Formal Methods in Software Development

Overview

Mădălina Erașcu

West University of Timișoara
Faculty of Mathematics and Informatics
Department of Computer Science

Based on slides of the lecture Satisfiability Checking (Erika Ábrahám), RTWH Aachen

WS 2019/2020

Organization

See `https://merascu.github.io`

What is this lecture about?

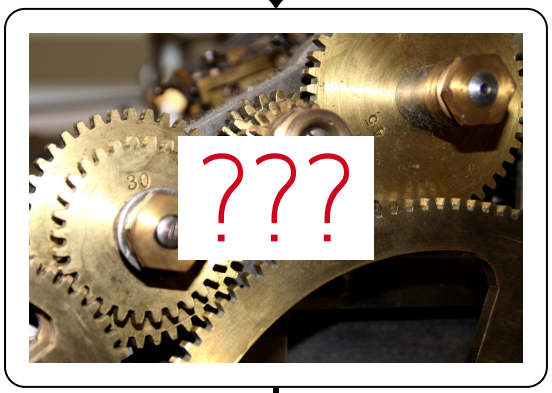
Quantifier-free logical formula

Solver

Satisfiability of the input formula

What is this lecture about?

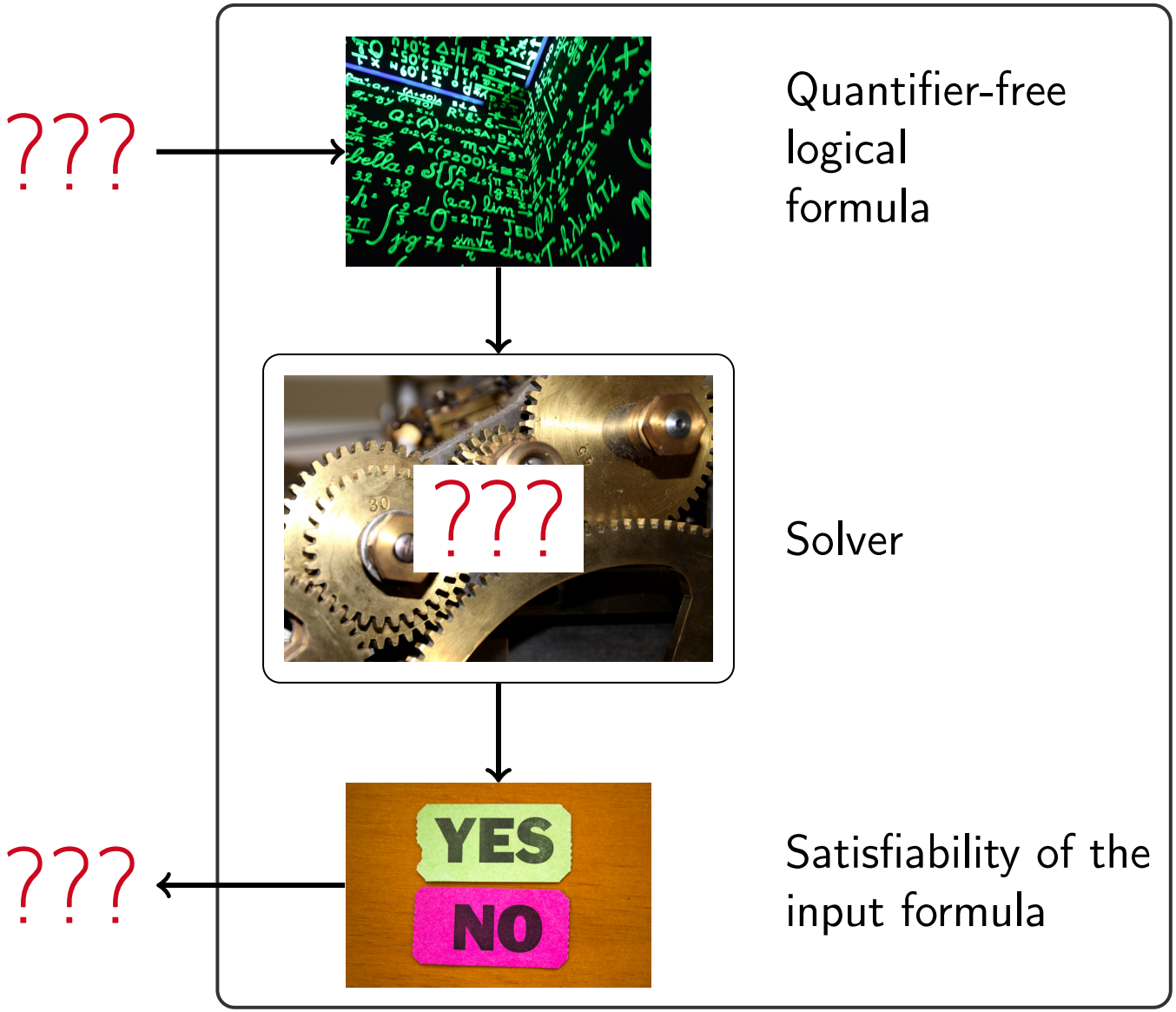
Quantifier-free logical formula



Solver

