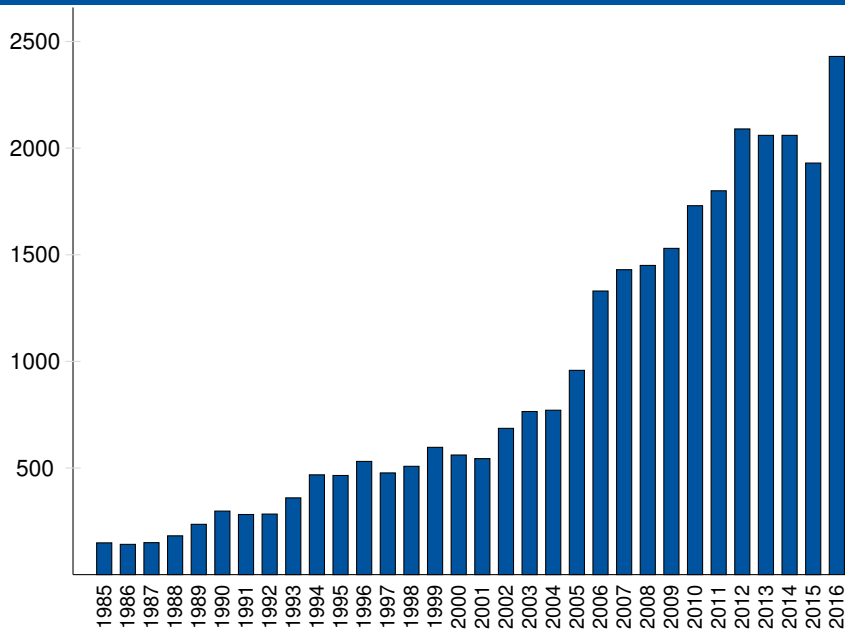


SAT solvers

Tool development (incomplete!)



Google Scholar search for “SAT modulo 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.**

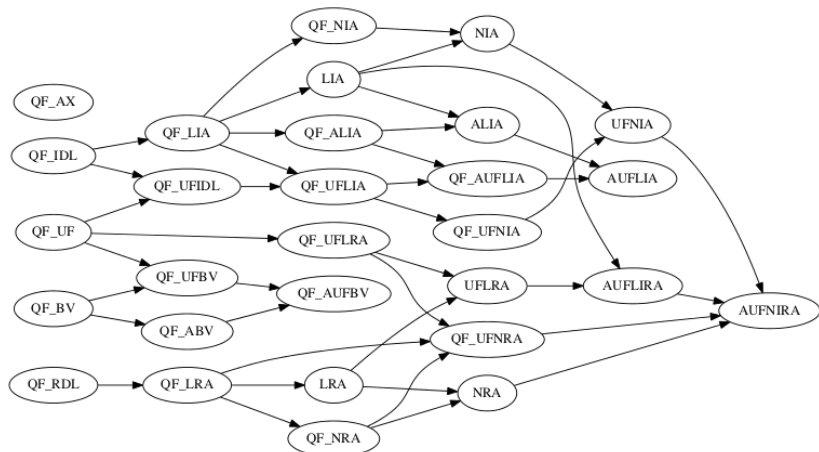
Satisfiability modulo theories (SMT) solving:

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

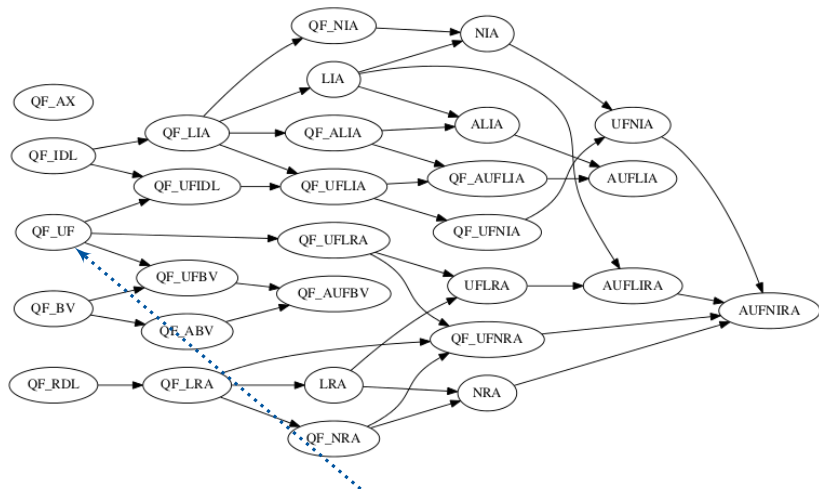
Community support:

- **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

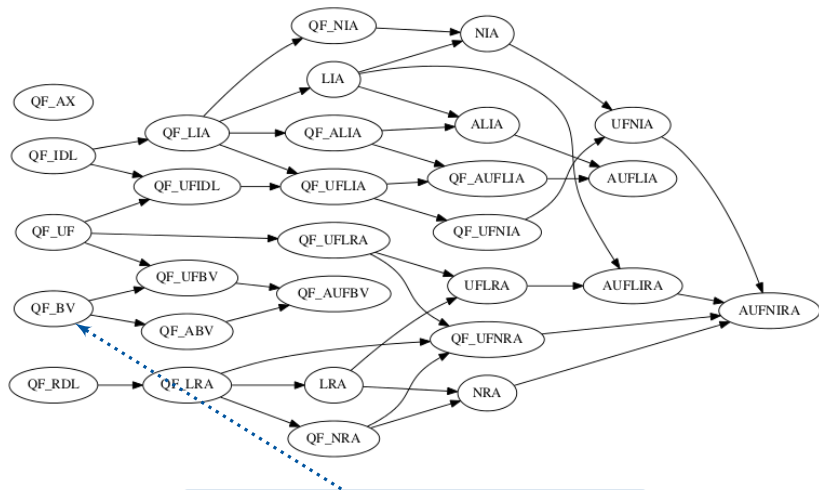


Source: <http://smtlib.cs.uiowa.edu/logics.shtml>

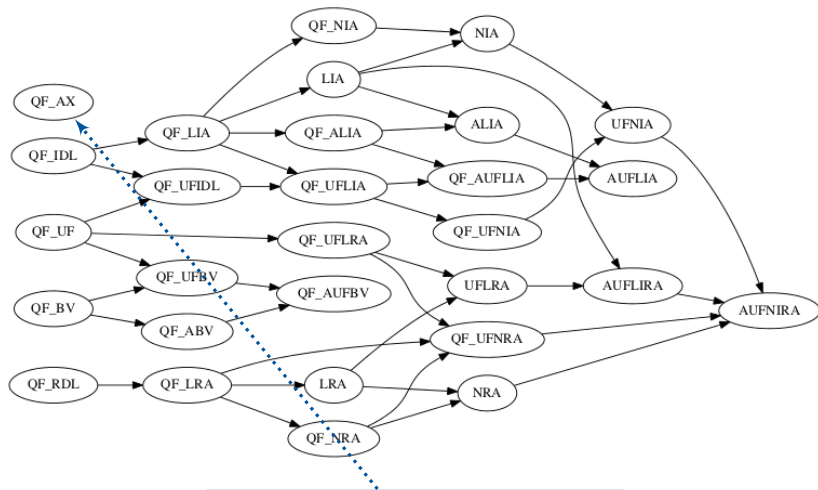


Quantifier-free equality logic with uninterpreted functions
 $(a = c \wedge b = d) \rightarrow f(a, b) = f(c, d)$

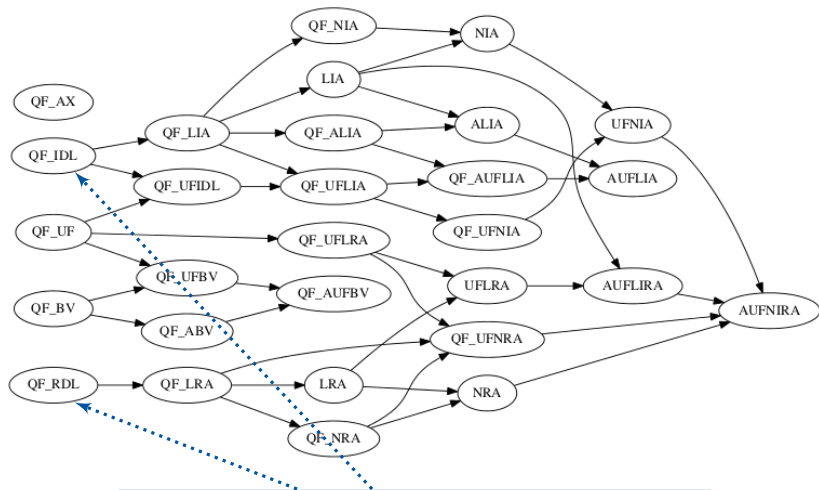
SMT-LIB logics



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



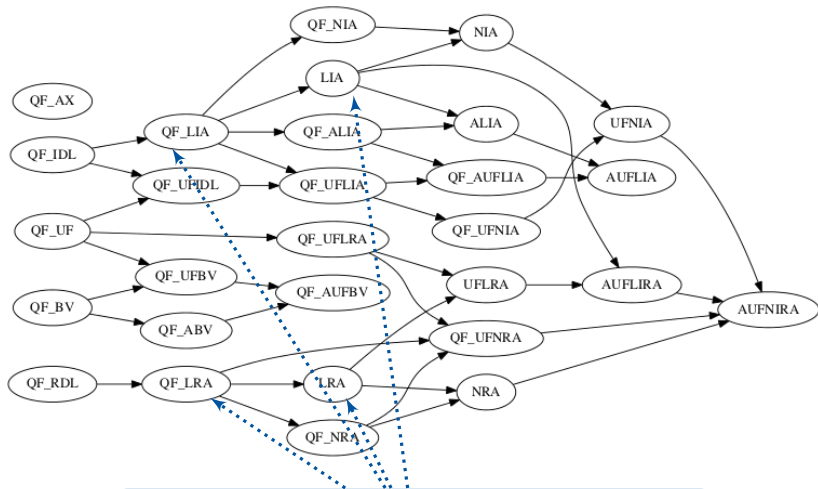
Quantifier-free array theory
 $i = j \rightarrow \text{read}(\text{write}(a, i, v), j) = v$



