Satisfiability Checking

Overview

Prof. Dr. Erika Ábrahám

RWTH Aachen University
Informatik 2
LuFG Theory of Hybrid Systems

WS 16/17

Literature

- Daniel Kroening and Ofer Strichman.
  Decision Procedures: An Algorithmic Point of View.
  Springer-Verlag, Berlin, 2008.
- Slides
- Video recordings from previous years
- Selected papers

Organization

- Language: English or German
- Lecture (V3): Monday 14:15-15:45, room AH I
  Wednesday 12:15-13:00, room AH I

  Registration in L^2P learning room via Campus required.

  All materials are available in the learning room.
- Exercise (Ü1): Wednesday, 13:00-13:45 room AH I, after the lecture

  Exercise sheets are distributed on Wednesday, and are due to Wednesday one week later.
- Exam: written

  Exercise solutions are no entrance requirement, but they are strongly recommended.

  Mandatory online tests in L^2P.
- Assistant: Gereon Kremer `gereon.kremer@cs.rwth-aachen.de`

What is this talk about?

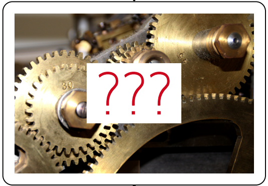
Quantifier-free logical formula

Solver

Satisfiability of the input formula

What is this talk about?

Quantifier-free logical formula



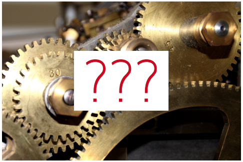
Solver

Satisfiability of the input formula

What is this talk about?

???

Quantifier-free logical formula



Solver

Satisfiability of the input formula

???

The Boolean satisfiability problem...

Satisfiability problem for propositional logic

Given a formula combining some atomic propositions
using the Boolean operators "and" ($\wedge$), "or" ($\vee$) and "not" ($\neg$),
decide whether we can substitute truth values for the propositions
such that the formula evaluates to true.

Example

Formula:

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

Satisfying assignment:

$$a = true, \quad b = false, \quad c = true$$

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

...and its extension to theories

Satisfiability modulo theories problem (informal)

Given a Boolean combination of constraints from some theories, decide whether we can substitute (type-correct) values for the (theory) variables such that the formula evaluates to true.

A non-linear real arithmetic example

Formula:

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

Satisfying assignment:

$$x = \sqrt{2}, \quad y = 2$$

Hard problems... non-linear integer arithmetic is even undecidable.

What is formal logic?

- A (formal) logic

What is formal logic?

- A (formal) logic defines a framework for inference and correct reasoning.

What is formal logic?

- A (formal) logic defines a framework for inference and correct reasoning.
- Studied in, e.g.,

What is formal logic?

- A (formal) logic defines a framework for inference and correct reasoning.
- Studied in, e.g., philosophy, mathematics, computer science.

What is formal logic?

- A (formal) logic defines a framework for inference and correct reasoning.
- Studied in, e.g., philosophy, mathematics, computer science.
- A logical system defines

What is formal logic?

- A (formal) logic defines a framework for inference and correct reasoning.
- Studied in, e.g., philosophy, mathematics, computer science.
- A logical system defines
  - the form of logical formulas (syntax) and

What is formal logic?

- A (formal) logic defines a framework for inference and correct reasoning.
- Studied in, e.g., philosophy, mathematics, computer science.
- A logical system defines
  - the form of logical formulas (syntax) and
  - a set of axioms and inference rules.

What is formal logic?

- A (formal) logic defines a framework for inference and correct reasoning.
- Studied in, e.g., philosophy, mathematics, computer science.
- A logical system defines
  - the form of logical formulas (syntax) and
  - a set of axioms and inference rules.
- What is the value of a logical formula?

What is formal logic?

- A (formal) logic defines a framework for inference and correct reasoning.
- Studied in, e.g., philosophy, mathematics, computer science.
- A logical system defines
  - the form of logical formulas (syntax) and
  - a set of axioms and inference rules.
- What is the value of a logical formula?
  - A structure for a logical system gives meaning (semantics) to the formulas.
  - The logical system allows to derive the meaning of formulas.

What is formal logic?

- A (formal) logic defines a framework for inference and correct reasoning.
- Studied in, e.g., philosophy, mathematics, computer science.
- A logical system defines
  - the form of logical formulas (syntax) and
  - a set of axioms and inference rules.
- What is the value of a logical formula?
  - A structure for a logical system gives meaning (semantics) to the formulas.
  - The logical system allows to derive the meaning of formulas.
- Important properties of logical systems:

What is formal logic?

- A (formal) logic defines a framework for inference and correct reasoning.
- Studied in, e.g., philosophy, mathematics, computer science.
- A logical system defines
  - the form of logical formulas (syntax) and
  - a set of axioms and inference rules.
- What is the value of a logical formula?
  - A structure for a logical system gives meaning (semantics) to the formulas.
  - The logical system allows to derive the meaning of formulas.
- Important properties of logical systems:
  - consistency
  - soundness
  - completeness

Historical view on logic

Historical development goes from

informal logic (natural language arguments) to

formal logic (formal language arguments)

Historical view on logic

Historical development goes from

informal logic (natural language arguments) to

formal logic (formal language arguments)

- Philosophical logic
  - 500 BC to 19th century
- Symbolic logic
  - Mid to late 19th century
- Mathematical logic
  - Late 19th to mid 20th century
- Logic in computer science

Historical view on logic

Historical development goes from

informal logic (natural language arguments) to

formal logic (formal language arguments)

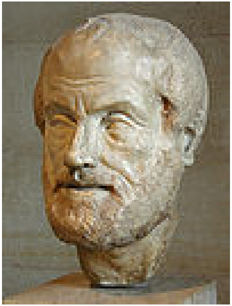
- Philosophical logic
  - 500 BC to 19th century
- Symbolic logic
  - Mid to late 19th century
- Mathematical logic
  - Late 19th to mid 20th century
- Logic in computer science

Philosophical logic

- 500 B.C - 19th century
- Logic dealing with sentences in the natural language used by humans.
- Example
  - All men are mortal.
  - Socrates is a man.
  - Therefore, Socrates is mortal.