Satisfiability of the input formula

What is this lecture about?



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: $a = true$, $b = false$, $c = true$

It is the perhaps most well-known NP-complete problem [Cook, 1971] [Levin, 1973].

...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: $\exists_x \left(x^2 + 1 \geq 0 \wedge x < 0\right)$

Satisfying assignment: $x = -1$

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

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
- ...

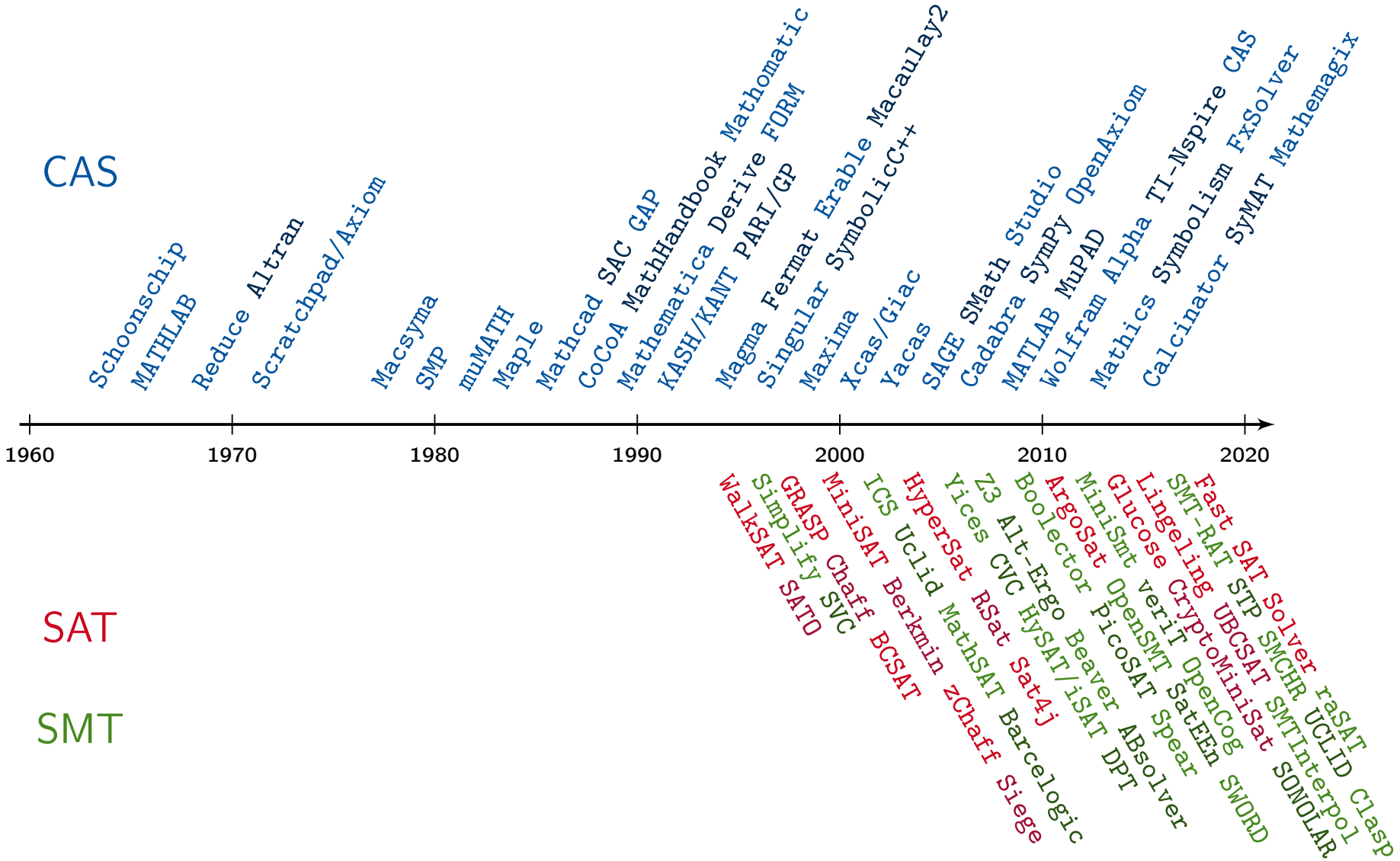
Satisfiability checking: Some milestones