Quantifier-free integer/rational difference logic

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

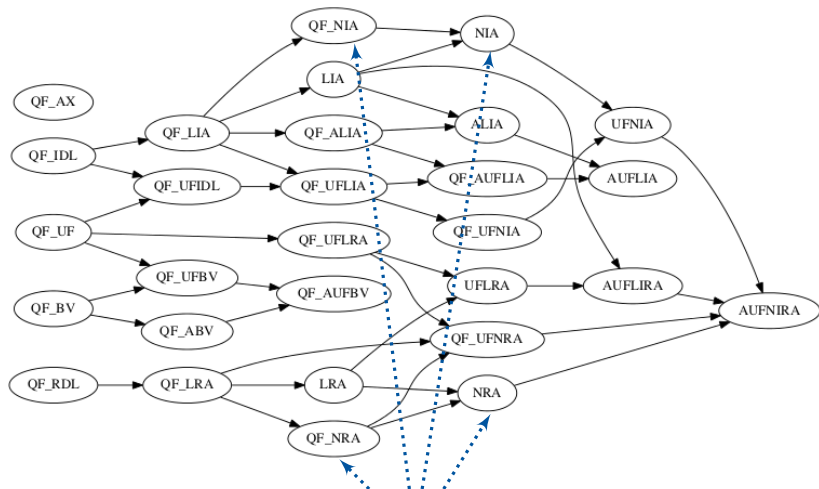
SMT-LIB logics



(Quantifier-free) real/integer linear arithmetic

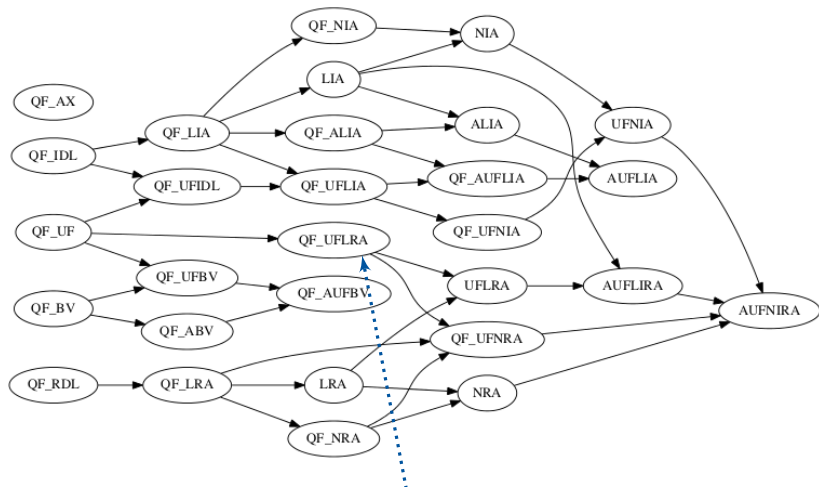
$$3x + 7y = 8$$

SMT-LIB logics



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

SMT-LIB logics



Combined theories
 $2f(x) + 5y > 0$

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

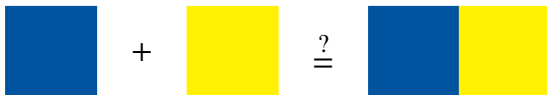
Concluding remarks

Strategic combinations of decision procedures

Strategic combinations of decision procedures



Strategic combinations of decision procedures



Strategic combinations of decision procedures



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

$$c_1 : (\neg a \vee b) \wedge$$

$$c_2 : (\neg b \vee \neg c) \wedge$$

$$c_3 : (\neg b \vee c) \wedge$$

...

DPLL SAT solving with conflict-directed clause learning

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

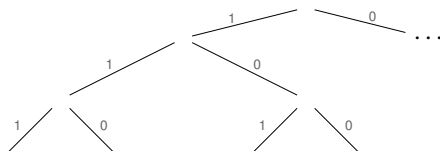
Ingredients: Enumeration

$$c_1 : (\neg a \vee b) \wedge$$

$$c_2 : (\neg b \vee \neg c) \wedge$$

$$c_3 : (\neg b \vee c) \wedge$$

...



DPLL SAT solving with conflict-directed clause learning

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

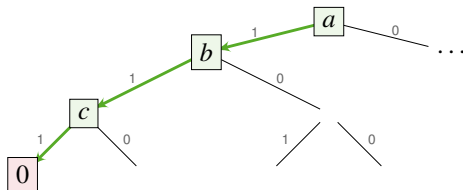
Ingredients: Enumeration

$$c_1 : (\neg a \vee b) \wedge$$

$$c_2 : (\neg b \vee \neg c) \wedge$$

$$c_3 : (\neg b \vee c) \wedge$$

...



DPLL SAT solving with conflict-directed clause learning

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

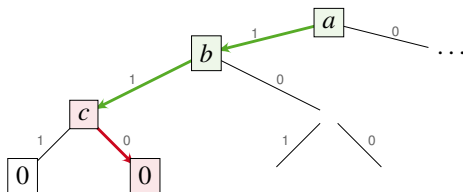
Ingredients: Enumeration

$$c_1 : (\neg a \vee b) \wedge$$

$$c_2 : (\neg b \vee \neg c) \wedge$$

$$c_3 : (\neg b \vee c) \wedge$$

...



DPLL SAT solving with conflict-directed clause learning

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

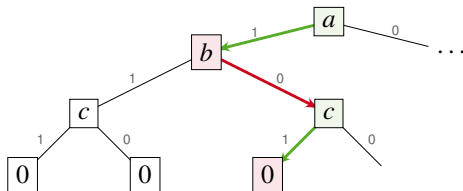
Ingredients: Enumeration

$$c_1 : (\neg a \vee b) \wedge$$

$$c_2 : (\neg b \vee \neg c) \wedge$$

$$c_3 : (\neg b \vee c) \wedge$$

...



DPLL SAT solving with conflict-directed clause learning

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

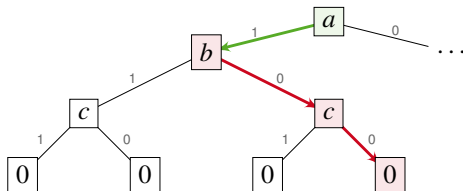
Ingredients: Enumeration

$$c_1 : (\neg a \vee b) \wedge$$

$$c_2 : (\neg b \vee \neg c) \wedge$$

$$c_3 : (\neg b \vee c) \wedge$$

...



DPLL SAT solving with conflict-directed clause learning

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

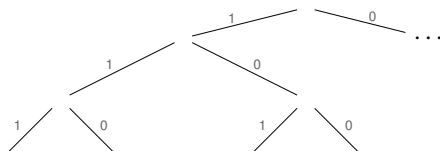
Ingredients: Enumeration + Boolean constraint propagation

$$c_1 : (\neg a \vee b) \wedge$$

$$c_2 : (\neg b \vee \neg c) \wedge$$

$$c_3 : (\neg b \vee c) \wedge$$

...



DPLL SAT solving with conflict-directed clause learning

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

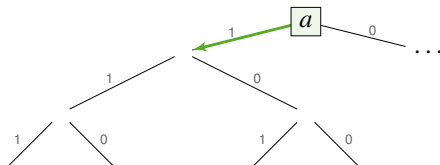
Ingredients: Enumeration + Boolean constraint propagation

$$c_1 : (\neg a \vee b) \wedge$$

$$c_2 : (\neg b \vee \neg c) \wedge$$

$$c_3 : (\neg b \vee c) \wedge$$

...

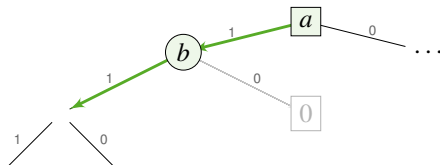


DPLL SAT solving with conflict-directed clause learning

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

Ingredients: Enumeration + Boolean constraint propagation

$$\begin{aligned}c_1 &: (\boxed{\neg a} \vee \boxed{b}) \wedge \\c_2 &: (\boxed{\neg b} \vee \neg c) \wedge \\c_3 &: (\boxed{\neg b} \vee c) \wedge \\&\dots\end{aligned}$$



DPLL SAT solving with conflict-directed clause learning

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

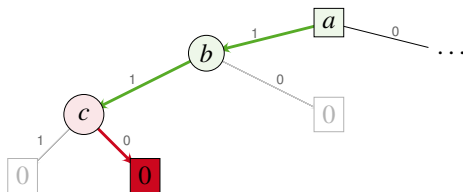
Ingredients: Enumeration + Boolean constraint propagation

$$c_1 : (\neg a \vee b) \wedge$$

$$c_2 : (\neg b \vee \neg c) \wedge$$

$$c_3 : (\neg b \vee c) \wedge$$

...



DPLL SAT solving with conflict-directed clause learning

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

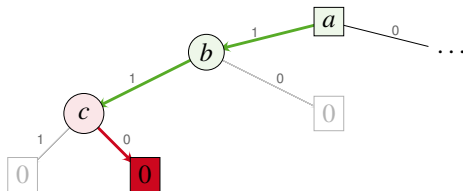
Ingredients: Enumeration + Boolean constraint propagation + Resolution

$$c_1 : (\boxed{\neg a} \vee \boxed{b}) \wedge$$

$$c_2 : (\boxed{\neg b} \vee \boxed{\neg c}) \wedge$$

$$c_3 : (\boxed{\neg b} \vee \boxed{c}) \wedge$$

...



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

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

Derivation rule form:

$$\frac{antecedent_1 \quad \dots \quad antecedent_n}{consequent} \quad Rule_name$$

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

Derivation rule form:

$$\frac{\textit{antecedent}_1 \quad \dots \quad \textit{antecedent}_n}{\textit{consequent}} \textit{Rule_name}$$

$$\frac{(l_1 \vee \dots \vee l_n \vee x) \quad (l'_1 \vee \dots \vee l'_m \vee \neg x)}{(l_1 \vee \dots \vee l_n \vee l'_1 \vee \dots \vee l'_m)} \textit{Rule}_{\textit{res}}$$

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

Derivation rule form:

$$\frac{\text{antecedent}_1 \quad \dots \quad \text{antecedent}_n}{\text{consequent}} \text{Rule_name}$$

$$\frac{(l_1 \vee \dots \vee l_n \vee x) \quad (l'_1 \vee \dots \vee l'_m \vee \neg x)}{(l_1 \vee \dots \vee l_n \vee l'_1 \vee \dots \vee l'_m)} \text{Rule}_{\text{res}}$$

$$\exists x. C_x \wedge C_{\neg x} \wedge C \quad \leftrightarrow \quad \text{Resolvents}(C_x, C_{\neg x}) \wedge C$$

DPLL SAT solving with conflict-directed clause learning

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

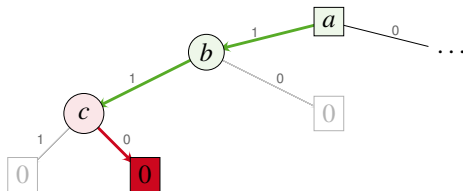
Ingredients: Enumeration + Boolean constraint propagation + Resolution

$$c_1 : (\text{ } \neg a \text{ } \vee \text{ } b \text{ }) \wedge$$

$$c_2 : (\text{ } \neg b \text{ } \vee \text{ } \neg c \text{ }) \wedge$$

$$c_3 : (\text{ } \neg b \text{ } \vee \text{ } c \text{ }) \wedge$$

...



DPLL SAT solving with conflict-directed clause learning

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

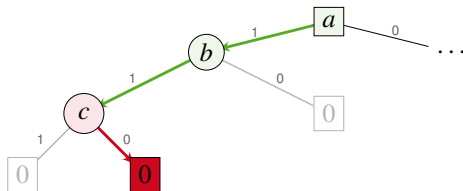
Ingredients: Enumeration + Boolean constraint propagation + Resolution

$$c_1 : (\textcolor{red}{\neg a} \vee \textcolor{green}{b}) \wedge$$

$$c_2 : (\textcolor{red}{\neg b} \vee \textcolor{green}{\neg c}) \wedge$$

$$c_3 : (\textcolor{red}{\neg b} \vee \textcolor{red}{c}) \wedge$$

...