Philosophical logic

- Natural languages are very ambiguous.
- Aristotle (384 BC – 322 BC) identified 13 types of fallacies in his *Sophistical Refutations*.



Fallacies

The fallacy of composition arises when one infers that something is true of the whole from the fact that it is true of some part of the whole.

Fallacies

The fallacy of composition arises when one infers that something is true of the whole from the fact that it is true of some part of the whole.

1. Human cells are invisible to the naked eye.
2. Humans are made up of human cells.
3. Therefore, humans are invisible to the naked eye.

Fallacies

A fallacy of division occurs when one reasons logically that something true of a thing must also be true of all or some of its parts.

Fallacies

A fallacy of division occurs when one reasons logically that something true of a thing must also be true of all or some of its parts.

> Famously and controversially, in the Greek philosophy it was assumed that the atoms constituting a substance must themselves have the properties of that substance: so atoms of water would be wet, atoms of iron would be hard, atoms of wool would be soft, etc.

Fallacies

A figure of speech is the use of a word or words diverging from its usual meaning.

Fallacies

A figure of speech is the use of a word or words diverging from its usual meaning.

I had butterflies in my stomach.

Fallacies

Affirming the consequent is a formal fallacy, committed by reasoning in the form:

1. If P, then Q.
2. Q.
3. Therefore, P.

Fallacies

Affirming the consequent is a formal fallacy, committed by reasoning in the form:

1. If P, then Q.
2. Q.
3. Therefore, P.

> 1. If I have the flu, then I have a sore throat.
> 2. I have a sore throat.
> 3. Therefore, I have the flu.

Other natural language issues

Besides such fallacies, natural languages allow to argue about the language itself.

Other natural language issues

Besides such fallacies, natural languages allow to argue about the language itself.

This sentence is a lie. *(The liar's paradox)*

Other natural language issues

Besides such fallacies, natural languages allow to argue about the language itself.

$\boxed{\text{This sentence is a lie. \textit{(The liar's paradox)}}}$

$\rightarrow$ inconsistency

Other natural language issues

Besides such fallacies, natural languages allow to argue about the language itself.

This sentence is a lie. *(The liar's paradox)*

$\rightarrow$ inconsistency

Rules for connecting language constructs are not working the expected way:

Other natural language issues

Besides such fallacies, natural languages allow to argue about the language itself.

$\boxed{\text{This sentence is a lie. \textit{(The liar's paradox)}}}$

$\rightarrow$ inconsistency

Rules for connecting language constructs are not working the expected way:

$\boxed{\text{This sectence has five words.}}$

Other natural language issues

Besides such fallacies, natural languages allow to argue about the language itself.

This sentence is a lie. *(The liar's paradox)*

$\rightarrow$ inconsistency

Rules for connecting language constructs are not working the expected way:

This sectence has five words.

This sentence has five words and this sectence has five words.

Other natural language issues

Besides such fallacies, natural languages allow to argue about the language itself.

This sentence is a lie. *(The liar's paradox)*

$\rightarrow$ inconsistency

Rules for connecting language constructs are not working the expected way:

This sectence has five words.

This sentence has five words and this sectence has five words.

$\rightarrow$ The conjunction of two true sentences is not always true.

Historical view on logic

Historical development goes from

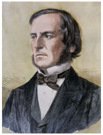
informal logic (natural language arguments) to

formal logic (formal language arguments)

- Philosophical logic
  - 500 BC to 19th century
- Symbolic logic
  - Mid to late 19th century
- Mathematical logic
  - Late 19th to mid 20th century
- Logic in computer science

Symbolic and mathematical logic

- 1854: George Boole introduced symbolic logic and the principles of what is now known as Boolean logic.
- 1879: Gottlob Frege created with his *Begriffsschrift* the basis of modern logic with the invention of quantifier notation.
- 1910-1913: Alfred Whitehead and Bertrand Russell published *Principia Mathematica* on the foundations of mathematics, attempting to derive mathematical truths from axioms and inference rules in symbolic logic.
- 1931: Gödel's and Turing's undecidability results (we will deal with them later).

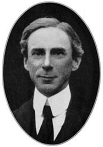


George Boole
(1815-1864)

Gottlob Frege
(1848-1925)

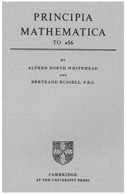
Symbolic and mathematical logic

- 1854: George Boole introduced symbolic logic and the principles of what is now known as Boolean logic.
- 1879: Gottlob Frege created with his *Begriffsschrift* the basis of modern logic with the invention of quantifier notation.
- 1910-1913: Alfred Whitehead and Bertrand Russell published *Principia Mathematica* on the foundations of mathematics, attempting to derive mathematical truths from axioms and inference rules in symbolic logic.
- 1931: Gödel's and Turing's undecidability results (we will deal with them later).



Bertrand Russell (1872-1970)

Alfred Whitehead (1861-1947)



PRINCIPIA
MATHEMATICA
TO *56

BY
ALFRED NORTH WHITEHEAD
AND
BERTRAND RUSSELL, F.R.S.

CAMBRIDGE
AT THE UNIVERSITY PRESS

Prinzipia Mathematica

$*54{\cdot}43.\quad \vdash :. \alpha, \beta \in 1 . \supset : \alpha \cap \beta = \Lambda . \equiv . \alpha \cup \beta \in 2$

Dem.

$\vdash . *54{\cdot}26 . \supset \vdash :. \alpha = \iota\text{‘}x . \beta = \iota\text{‘}y . \supset : \alpha \cup \beta \in 2 . \equiv . x \neq y .$

$[*51{\cdot}231] \qquad \equiv . \iota\text{‘}x \cap \iota\text{‘}y = \Lambda .$

$[*13{\cdot}12] \qquad \equiv . \alpha \cap \beta = \Lambda \qquad (1)$

$\vdash . (1) . *11{\cdot}11{\cdot}35 . \supset$

$\qquad \vdash :. (\exists x, y) . \alpha = \iota\text{‘}x . \beta = \iota\text{‘}y . \supset : \alpha \cup \beta \in 2 . \equiv . \alpha \cap \beta = \Lambda \qquad (2)$

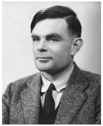
$\vdash . (2) . *11{\cdot}54 . *52{\cdot}1 . \supset \vdash . \text{Prop}$

From this proposition it will follow, when arithmetical addition has been defined, that $1 + 1 = 2$.