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

1940 Computer architecture development

	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	FGLM algorithm Partial CAD Comprehensive Gröbner bases	Conflict-directed backjumping	
2000	Virtual substitution	CDCL 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.

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.

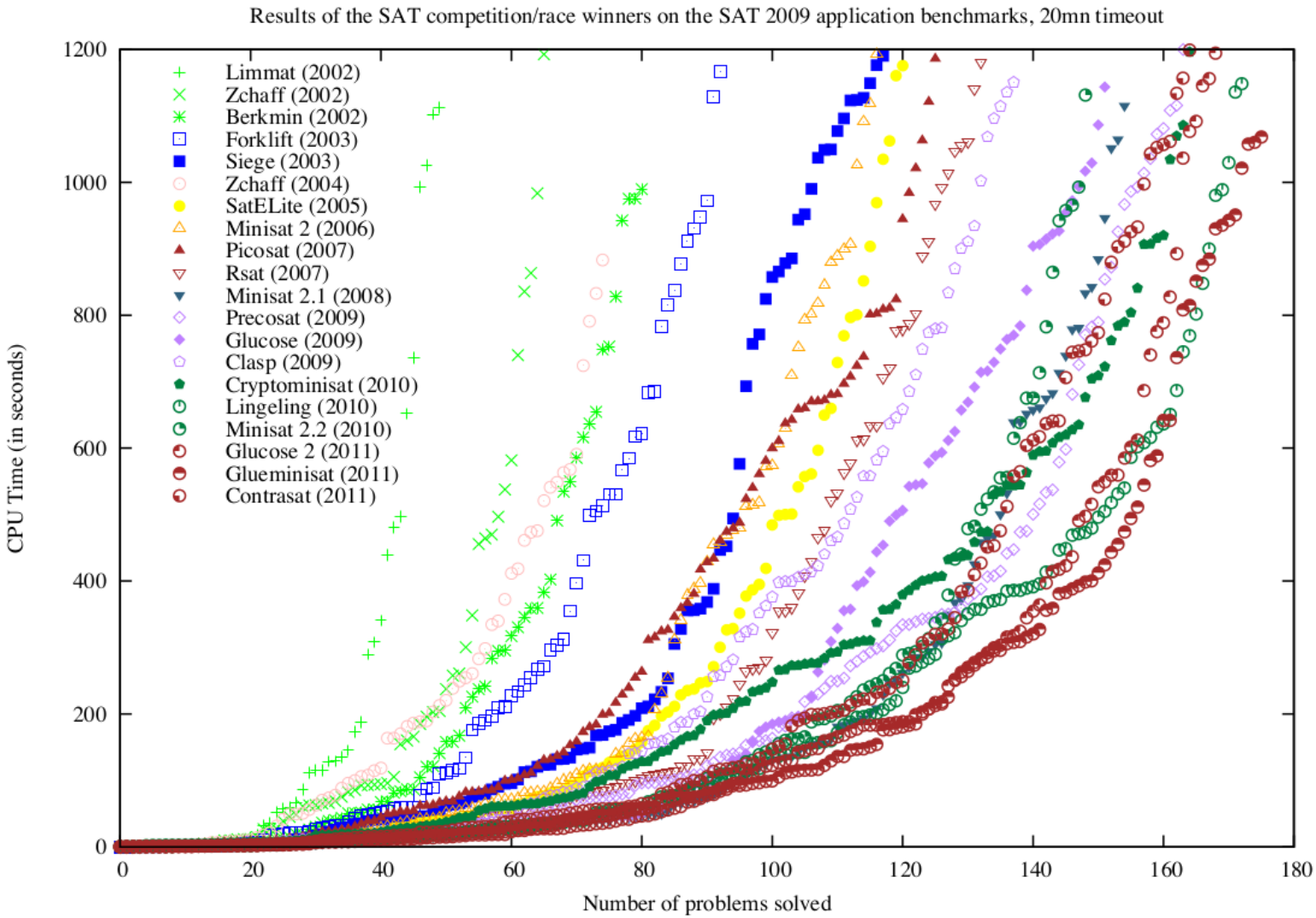
Community support:

- 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.

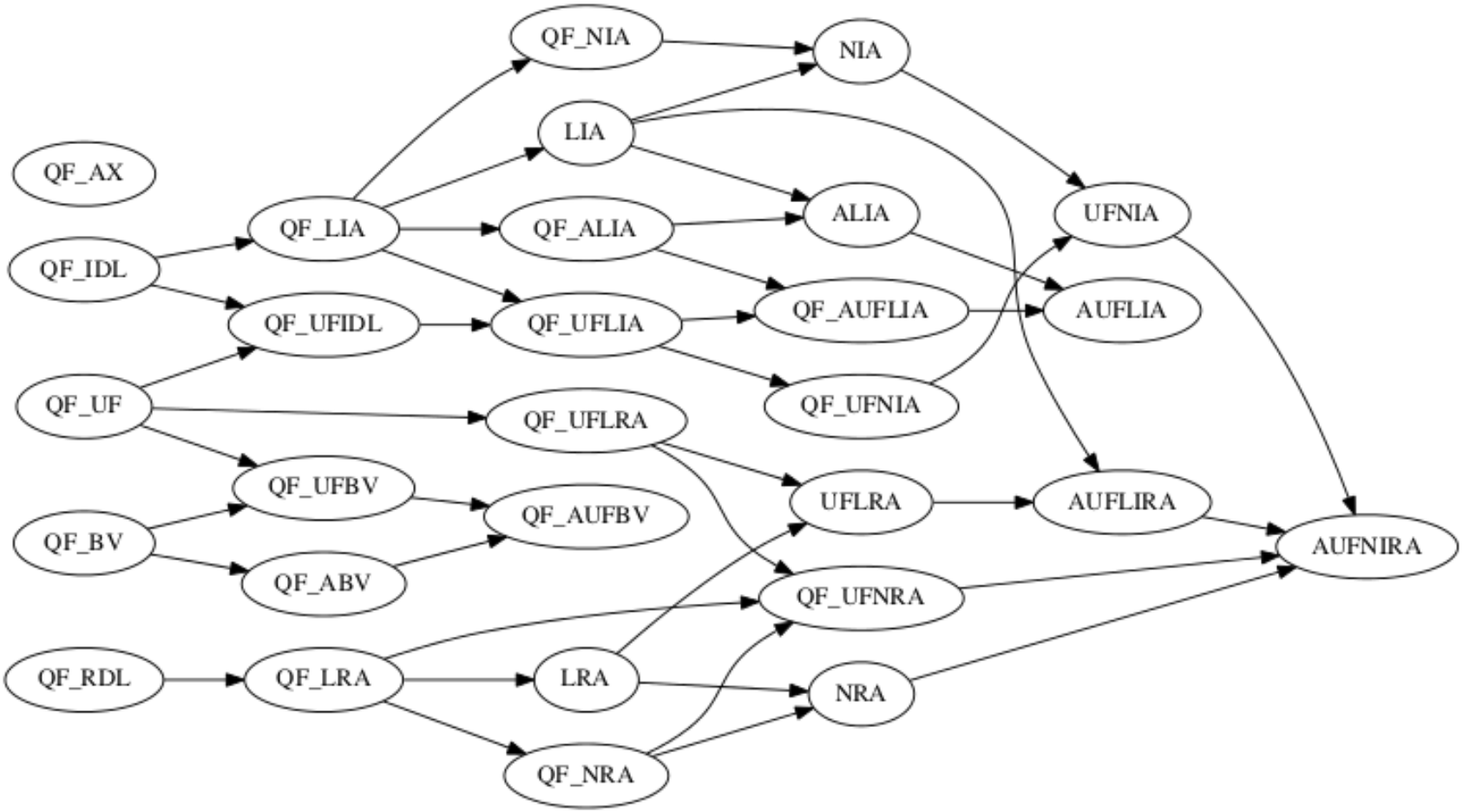
Satisfiability modulo theories solving

- 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

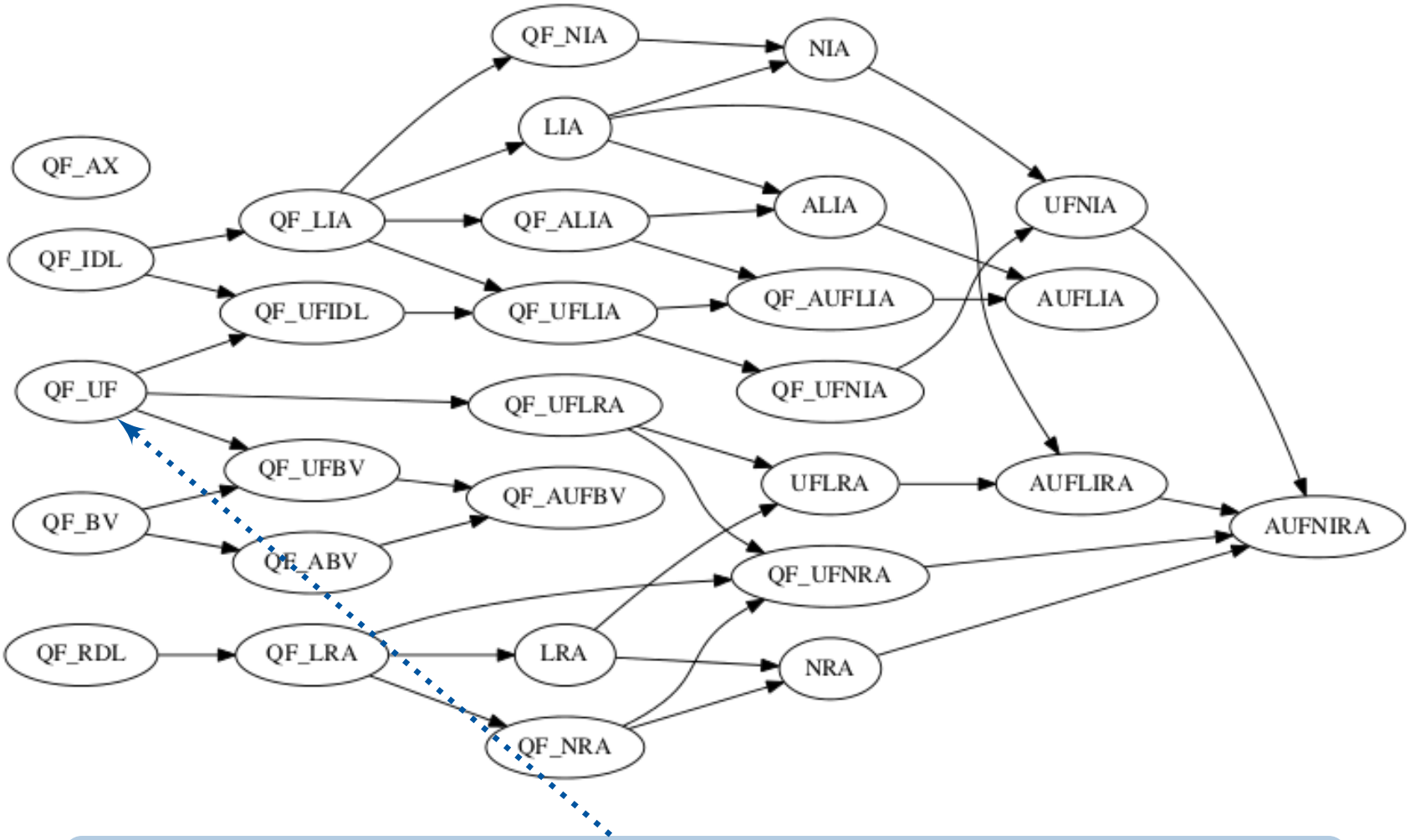
- Propositional logic is sometimes too weak for modelling.
- We need more expressive logics and decision procedures for them.
- Logics:
  quantifier-free fragments of first-order logic over various theories.
- Our focus: SAT-modulo-theories (SMT) solving.
- SMT-LIB as standard input language since 2004.
- 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