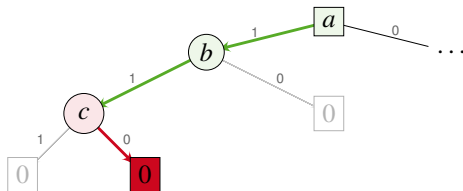
$$\frac{c_3 : (\neg b \vee \textcolor{red}{c}) \quad c_2 : (\neg b \vee \textcolor{green}{\neg c})}{c_4 : (\neg b)} \quad \textit{Resolution}$$

DPLL SAT solving with conflict-directed clause learning

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

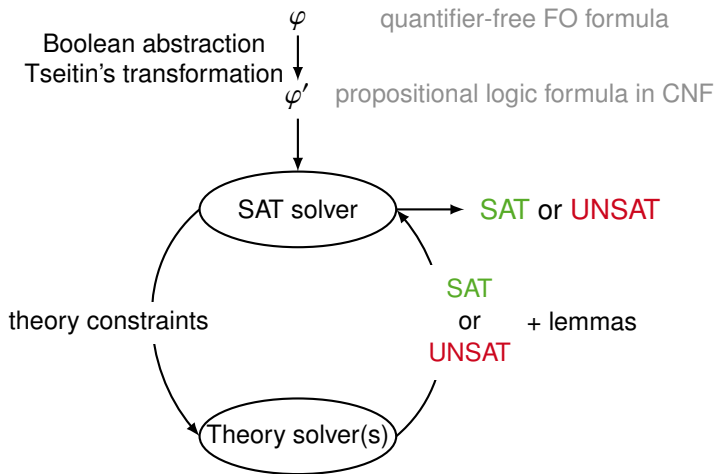
Ingredients: Enumeration + Boolean constraint propagation + Resolution

$$\begin{aligned} c_1 : & (\neg a \vee b) \wedge \\ c_2 : & (\neg b \vee \neg c) \wedge \\ c_3 : & (\neg b \vee c) \wedge \\ c_4 : & (\neg b) \wedge \\ & \dots \end{aligned}$$



$$\frac{c_3 : (\neg b \vee \textcolor{red}{c}) \quad c_2 : (\neg b \vee \textcolor{green}{\neg c})}{c_4 : (\neg b)} \quad \textit{Resolution}$$

(Full/less) lazy SMT solving



Less lazy SMT solving

Less lazy SMT solving

$$(x < 0 \vee x > 2) \wedge (x^2 = 1 \vee x^2 < 0)$$

Less lazy SMT solving

$$(x < 0 \vee x > 2) \wedge (x^2 = 1 \vee x^2 < 0)$$

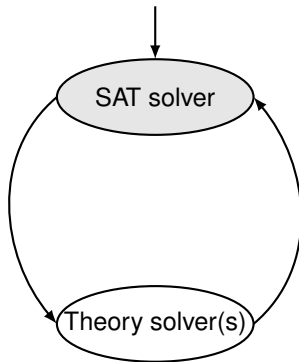


$$(a \vee b) \wedge (c \vee d)$$

Less lazy SMT solving

$$(x < 0 \vee x > 2) \wedge (x^2 = 1 \vee x^2 < 0)$$

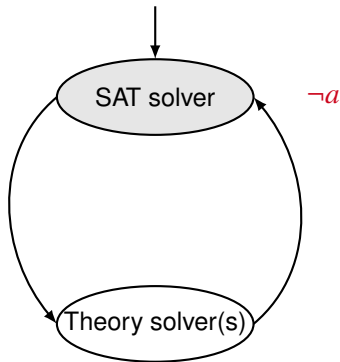
$$(\textcolor{red}{a} \vee \textcolor{blue}{b}) \wedge (\textcolor{green}{c} \vee \textcolor{brown}{d})$$



Less lazy SMT solving

$$(x < 0 \vee x > 2) \wedge (x^2 = 1 \vee x^2 < 0)$$

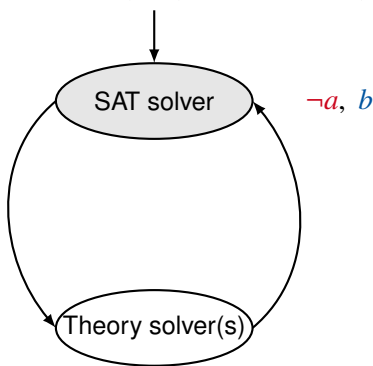
$$(\textcolor{red}{a} \vee \textcolor{blue}{b}) \wedge (\textcolor{green}{c} \vee \textcolor{brown}{d})$$



Less lazy SMT solving

$$(x < 0 \vee x > 2) \wedge (x^2 = 1 \vee x^2 < 0)$$

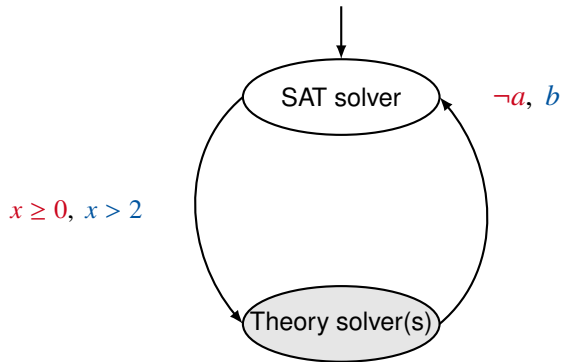
$$(\textcolor{red}{a} \vee \textcolor{blue}{b}) \wedge (\textcolor{green}{c} \vee \textcolor{brown}{d})$$



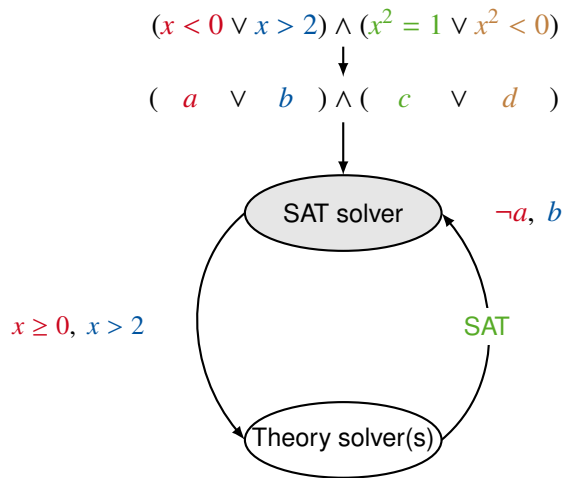
Less lazy SMT solving

$$(x < 0 \vee x > 2) \wedge (x^2 = 1 \vee x^2 < 0)$$

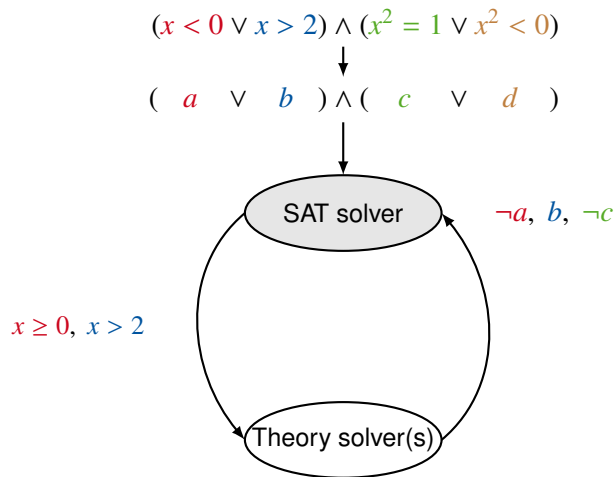
$$(\textcolor{red}{a} \vee \textcolor{blue}{b}) \wedge (\textcolor{green}{c} \vee \textcolor{brown}{d})$$