Symbolic and mathematical logic

- 1854: George Boole introduced symbolic logic and the principles of what is now known as Boolean logic.
- 1879: Gottlob Frege created with his *Begriffsschrift* the basis of modern logic with the invention of quantifier notation.
- 1910-1913: Alfred Whitehead and Bertrand Russell published *Principia Mathematica* on the foundations of mathematics, attempting to derive mathematical truths from axioms and inference rules in symbolic logic.
- 1931: Gödel's and Turing's undecidability results (we will deal with them later).

Kurt Gödel
(1906-1978)



Alan Turing
(1912-1954)

Historical view on logic

Historical development goes from

informal logic (natural language arguments) to

formal logic (formal language arguments)

- Philosophical logic
  - 500 BC to 19th century
- Symbolic logic
  - Mid to late 19th century
- Mathematical logic
  - Late 19th to mid 20th century
- Logic in computer science

Logic in computer science

Logic has a profound impact on computer science. Some examples:

Logic in computer science

Logic has a profound impact on computer science. Some examples:

- Propositional logic - the foundation of computers and circuits
- Databases - Query languages
- Programming languages (e.g. Prolog)
- Specification and verification
- ...

Logic in computer science

- Propositional logic
- First order logic
- Higher order logic
- Temporal logic
- ...

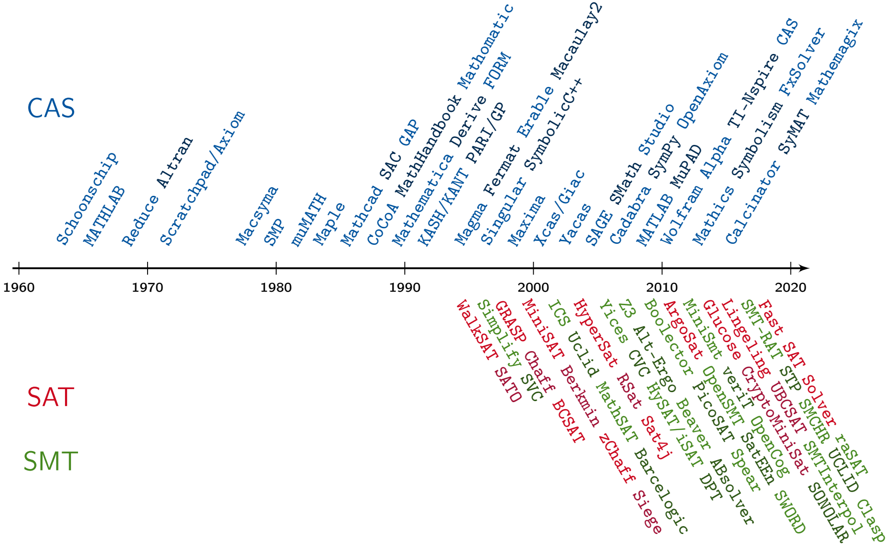
Satisfiability checking: Some milestones

Decision procedures for first-order logic over arithmetic theories in mathematical logic

1940 Computer architecture development

Year	CAS (Symbolic Computation)	SAT (propositional logic)	SMT (SAT modulo theories)
		Enumeration	
1960	Computer algebra systems (CAS)	DP (resolution) DPLL (propagation)	
1970	Gröbner bases CAD (cylindrical algebraic decomposition)	NP-completeness	Decision procedures for combined theories
1980		Conflict-directed backjumping	
	FGLM algorithm Partial CAD Comprehensive Gröbner bases	CDCL	
2000	Virtual substitution	Watched literals Clause learning/forgetting Variable ordering heuristics Restarts	DPLL(T) Equalities Uninterpreted functions Bit-vectors Array theory Arithmetic theories
2010			
2015	Truth table invariant CAD		

Satisfiability checking: Tool development (not exhaustive)



Satisfiability checking for propositional logic

Success story: SAT-solving

- Practical problems with millions of variables are solvable.
- Frequently used in different research areas for, e.g., analysis, synthesis and optimisation.
- Also massively used in industry for, e.g., digital circuit design and verification.

Success story: SAT-solving

- Practical problems with millions of variables are solvable.
- Frequently used in different research areas for, e.g., analysis, synthesis and optimisation.
- Also massively used in industry for, e.g., digital circuit design and verification.

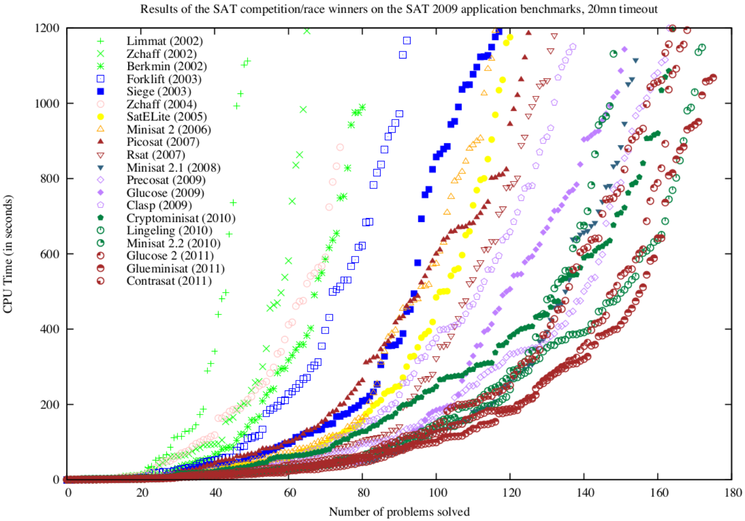
Community support:

- Standardised input language, lots of benchmarks available.
- Competitions since 2002.

  2016 SAT Competition: 6 tracks, 29 solvers in the main track.

  SAT Live! forum as community platform, dedicated conferences, journals, etc.

An impression of the SAT solver development



Source: Jarvisalo, Le Berre, Roussel, Simon. *The International SAT Solver Competitions*. AI Magazine, 2012.