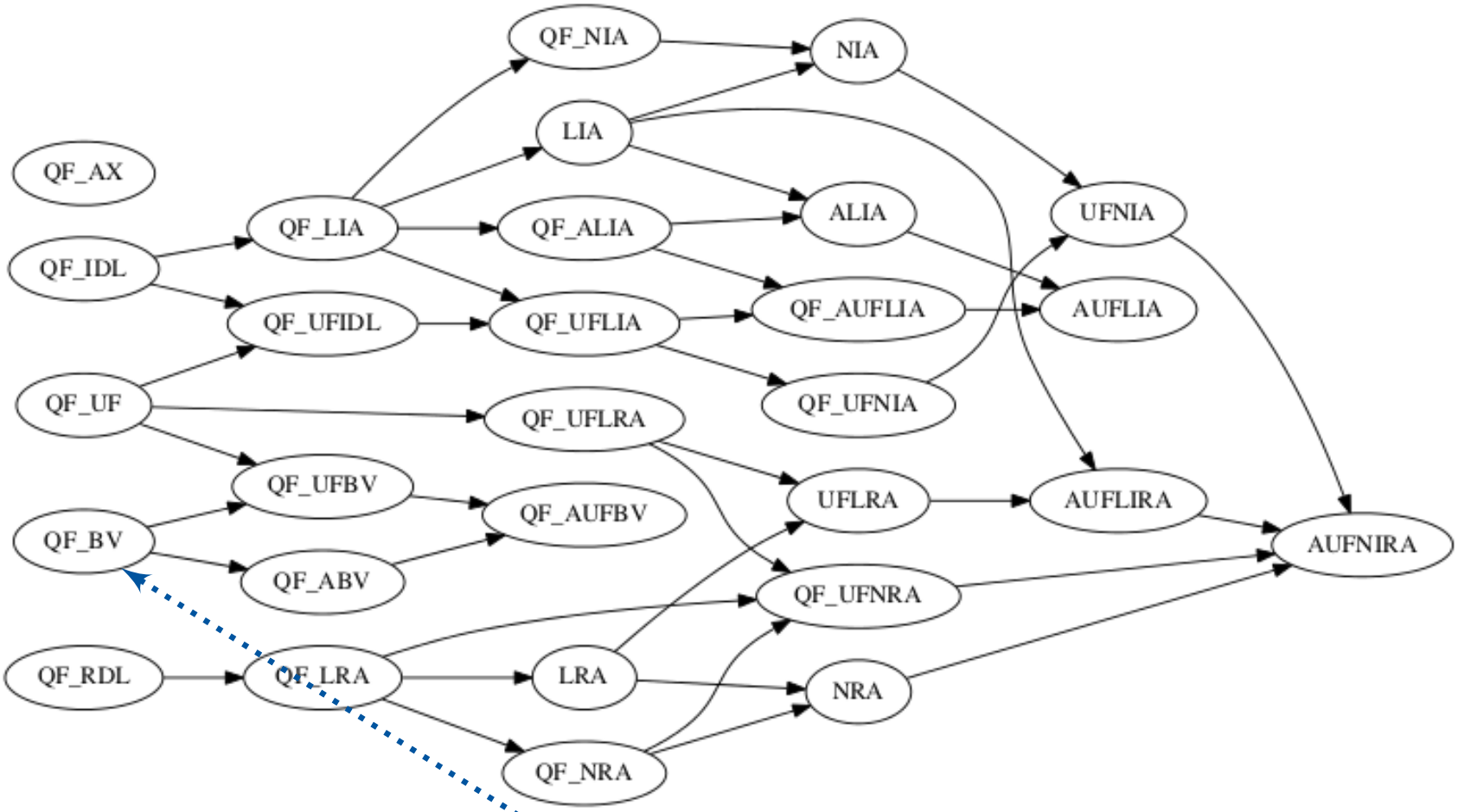


Quantifier-free equality logic with uninterpreted functions

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

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