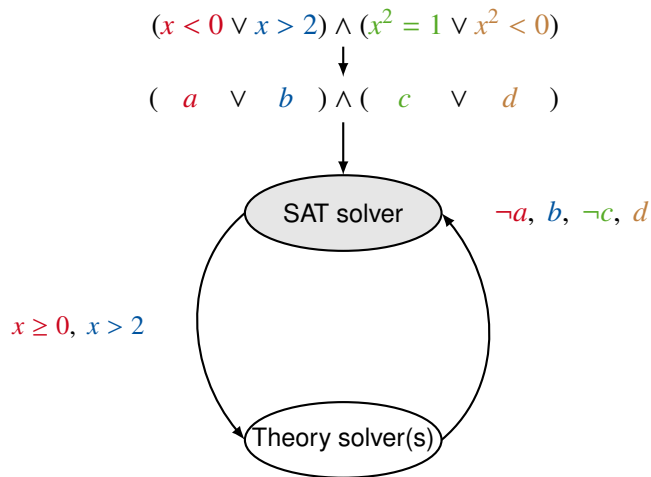
Less lazy SMT solving



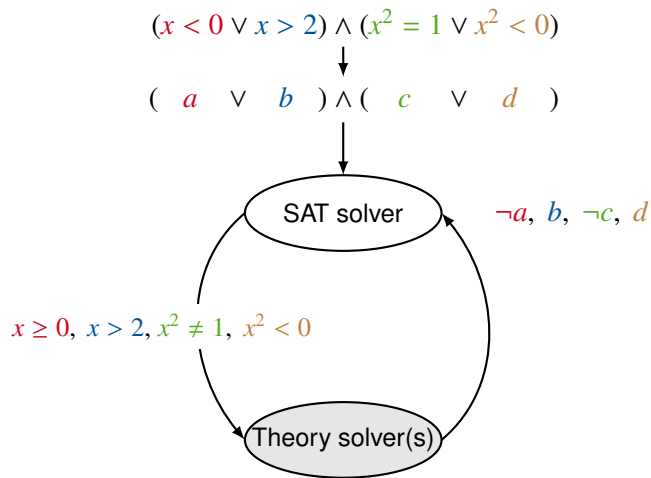
Less lazy SMT solving



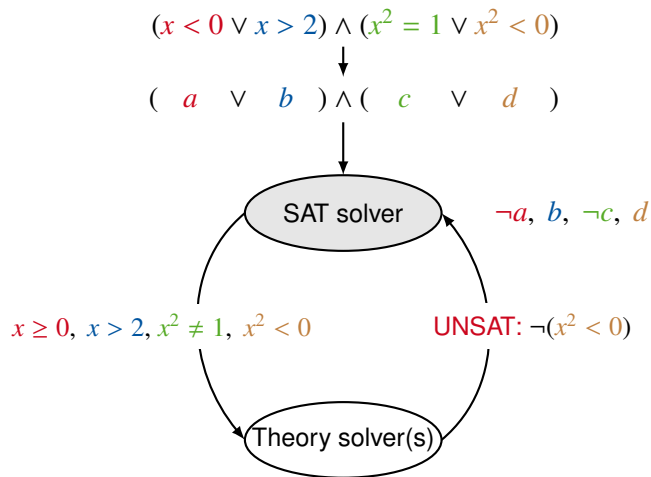
Less lazy SMT solving



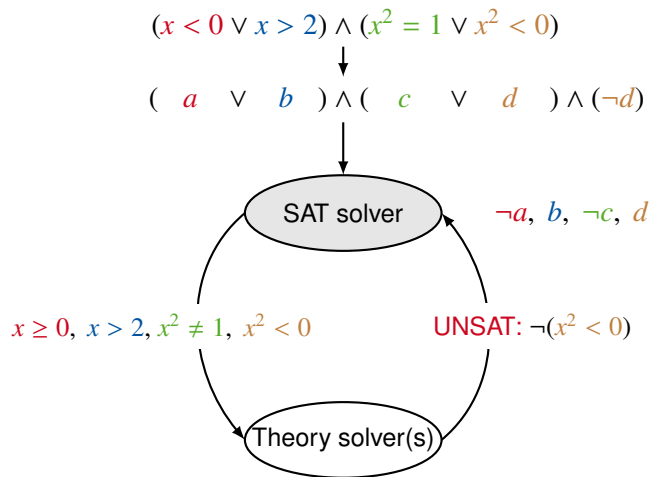
Less lazy SMT solving



Less lazy SMT solving



Less lazy SMT solving

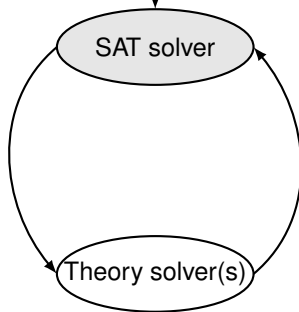


Less lazy SMT solving

$$(x < 0 \vee x > 2) \wedge (x^2 = 1 \vee x^2 < 0)$$



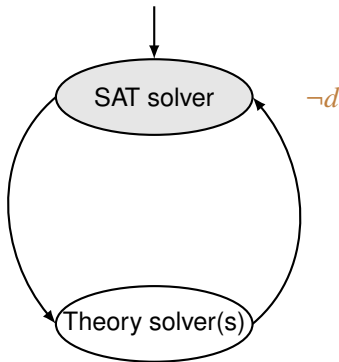
$$(a \vee b) \wedge (c \vee d) \wedge (\neg d)$$



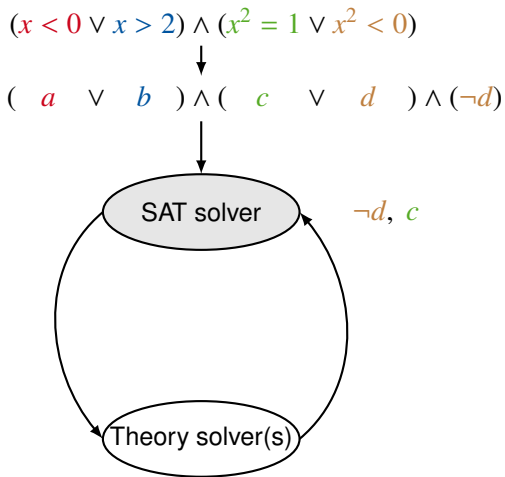
Less lazy SMT solving

$$(x < 0 \vee x > 2) \wedge (x^2 = 1 \vee x^2 < 0)$$

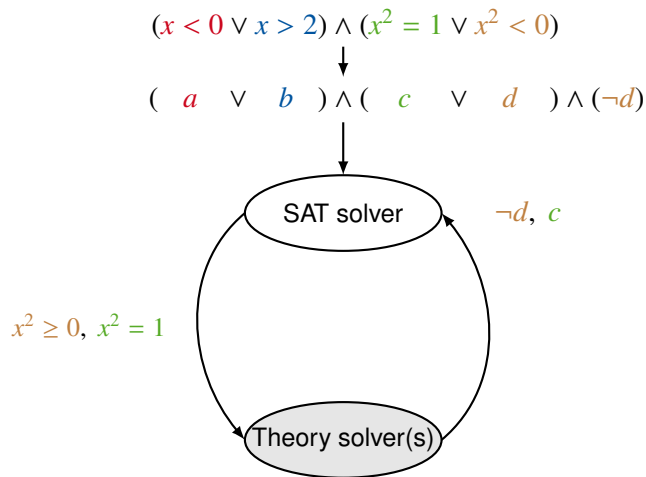
$$(\textcolor{red}{a} \vee \textcolor{blue}{b}) \wedge (\textcolor{green}{c} \vee \textcolor{brown}{d}) \wedge (\neg \textcolor{brown}{d})$$



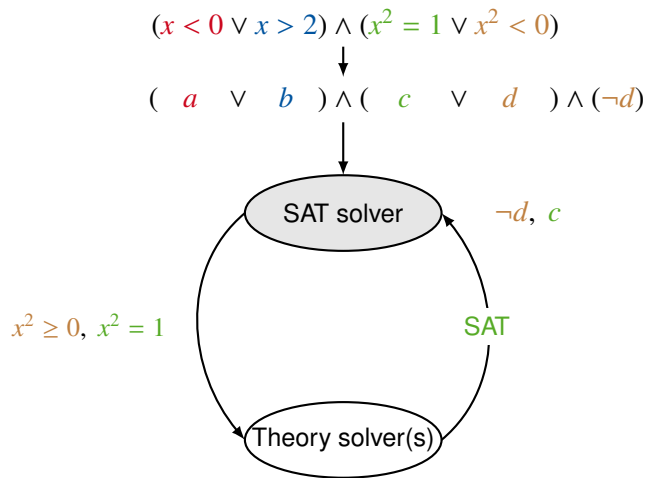
Less lazy SMT solving



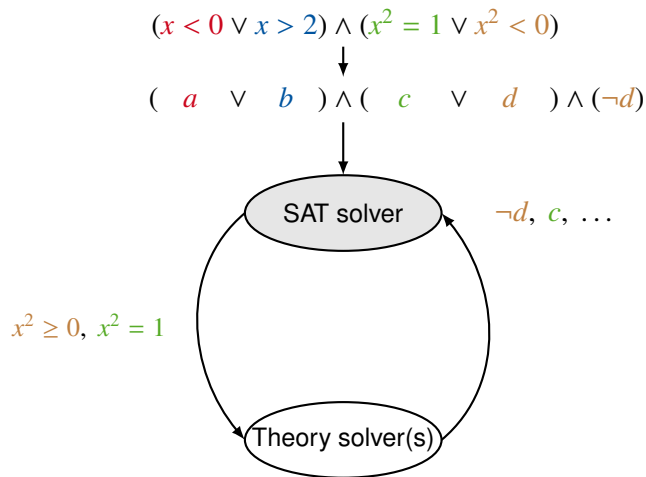
Less lazy SMT solving



Less lazy SMT solving



Less lazy SMT solving



Model constructing satisfiability calculus (MCSAT)

\mathbb{B} -decision

\mathbb{B} -propagation

\mathbb{B} -conflict resolution

Model constructing satisfiability calculus (MCSAT)

B-decision

T-decision

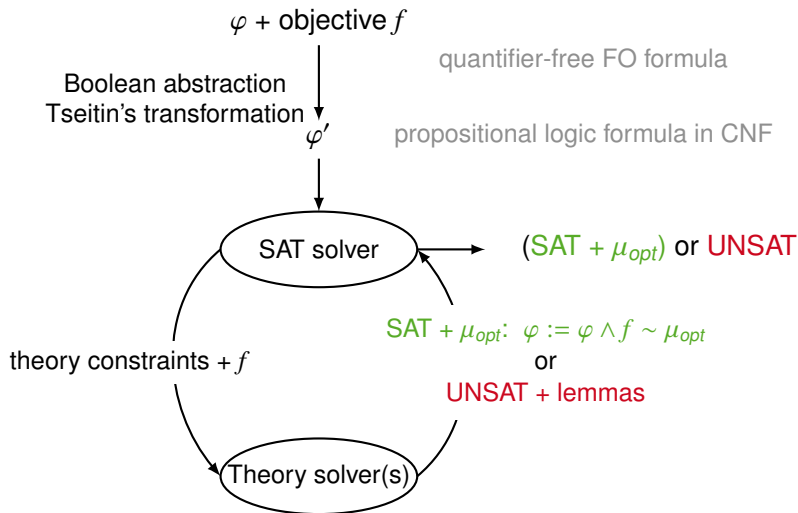
B-propagation

T-propagation

B-conflict resolution

T-conflict resolution

Optimization modulo theories (full lazy case)



Some theory solver candidates for arithmetic theories

Linear real arithmetic:

- Simplex
- Ellipsoid method
- Fourier-Motzkin variable elimination
(mostly preprocessing)
- 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)

Linear integer arithmetic:

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

Non-linear integer arithmetic:

- Generalised branch-and-bound
(incomplete)
- Bit-blasting (eager, incomplete)
- Interval constraint propagation
(incomplete)

Problem solved?

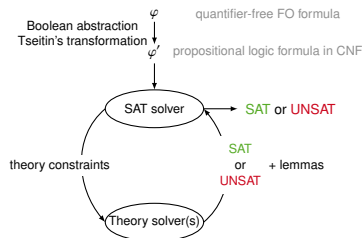
Can we simply plug in available implementations of such methods as theory solvers into an SMT solver?

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**.

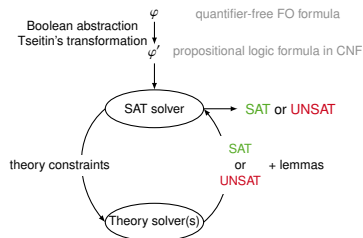


Problem solved?

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**.



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

Bridging Two Communities to Solve Real Problems

Coordination and Support Activity

SUMMARY

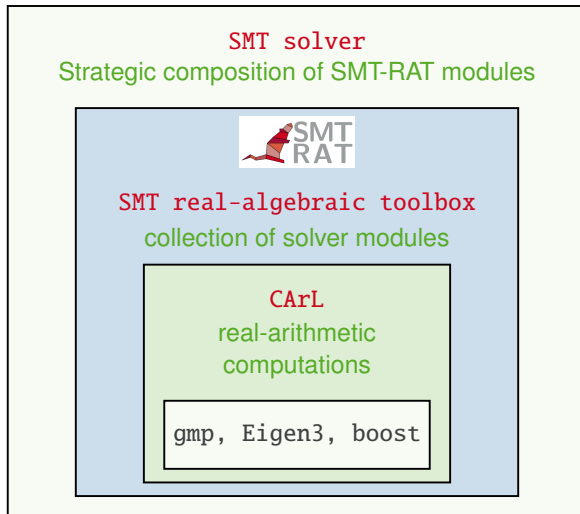
This project is funded (subject to contract) as project H2020-FETOPN-2015-CSA_712689 of the European Union. It is the start of the general push to create a real **SC² community**.

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!)