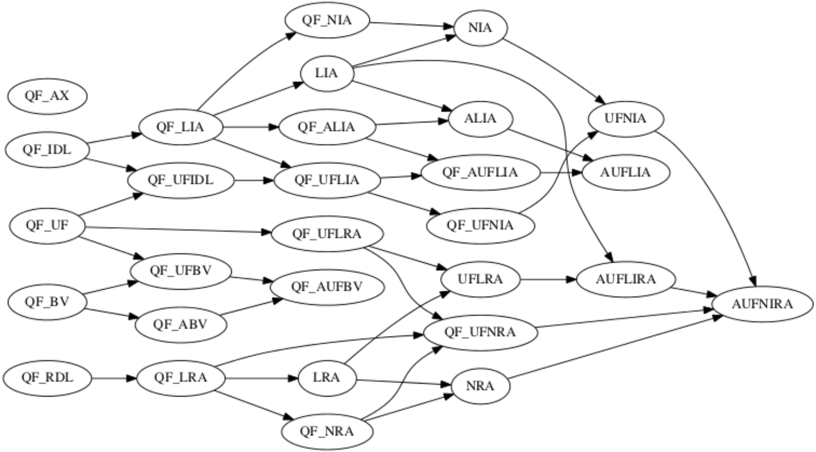
Satisfiability modulo theories solving

- Propositional logic is sometimes too weak for modelling.
- We need more expressive logics and decision procedures for them.
- Logics:
  quantifier-free fragments of first-order logic over various theories.
- Our focus: SAT-modulo-theories (SMT) solving.

Satisfiability modulo theories solving

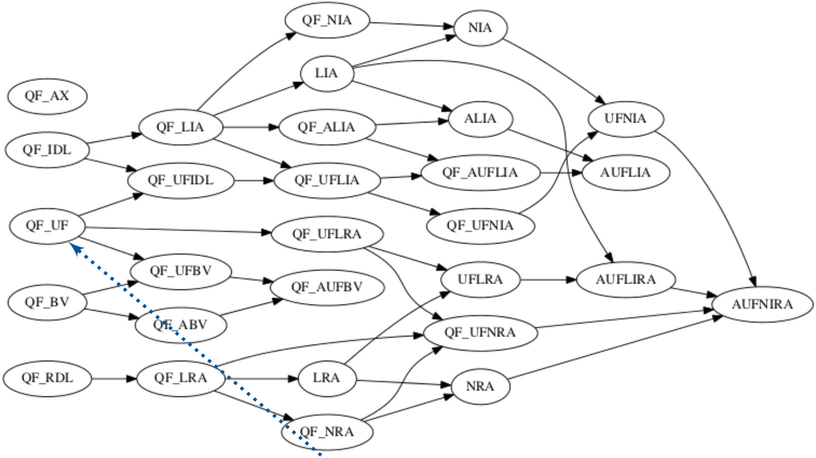
- Propositional logic is sometimes too weak for modelling.
- We need more expressive logics and decision procedures for them.
- Logics:
  quantifier-free fragments of first-order logic over various theories.
- Our focus: SAT-modulo-theories (SMT) solving.
- SMT-LIB as standard input language since 2004.
- Competitions since 2005.
- SMT-COMP 2016 competition:
  - 4 tracks, 41 logical categories.
  - QF linear real arithmetic: $7 + 2$ solvers, 1626 benchmarks.
  - QF linear integer arithmetic: $6 + 2$ solvers, 5839 benchmarks.
  - QF non-linear real arithmetic: $5 + 1$ solvers, 10245 benchmarks.
  - QF non-linear integer arithmetic: $7 + 1$ solvers, 8593 benchmarks.