SMT-LIB theories

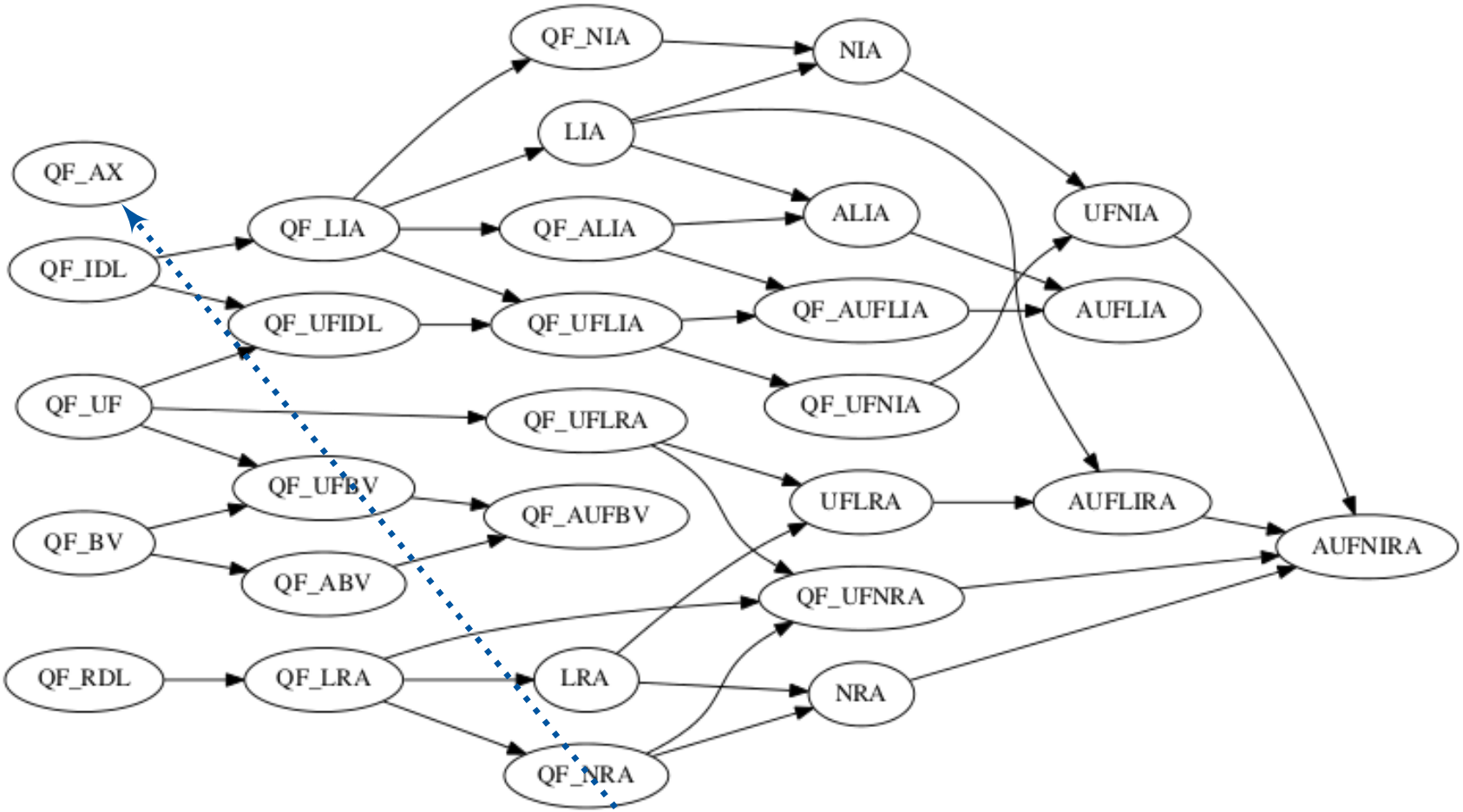


Quantifier-free bit-vector arithmetic

$$a + b \geq 0 \wedge (a|b) \leq (a\&b)$$

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