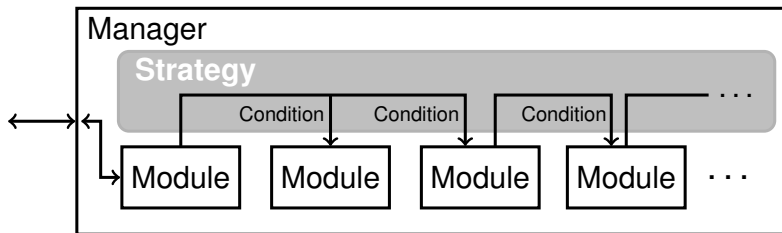
- AProVE (RWTH Aachen University, Germany) [Giesl et al., 2004]
- CVC4 (New York and Iowa, USA) [Deters et al., 2014]
- MathSAT 5 (FBK, Italy) [Cimatti et al., 2013]
- MiniSmt (University of Innsbruck, Austria) [Zankl and Middeldorp, 2010]
- Boolector (JKU, Austria) [Niemetz et al., 2014]
- SMT-RAT (RWTH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



- 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 → parallelism



Module

Implements

- `add(Formula)`
- `remove(Formula)`
- `check()`
- `updateModel()`

Module

Implements

- `add(Formula)`
- `remove(Formula)`
- `check()`
- `updateModel()`

`check()` may

- forward (sub-)problems to **backend** modules
- return **sat** or **unsat**
- return a **lemma** or **split**
- return **unknown**

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]

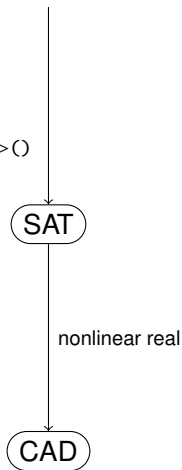
Virtual substitution [FCT'11, SC²'17, PhD Corzilius]

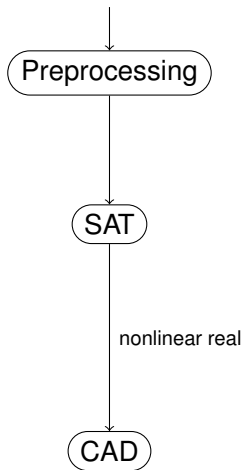
Generalized branch-and-bound [CASC'16]

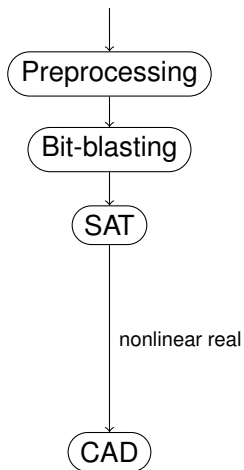
Cube tests

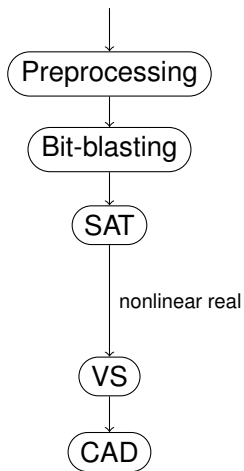
SMT-RAT strategies

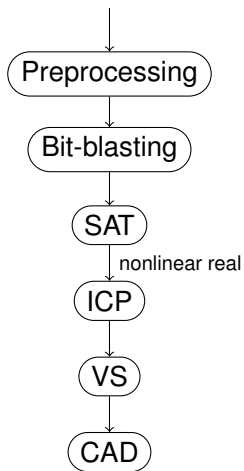
```
class myStrategy: public Manager {  
  myStrategy(): Manager() {  
    setStrategy(  
      addBackend<SATModule<SATSettings>>(  
        addBackend<CADModule<CADSettings>>()  
      )  
    );  
  }  
};
```