SMT-LIB theories



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

SMT-LIB theories

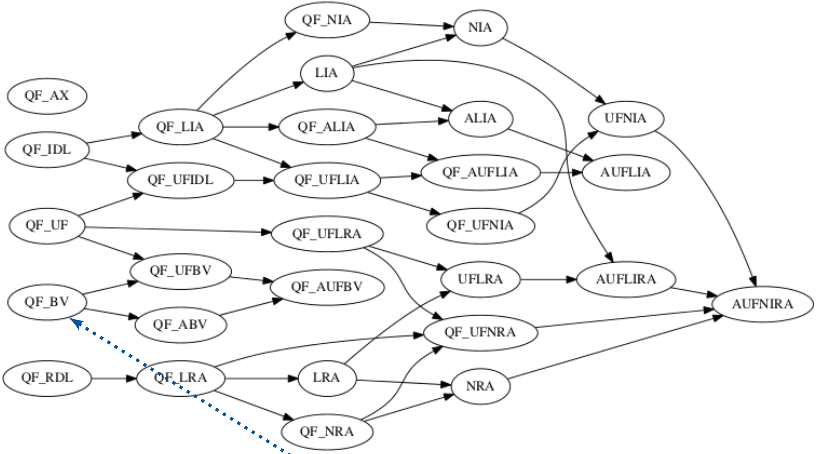


Quantifier-free equality logic with uninterpreted functions

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

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

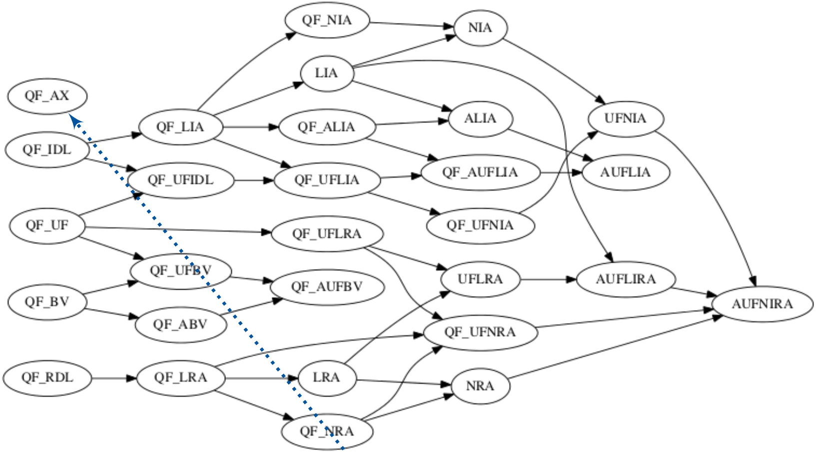
SMT-LIB theories



Quantifier-free bit-vector arithmetic
$a + b \geq 0 \wedge (a|b) \leq (a\&b)$

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

SMT-LIB theories

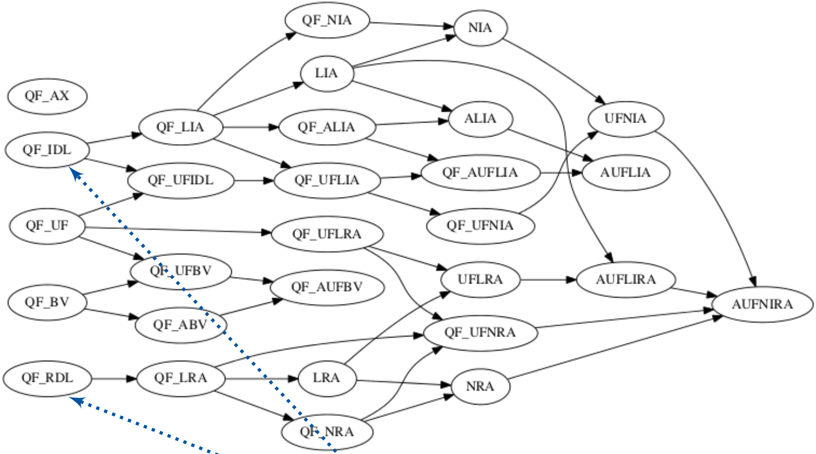


Quantifier-free array theory

$i = j \ \rightarrow \ read(write(a, i, v), j) = v$

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

SMT-LIB theories

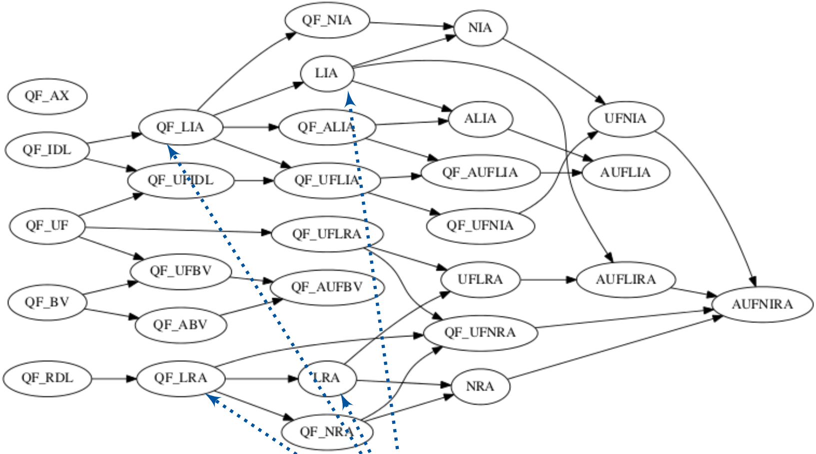


Quantifier-free integer/rational difference logic

$x - y \geq 0 \vee x - z < 0$

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

SMT-LIB theories

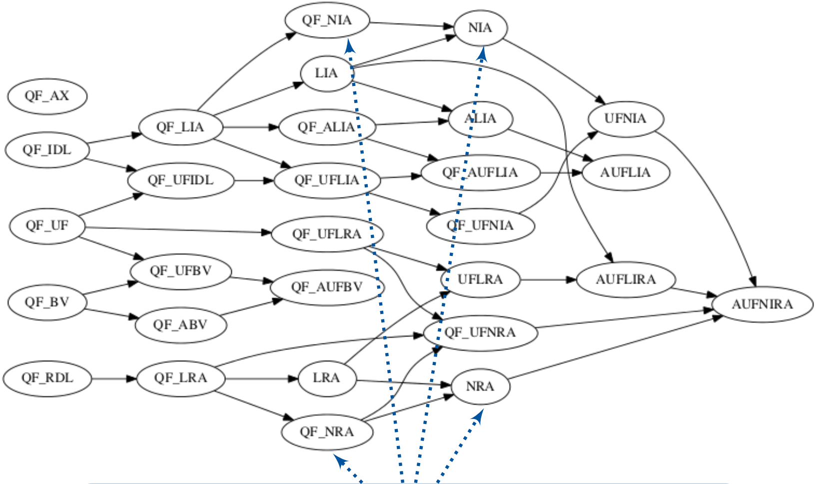


(Quantifier-free) real/integer linear arithmetic

$$4x + 7y = 8 \wedge (y = 0 \vee x > y)$$

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

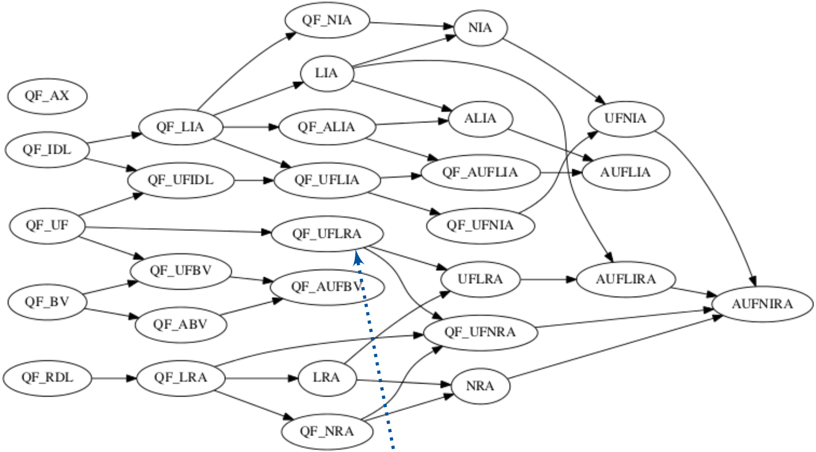
SMT-LIB theories



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

$$x^2 + 2xy + y^2 > 0 \vee (x \geq 1 \wedge xz + yz^2 = 0)$$

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

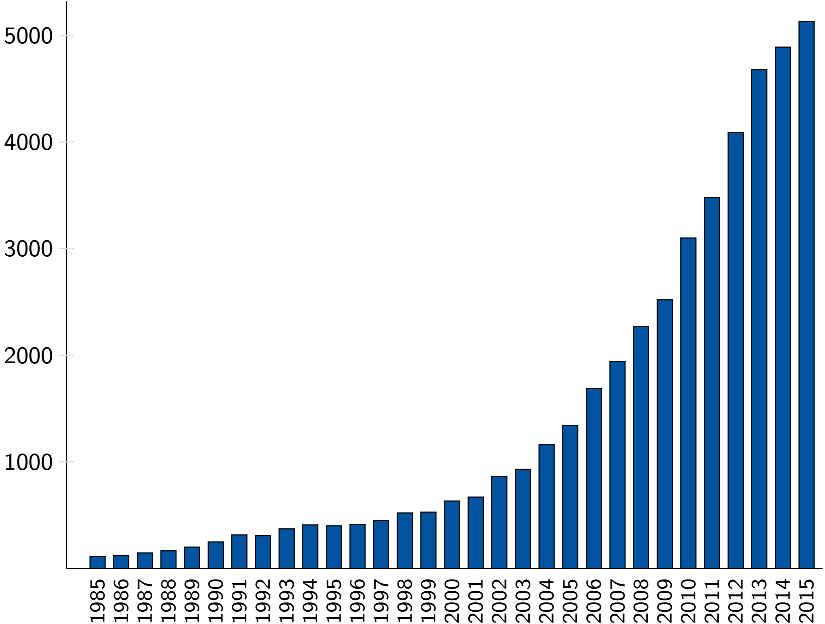


Combined theories

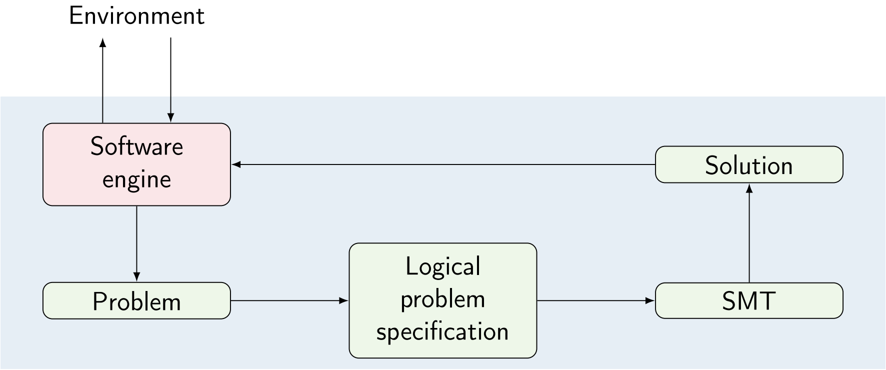
$$2f(x) + 5y > 0 \wedge \neg(f(x) = y \vee x + 2y = 0)$$

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

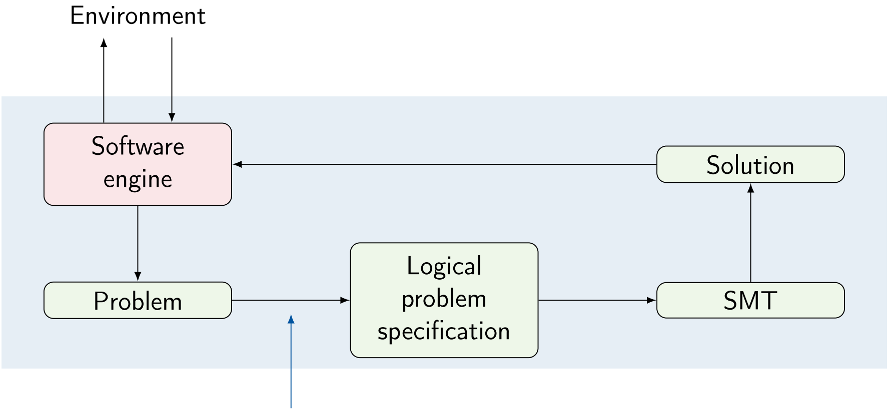
Google Scholar search for "SAT modulo theories"



SAT/SMT embedding structure



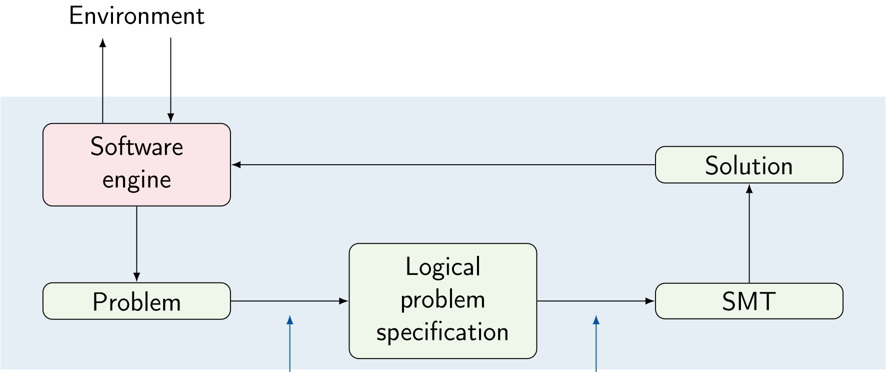
SAT/SMT embedding structure



Encoding: SAT/SMT-LIB standard

elaborate encoding is extremely important!

SAT/SMT embedding structure



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

standard input language $\rightarrow$ free solver choice

Application example: Hardware verification

Problem 1: Given two circuits, are they equivalent?

Problem 2: Given a circuit and a property specification, does the circuit fulfill the specification?

Problem 3: Given a partially specified circuit with a black-box component (at early design stage) and a property specification, is the partial circuit realisable, i.e., is there an implementation of the black box such that the circuit fulfills the property?

Many hardware producers develop and use own SAT solvers for these tasks.

Application example: Symbolic execution

Program 1.2.1 A recursion-free program with bounded loops and an SSA unfolding.

```
int Main(int x, int y)
{
  if (x < y)
     x = x + y;
  for (int i = 0; i < 3; ++i) {
      y = x + Next(y);
  }
  return x + y;
}

int Next(int x) {
  return x + 1;
}
```

```
int Main(int x0, int y0)
{
  int x1;
  if (x0 < y0)
     x1 = x0 + y0;
  else
     x1 = x0;
  int y1 = x1 + y0 + 1;
  int y2 = x1 + y1 + 1;
  int y3 = x1 + y2 + 1;
  return x1 + y3;
}
```

$$\exists x_1, y_1, y_2, y_3 \begin{pmatrix} (x_0 < y_0 \implies x_1 = x_0 + y_0) \ \wedge \ (\neg(x_0 < y_0) \implies x_1 = x_0) \wedge \\ y_1 = x_1 + y_0 + 1 \ \wedge \ y_2 = x_1 + y_1 + 1 \ \wedge \ y_3 = x_1 + y_2 + 1 \ \wedge \\ result = x_1 + y_3 \end{pmatrix}$$

Source: Nikolaj Bjørner and Leonardo de Moura. *Applications of SMT solvers to Program Verification*.