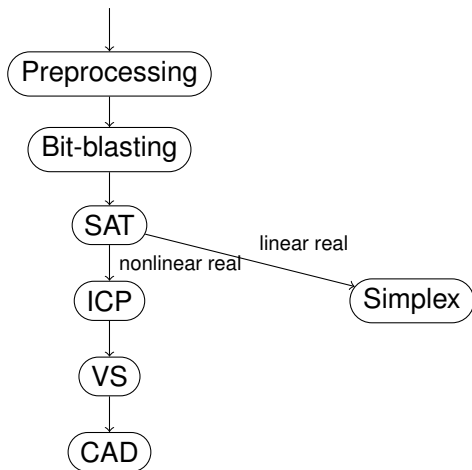


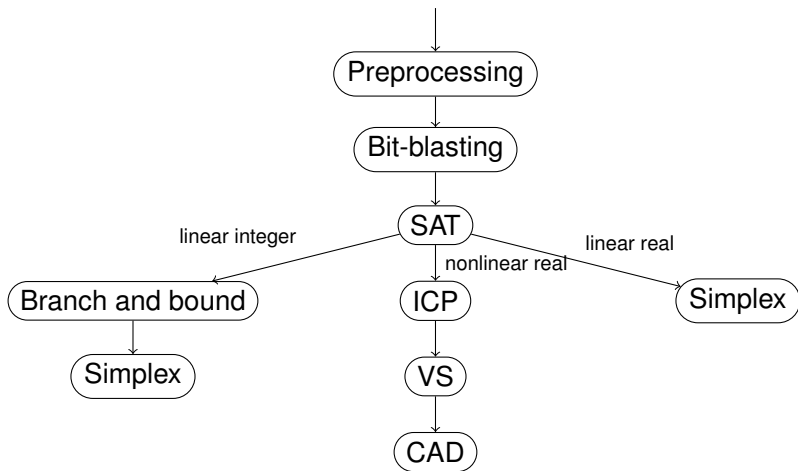












Building an SMT solver from SMT-RAT modules

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
```

SMT solving

I Historical notes

II SAT and SMT solving

III Some applications outside planning

SMT solving for planning

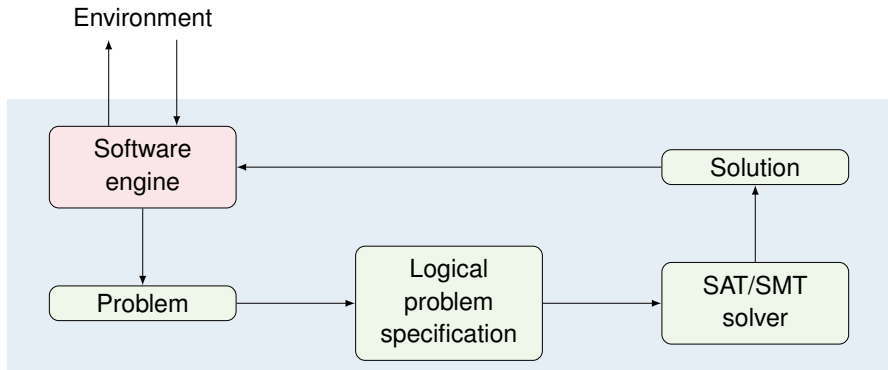
IV SMT and planning

V Application: optimal planning with OMT

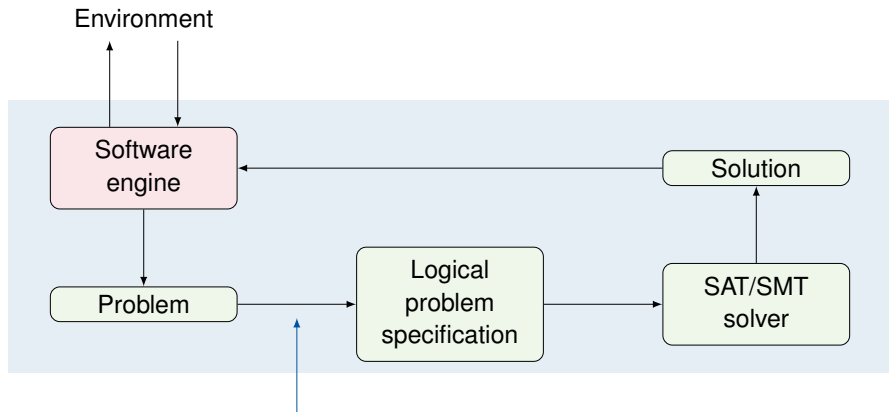
Concluding remarks

- 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

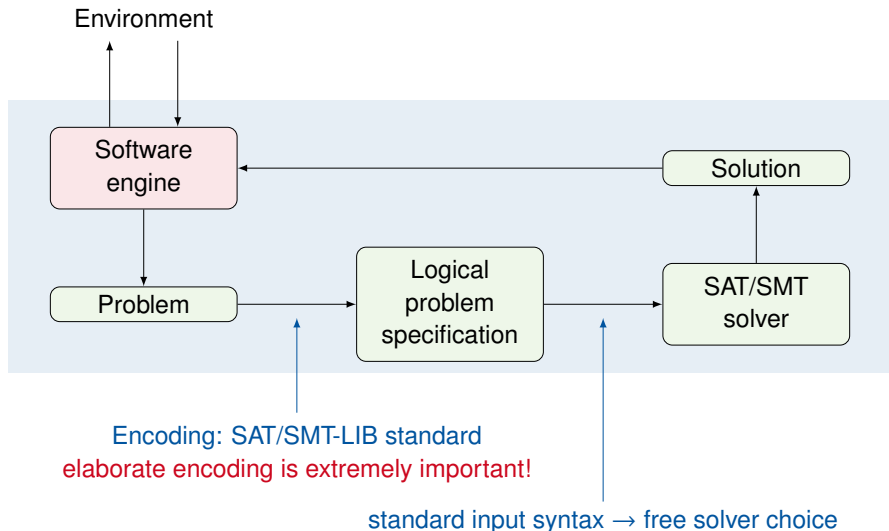


Embedding SAT/SMT solvers

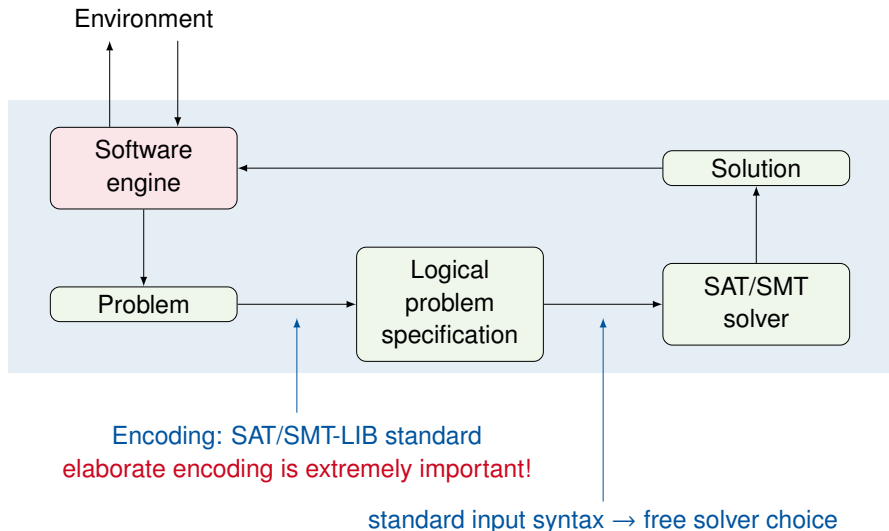


Encoding: SAT/SMT-LIB standard
elaborate encoding is extremely important!

Embedding SAT/SMT solvers



Embedding SAT/SMT solvers



In the following: applications of **SMT** solvers

Bounded model checking for C/C++ [Kroening and Tautschnig, 2014]



**Bounded Model Checking
for Software**



CBMC About CBMC

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.

Source: D. Kroening. **CBMC home page**. <http://www.cprover.org/cbmc/>

Bounded model checking for C/C++ [Kroening and Tautschnig, 2014]



Bounded Model Checking
for Software



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.

Source: D. Kroening. **CBMC home page**. <http://www.cprover.org/cbmc/>

Bounded model checking for C/C++ [Kroening and Tautschnig, 2014]



Bounded Model Checking
for Software



CBMC
About CBMC

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](#).



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

tions 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.

Source: D. Kroening. **CBMC home page**. <http://www.cprover.org/cbmc/>

Bounded model checking for C/C++ [Kroening and Tautschnig, 2014]



Bounded Model Checking
for Software



CBMC
About CBMC

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](#).



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

tions and user-specified assertions. Furthermore, it can check C and C++ for consistency with other languages, such as Verilog. The verification passing th

While CBMC using ma

CBMC is a Solaris 11

CBMC co alternative

solvers we recommend are (in no particular order) [Boolector](#), [MathSAT](#), [ices 2](#) and [Z3](#). Note that these solvers need to be installed separately and have different licensing conditions.

Application examples:

Error localisation and explanation

Equivalence checking

Test case generation

Worst-case execution time



location

edora),

As an 3. The

Source: D. Kroening. **CBMC home page**. <http://www.cprover.org/cbmc/>

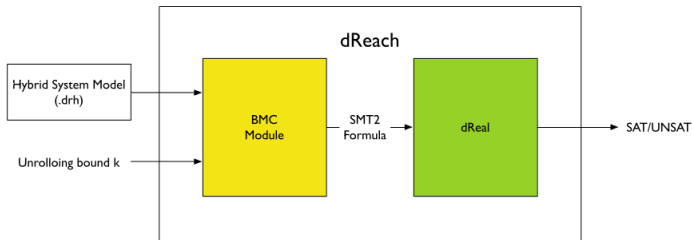
Hybrid systems reachability analysis [Kong et al., 2015]



[DREAL](#) [DREACH](#) [BENCHMARKS](#) [PUBLICATION](#) [DOWNLOAD](#) [TRY ONLINE](#) [PEOPLE](#)

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.



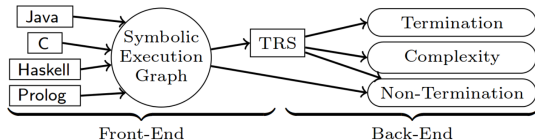
Source: D. Bryce, J. Sun, P. Zuliani, Q. Wang, S. Gao, F. Shmarov, S. Kong, W. Chen, Z. Tavares.

dReach home page. <http://dreal.github.io/dReach/>

Termination analysis for programs [Ströder et al., 2015]

AProVE

Automated Program Verification Environment



Source: T. Ströder, C. Aschermann, F. Frohn, J. Hensel, J. Giesl.

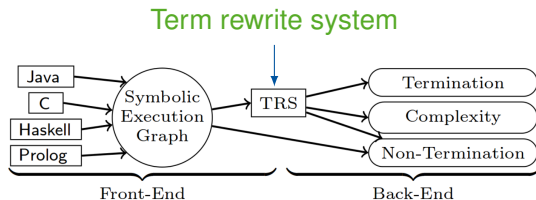
AProVE: Termination and memory safety of C programs (competition contribution).

In Proc. TACAS'15.

Termination analysis for programs [Ströder et al., 2015]

AProVE

Automated Program Verification Environment



Source: T. Ströder, C. Aschermann, F. Frohn, J. Hensel, J. Giesl.

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

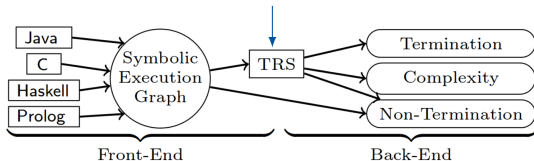
In Proc. TACAS'15.

Termination analysis for programs [Ströder et al., 2015]

APROVE

Automated Program Verification Environment

Term rewrite system



Term rewrite system

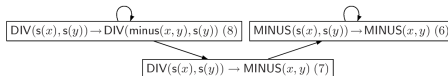


Dependency pairs



Chains

$\text{minus}(x, 0) \rightarrow x$	(1)	$\text{div}(0, s(y)) \rightarrow 0$	(4)
$\text{minus}(0, s(y)) \rightarrow 0$	(2)	$\text{div}(s(x), s(y)) \rightarrow s(\text{div}(\text{minus}(x, y), s(y)))$	(5)
$\text{minus}(s(x), s(y)) \rightarrow \text{minus}(x, y)$	(3)		
$\text{MINUS}(s(x), s(y)) \rightarrow \text{MINUS}(x, y)$	(6)	$\text{DIV}(s(x), s(y)) \rightarrow \text{MINUS}(x, y)$	(7)
		$\text{DIV}(s(x), s(y)) \rightarrow \text{DIV}(\text{minus}(x, y), s(y))$	(8)



Logical encoding for well-founded orders.

Source: T. Ströder, C. Aschermann, F. Frohn, J. Hensel, J. Giesl.

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

In Proc. TACAS'15.

jUnit_{RV}: 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

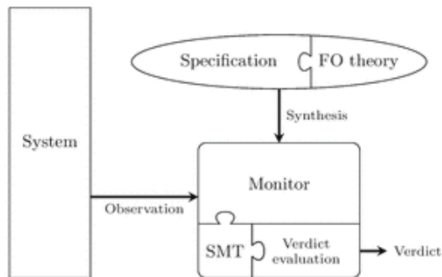


Fig. 1 Schematic overview of the monitoring approach

Source: N. Decker, M. Leucker, D. Thoma.

Monitoring modulo theories.

International Journal on Software Tools for Technology Transfer, 18(2):205-225, April 2016.

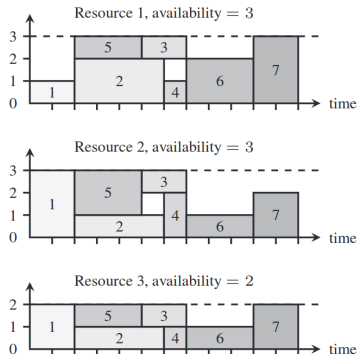
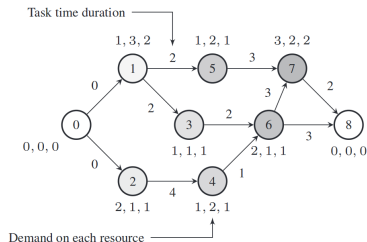


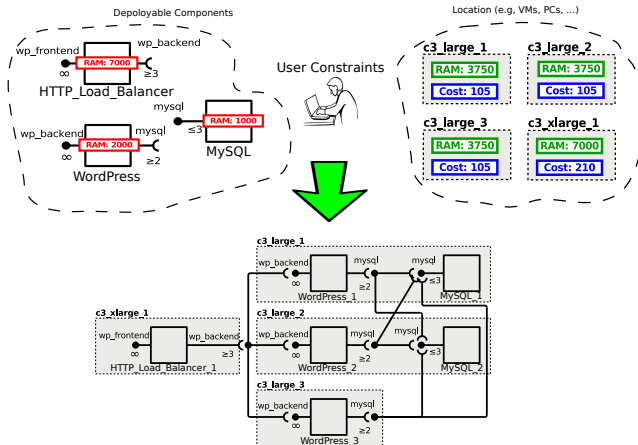
Figure 1: An example of RCPSP (Liess and Michelon 2008)

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]

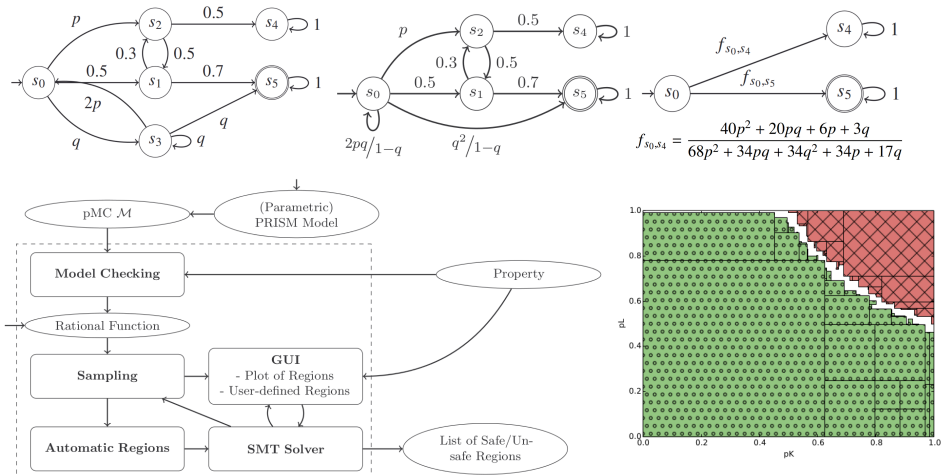


Source: E. Ábrahám, F. Corzilius, E. Broch Johnsen, G. Kremer, J. Mauro.

Zephyrus2: On the fly deployment optimization using SMT and CP technologies.

SETTA'16.

Parameter synthesis for probabilistic systems [Dehnert et al., 2015]



Source: C. Dehnert, S. Junges, N. Jansen, F. Corzilius, M. Volk, H. Bruinjtes, J.-P. Katoen, E. Ábrahám.

PRoPhESY: A probabilistic parameter synthesis tool.

In Proc. of CAV'15.

SMT solving

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

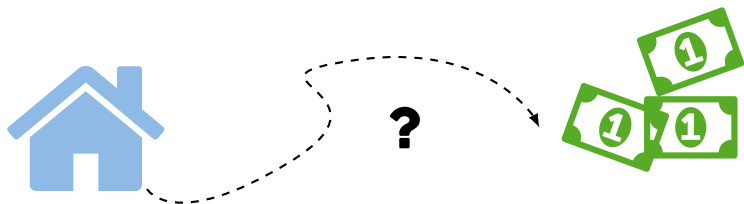
SMT solving for planning

- IV SMT and planning
- V Application: optimal planning with OMT

Concluding remarks

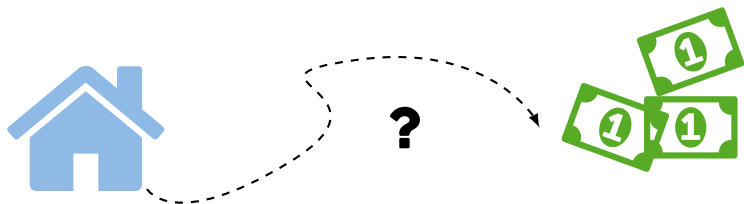
From planning to satisfiability checking

Classical planning



From planning to satisfiability checking

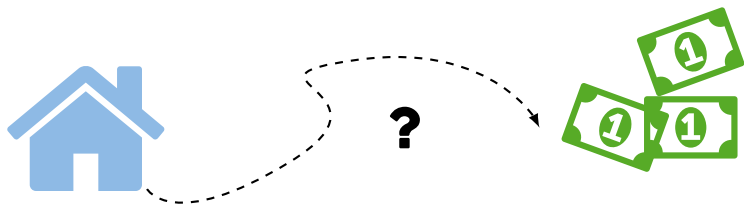
Classical planning



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

From planning to satisfiability checking

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 \mathcal{F} and \mathcal{A} be the sets of *fluents* and *actions*.

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

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

- $I(\mathcal{F})$ represents the set of *initial* states
- $T(\mathcal{X}, \mathcal{X}')$ describes how actions *affect* states
- $G(\mathcal{F})$ represents the set of *goal* states

Encoding Π in SAT - renaming

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

Furthermore, let

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

Encoding Π in SAT - renaming

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

Furthermore, let

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

Encoding Π in SAT - the formula

The planning problem Π with makespan k is the formula

$$\varphi_{(\Pi, k)} := I(\mathcal{X}_0) \wedge \bigwedge_{i=0}^{k-1} T(\mathcal{X}_i, \mathcal{X}_{i+1}) \wedge G(\mathcal{X}_k)$$

Encoding Π 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.

Encoding planning problems: how to?

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

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)

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)

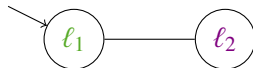
Disclaimer

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

Working example: TSP

Problem statement

- Set of locations: ℓ_1, ℓ_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