Rough notes for SSFT 2014.

Application example: Bounded model checking

Problem: Given a program (automaton, circuit, term rewrite system, etc.), find an execution path of length at most k which leads to a state with a certain property (used for detecting, e.g., division by zero, violating functional requirements, etc.).

Application example: Bounded model checking for C/C++

Carnegie Mellon

CBMC Homepage

Bounded Model Checking for Software

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/`

Application example: Bounded model checking for C/C++

Carnegie Mellon

CBMC Homepage

Bounded Model Checking for Software

Logical encoding of finite unsafe paths

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/`

Application example: Bounded model checking for C/C++

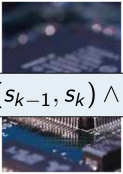
Carnegie Mellon

CBMC Homepage

Bounded Model Checking for Software

Logical encoding of finite unsafe paths

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.

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

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/`

Application example: Bounded model checking for C/C++

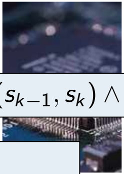
Carnegie Mellon

CBMC Homepage

Bounded Model Checking for Software

Logical encoding of finite unsafe paths

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.

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

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 CBM… ocation using mal…

CBMC is a… edora), Solaris 11, …

CBMC co… As an alternative … 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.

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/`

Application example: BMC for graph transformation systems

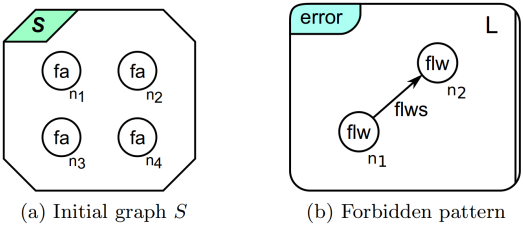


Fig. 1. Part of the car platooning GTS [1]

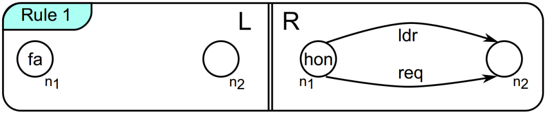


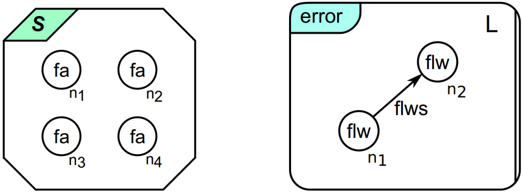
Fig. 2. Rule 1 of the car platooning GTS [1]

Source: T. Isenberg, D. Steenken, and H. Wehrheim.

Bounded Model Checking of Graph Transformation Systems via SMT Solving.

In Proc. FMOODS/FORTE'13.

Application example: BMC for graph transformation systems



(a) Initial graph S (b) Forbidden pattern

Fig. 1. Part of the car platooning GTS [1]

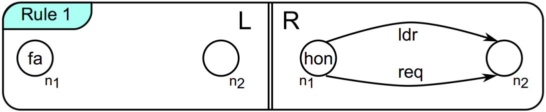


Fig. 2. Rule 1 of the car platooning GTS [1]

Encode initial and forbidden state graphs and
the graph transformation rules in first-order logic.

↓

Apply
bounded model checking

Source: T. Isenberg, D. Steenken, and H. Wehrheim.

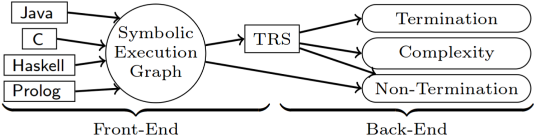
Bounded Model Checking of Graph Transformation Systems via SMT Solving.

In Proc. FMOODS/FORTE'13.

Application example: Termination analysis for programs

APROVE

Automated Program Verification Environment



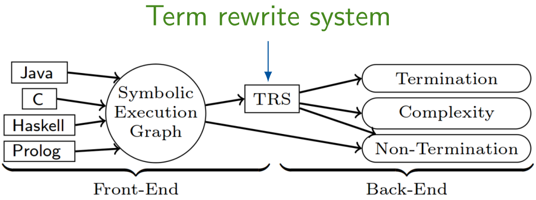
Source: T. Ströder, C. Aschermann, F. Frohn, J. Hensel, J. Giesl.

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

In Proc. TACAS'15.

Application example: Termination analysis for programs

Automated Program Verification Environment



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

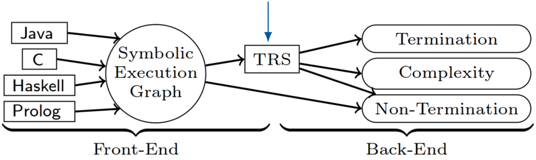
AProVE: Termination and memory safety of C programs (competition contribution).

In Proc. TACAS'15.

Application example: Termination analysis for programs

Automated Program Verification Environment

Term rewrite system



Term rewrite system

$$\mathsf{minus}(x,0) \to x \quad (1) \qquad \mathsf{div}(0,\mathsf{s}(y)) \to 0 \quad (4)$$
$$\mathsf{minus}(0,\mathsf{s}(y)) \to 0 \quad (2) \qquad \mathsf{div}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{s}(\mathsf{div}(\mathsf{minus}(x,y),\mathsf{s}(y))) \quad (5)$$
$$\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{minus}(x,y) \quad (3)$$

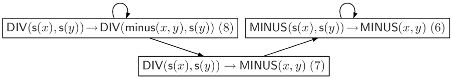
↓

Dependency pairs

$$\mathsf{MINUS}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{MINUS}(x,y) \quad (6) \qquad \mathsf{DIV}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{MINUS}(x,y) \quad (7)$$
$$\mathsf{DIV}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{DIV}(\mathsf{minus}(x,y),\mathsf{s}(y)) \quad (8)$$

↓

Chains



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.

Application example: jUnit$_{RV}$ for runtime verification of multi-threaded, object-oriented systems

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

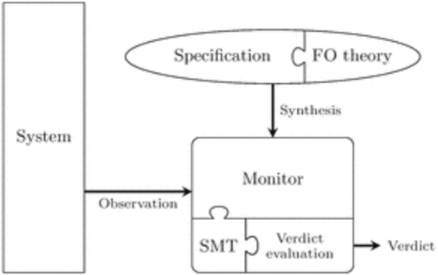


Fig. 1 Schematic overview of the monitoring approach

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

Monitoring modulo theories.

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

Application example: Planning

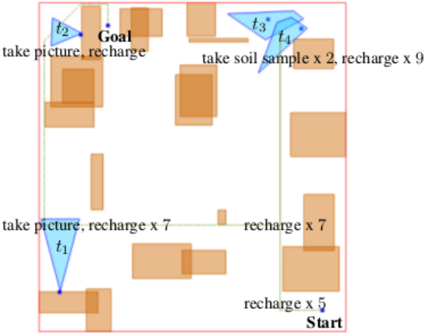


Figure 1: A GEOMETRIC ROVERS example instance, showing the starting and goal locations of the rover, areas where tasks can be performed (blue) and obstacles (orange) and a plan solving the task (green). The red box indicates the bounds of the environment.

Source: E. Scala, M. Ramirez, P. Haslum, S. Thiebaux.

Numeric planning with disjunctive global constraints via SMT.

In Proc. of ICASP'16.

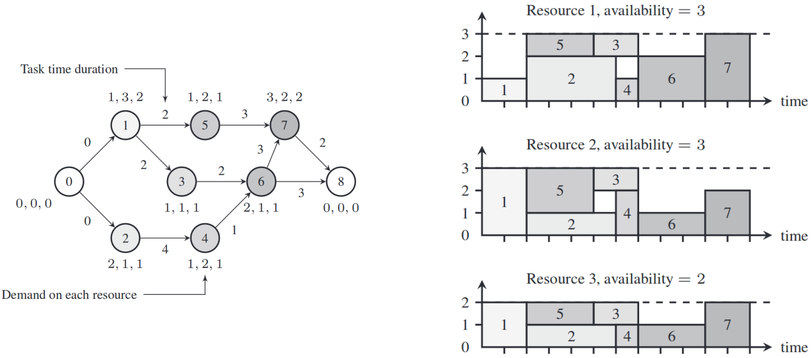


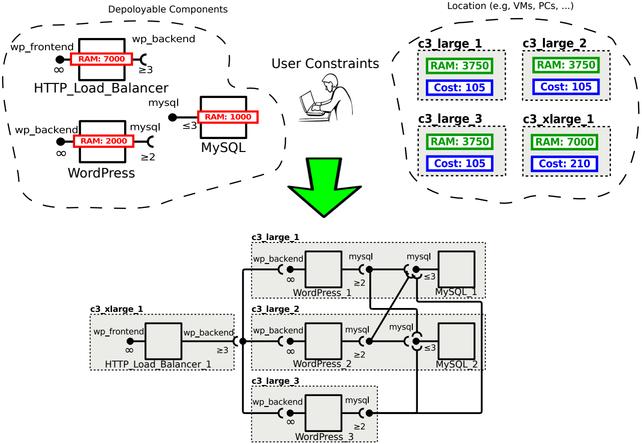
Figure 1: An example of RCPSP (Liess and Michelon 2008)

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.

Application example: Deployment optimisation on the cloud

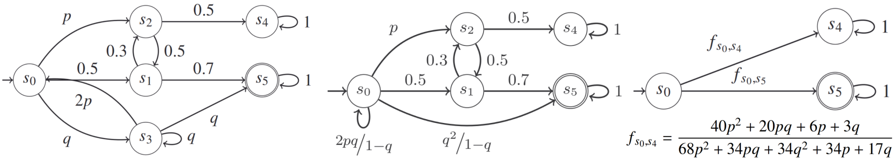


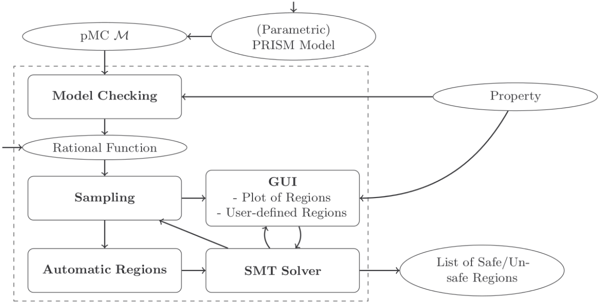
Source: E. Ábrahám, F. Corzilius, E. Broch Johnsen, G. Kremer, J. Mauro.

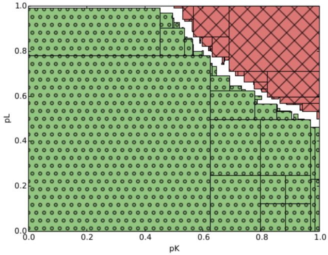
Zephyrus2: On the fly deployment optimization using SMT and CP technologies.

Submitted to SETTA'16.

Application example: Parameter synthesis for probabilistic systems







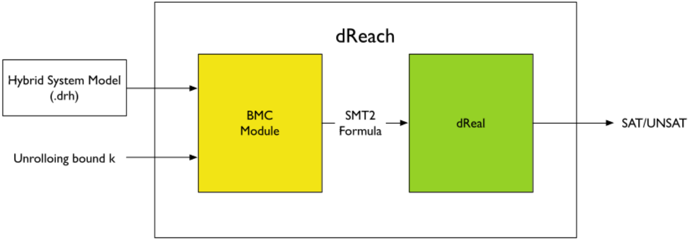
Source: C. Dehnert, S. Junges, N. Jansen, F. Corzilius, M. Volk, H. Bruintjes, J.-P. Katoen, E. Ábrahám.

Application example: Hybrid systems reachability analysis

DREAL DREACH BENCHMARKS PUBLICATION DOWNLOAD TRY ONLINE PEOPLE

dReach is a tool for safety verfication 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 hyrbid 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/`