```
(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

```
(define (problem tsp-2)
  (:domain tsp)
  (:objects 11 12 )
  (:init
   (at 11))
  (:goal
   (and (visited 11) (visited 12))))
```

Syntax of core theory

```
:sorts ((Bool 0))
:fun (
  (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))
:fun (
...
(+ 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>

Boolean example

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
```

Propositional encoding - III

```
; 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 φ is unsat...we would like to know why!

* More here: [Liffiton and Sakallah, 2008]

Unsat cores - intuition*

Let's assume our input formula φ is unsat...we would like to know why!

Recall: φ 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]

Unsat cores - intuition*

Let's assume our input formula φ is unsat...we would like to know why!

Recall: φ 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

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

* More here: [Liffiton and Sakallah, 2008]

Unsat cores - intuition*

Let's assume our input formula φ is unsat...we would like to know why!


Recall: φ 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

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

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]

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))
```

Propositional encoding - III

```
...  
  
; 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))))
```

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)))
```

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 - I

```
; 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) ...))
```

OMT encoding - II

...

```
(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)
;)
```

SMT solving

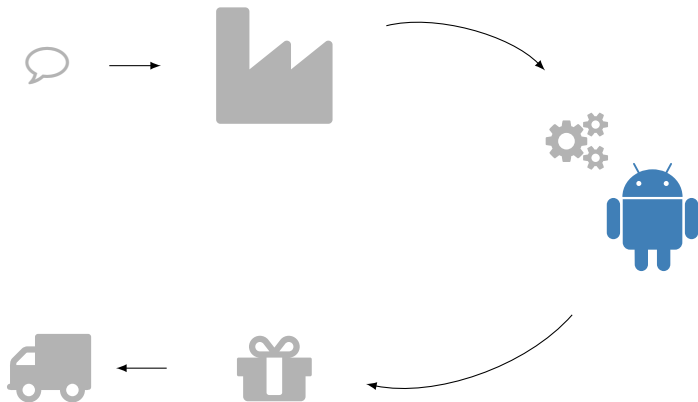
- I Historical notes
- II SAT and SMT solving
- III Some applications outside planning

SMT solving for planning

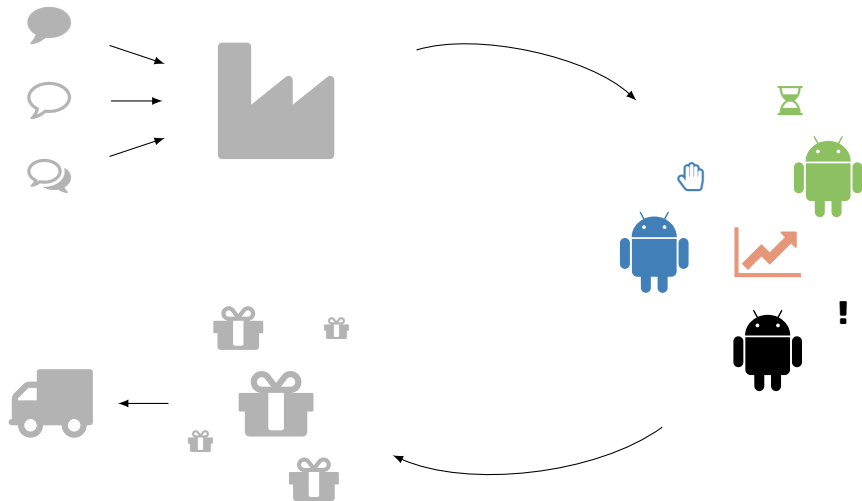
- IV SMT and planning
- V Application: optimal planning with OMT

Concluding remarks

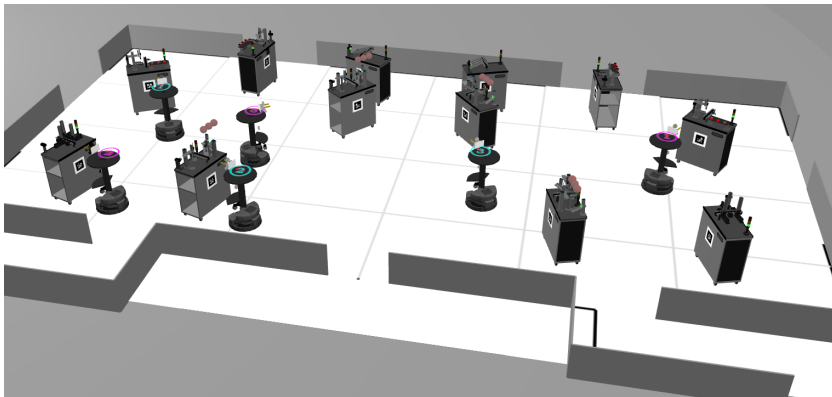
Planning in the era of *Smart Factories*



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].

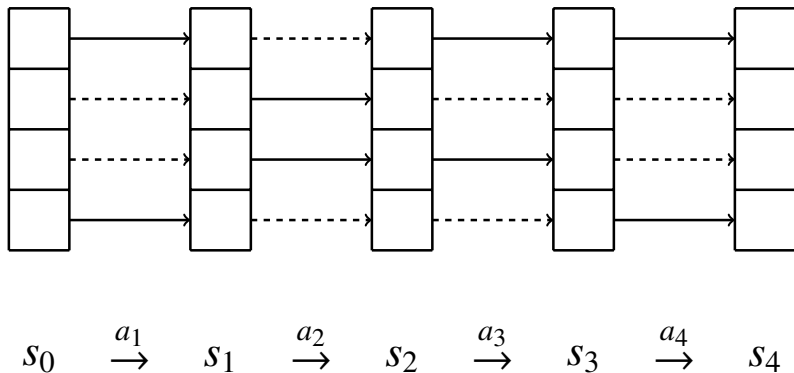
Planning & Execution Competition for Logistics Robots in Simulation [Niemueller et al., 2015]

Temporal planning with OMT for the RCLL. What's hard?

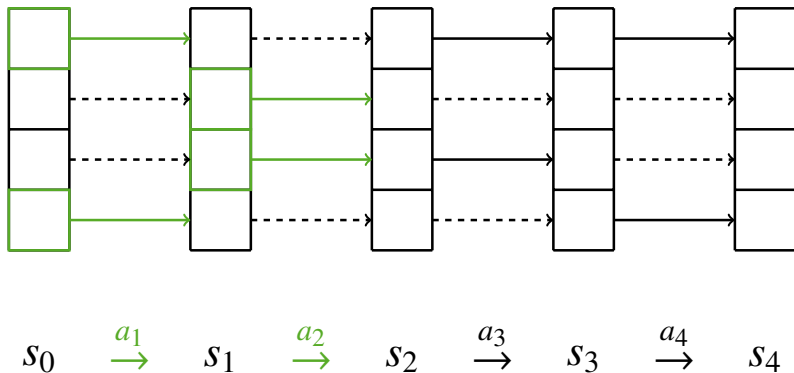
- time windows
- domain representation: over 250 configurations possible!
- combinatorics
- scalability

 compact representations are needed to help solvers

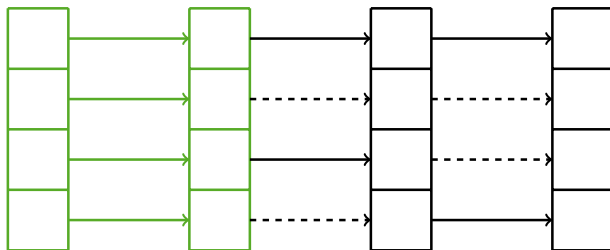
The need for compact encodings



The need for compact encodings

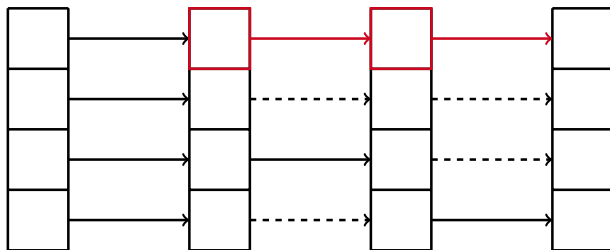


The need for compact encodings



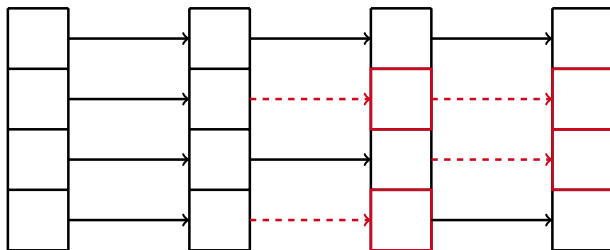
$$s_0 \xrightarrow{a_1 \wedge a_2} s_1 \xrightarrow{a_3} s_2 \xrightarrow{a_4} s_3$$

The need for compact encodings



$$s_0 \xrightarrow{a_1 \wedge a_2} s_1 \xrightarrow{a_3} s_2 \xrightarrow{a_4} s_3$$

The need for compact encodings



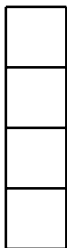
$$s_0 \xrightarrow{a_1 \wedge a_2} s_1 \xrightarrow{a_3} s_2 \xrightarrow{a_4} s_3$$

The need for compact encodings



A reduced encoding [Leofante et al., 2018]

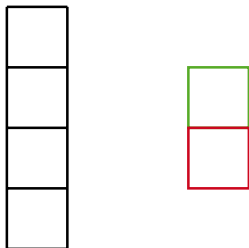
A reduced encoding [Leofante et al., 2018]



A reduced encoding [Leofante et al., 2018]



A reduced encoding [Leofante et al., 2018]



A reduced encoding [Leofante et al., 2018]



s_0



s_1



s_2

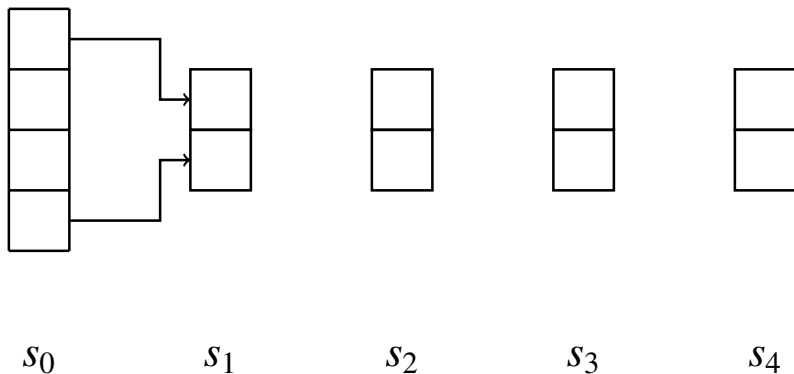


s_3

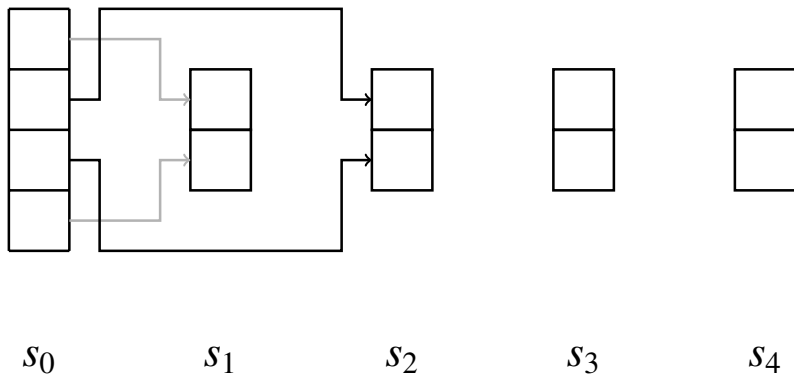


s_4

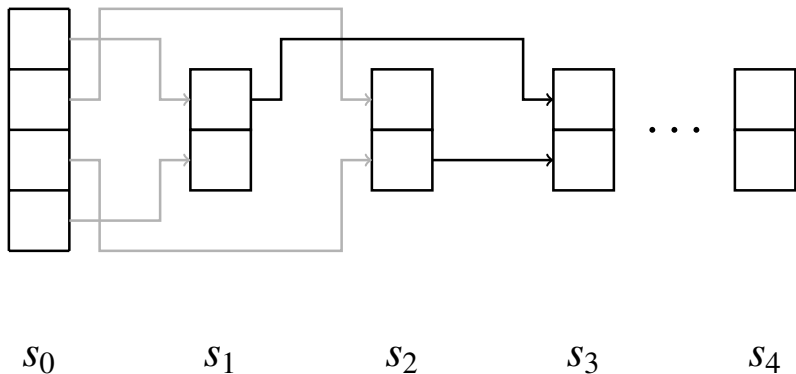
A reduced encoding [Leofante et al., 2018]



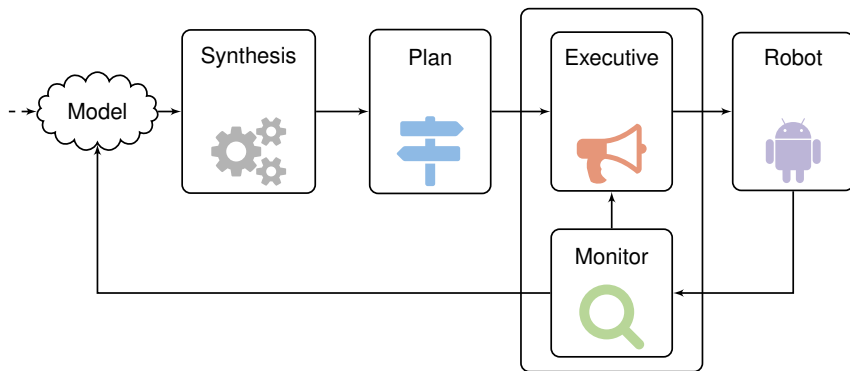
A reduced encoding [Leofante et al., 2018]



A reduced encoding [Leofante et al., 2018]



Integrated synthesis and execution



A reduced encoding - towards a general approach

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?”

(The New York Times)

A reduced encoding - towards a general approach

We implemented a domain-specific planner for the RCLL

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

(The New York Times)



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



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!

References I



zchaff webpage.

<https://www.princeton.edu/~chaff/zchaff.html>.

Accessed: 2018-06-13.



Ábrahám, E., Corzilius, F., Johnsen, E. B., Kremer, G., and Mauro, J. (2016).

Zephyrus2: On the fly deployment optimization using SMT and CP technologies.

In Dependable Software Engineering: Theories, Tools, and Applications - Second International Symposium, SETTA 2016, Beijing, China, November 9-11, 2016, Proceedings, pages 229–245.



Ansótegui, C., Bofill, M., Palahí, M., Suy, J., and Villaret, M. (2011).

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

In Proceedings of the Ninth Symposium on Abstraction, Reformulation, and Approximation, SARA 2011, Parador de Cardona, Cardona, Catalonia, Spain, July 17-18, 2011.



Cashmore, M., Fox, M., Long, D., and Magazzeni, D. (2016).

A compilation of the full PDDL+ language into SMT.

In Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016., pages 79–87.

References II



Cimatti, A., Griggio, A., Schaafsma, B. J., and Sebastiani, R. (2013).

The mathsat5 SMT solver.

In Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings, pages 93–107.



Cook, S. A. (1971).

The complexity of theorem-proving procedures.

In Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA, pages 151–158.



Corzilius, F., Kremer, G., Junges, S., Schupp, S., and Ábrahám, E. (2015).

SMT-RAT: an open source C++ toolbox for strategic and parallel SMT solving.

In Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings, pages 360–368.



Corzilius, F., Loup, U., Junges, S., and Ábrahám, E. (2012).

SMT-RAT: an smt-compliant nonlinear real arithmetic toolbox - (tool presentation).

In Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings, pages 442–448.

References III



de Moura, L. M. and Bjørner, N. (2008).

Z3: an efficient SMT solver.

In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, pages 337–340.



Decker, N., Leucker, M., and Thoma, D. (2016).

Monitoring modulo theories.

STTT, 18(2):205–225.



Dehnert, C., Junges, S., Jansen, N., Corzilius, F., Volk, M., Brientjes, H., Katoen, J., and Ábrahám, E. (2015).

Prophecy: A probabilistic parameter synthesis tool.

In *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, pages 214–231.



Deters, M., Reynolds, A., King, T., Barrett, C. W., and Tinelli, C. (2014).

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

In *Formal Methods in Computer-Aided Design, FMCAD 2014, Lausanne, Switzerland, October 21-24, 2014*, page 7.

References IV



Dutertre, B. (2014).

Yices 2.2.

In Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings, pages 737–744.



Giesl, J., Thiemann, R., Schneider-Kamp, P., and Falke, S. (2004).

Automated termination proofs with approve.

In Rewriting Techniques and Applications, 15th International Conference, RTA 2004, Aachen, Germany, June 3-5, 2004, Proceedings, pages 210–220.



Giunchiglia, E. (2000).

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

In KR 2000, Principles of Knowledge Representation and Reasoning Proceedings of the Seventh International Conference, Breckenridge, Colorado, USA, April 11-15, 2000., pages 657–666.



Giunchiglia, E. and Maratea, M. (2007).

Planning as satisfiability with preferences.

In Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada, pages 987–992.

References V



Järvisalo, M., Berre, D. L., Roussel, O., and Simon, L. (2012).
The international SAT solver competitions.
AI Magazine, 33(1).



Kautz, H. A., McAllester, D. A., and Selman, B. (1996).
Encoding plans in propositional logic.
In Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96), Cambridge, Massachusetts, USA, November 5-8, 1996., pages 374–384.



Kautz, H. A. and Selman, B. (1992).
Planning as satisfiability.
In ECAI, pages 359–363.



Kong, S., Gao, S., Chen, W., and Clarke, E. M. (2015).
dreach: δ -reachability analysis for hybrid systems.
In Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings, pages 200–205.

References VI



Kroening, D. and Tautschnig, M. (2014).

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

In Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings, pages 389–391.



Leofante, F., Ábrahám, E., Niemueller, T., Lakemeyer, G., and Tacchella, A. (2018).

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



Liffiton, M. H. and Sakallah, K. A. (2008).

Algorithms for computing minimal unsatisfiable subsets of constraints.
J. Autom. Reasoning, 40(1):1–33.



Niemetz, A., Preiner, M., and Biere, A. (2014).

Boolector 2.0.

JSAT, 9:53–58.



Niemueller, T., Lakemeyer, G., and Ferrein, A. (2015).

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



RCLL Technical Committee (2017).

RoboCup Logistics League – Rules and regulations 2017.

References VII



Rintanen, J. (2009).
Planning and SAT.
In Handbook of Satisfiability, pages 483–504.



Rintanen, J. (2015).
Discretization of temporal models with application to planning with SMT.
In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA., pages 3349–3355.



Rintanen, J. (2017).
Temporal planning with clock-based SMT encodings.
In Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017, pages 743–749.



Rintanen, J., Heljanko, K., and Niemelä, I. (2006).
Planning as satisfiability: parallel plans and algorithms for plan search.
Artif. Intell., 170(12-13):1031–1080.



SATO webpage.
SATO solver.
<http://homepage.divms.uiowa.edu/~hzhang/sato/>.
Accessed: 2018-06-13.

References VIII



Scala, E., Ramírez, M., Haslum, P., and Thiébaux, S. (2016).
Numeric planning with disjunctive global constraints via SMT.
In Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016., pages 276–284.



Shin, J. and Davis, E. (2005).
Processes and continuous change in a sat-based planner.
Artif. Intell., 166(1-2):194–253.



Ströder, T., Aschermann, C., Frohn, F., Hensel, J., and Giesl, J. (2015).
Aprove: Termination and memory safety of C programs - (competition contribution).
In Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings, pages 417–419.



Wolfman, S. A. and Weld, D. S. (1999).
The LPSAT engine & its application to resource planning.
In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages, pages 310–317.



Zankl, H. and Middeldorp, A. (2010).

Satisfiability of non-linear (ir)rational arithmetic.

In Logic for Programming, Artificial Intelligence, and Reasoning - 16th International Conference, LPAR-16, Dakar, Senegal, April 25-May 1, 2010, Revised Selected Papers, pages 481–500.



Zwilling, F., Niemueller, T., and Lakemeyer, G. (2014).

Simulation for the RoboCup Logistics League with real-world environment agency and multi-level abstraction.

In Robot Soccer World Cup, pages 220–232. Springer.