SMT-LIB theories

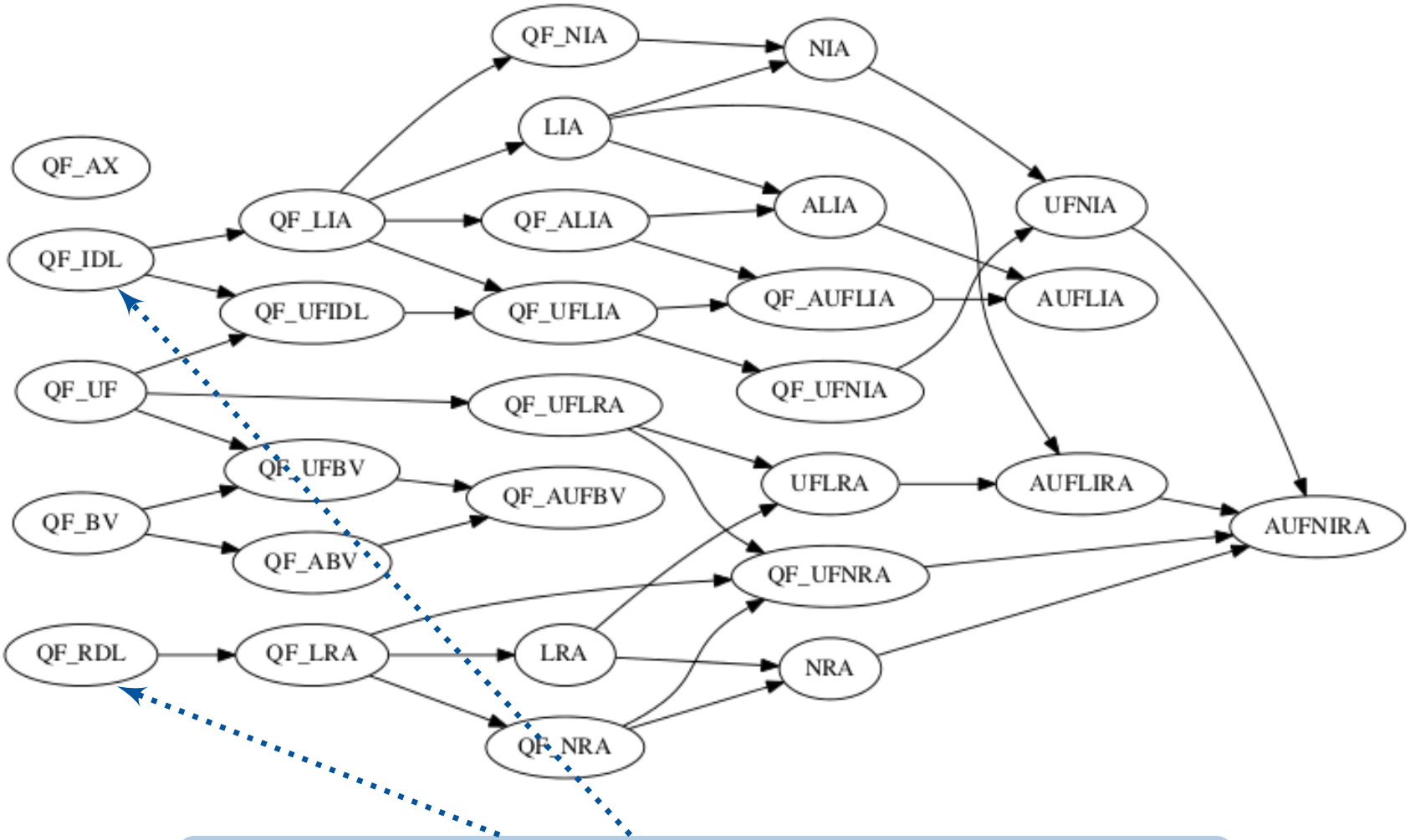


Quantifier-free array theory

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

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

SMT-LIB theories

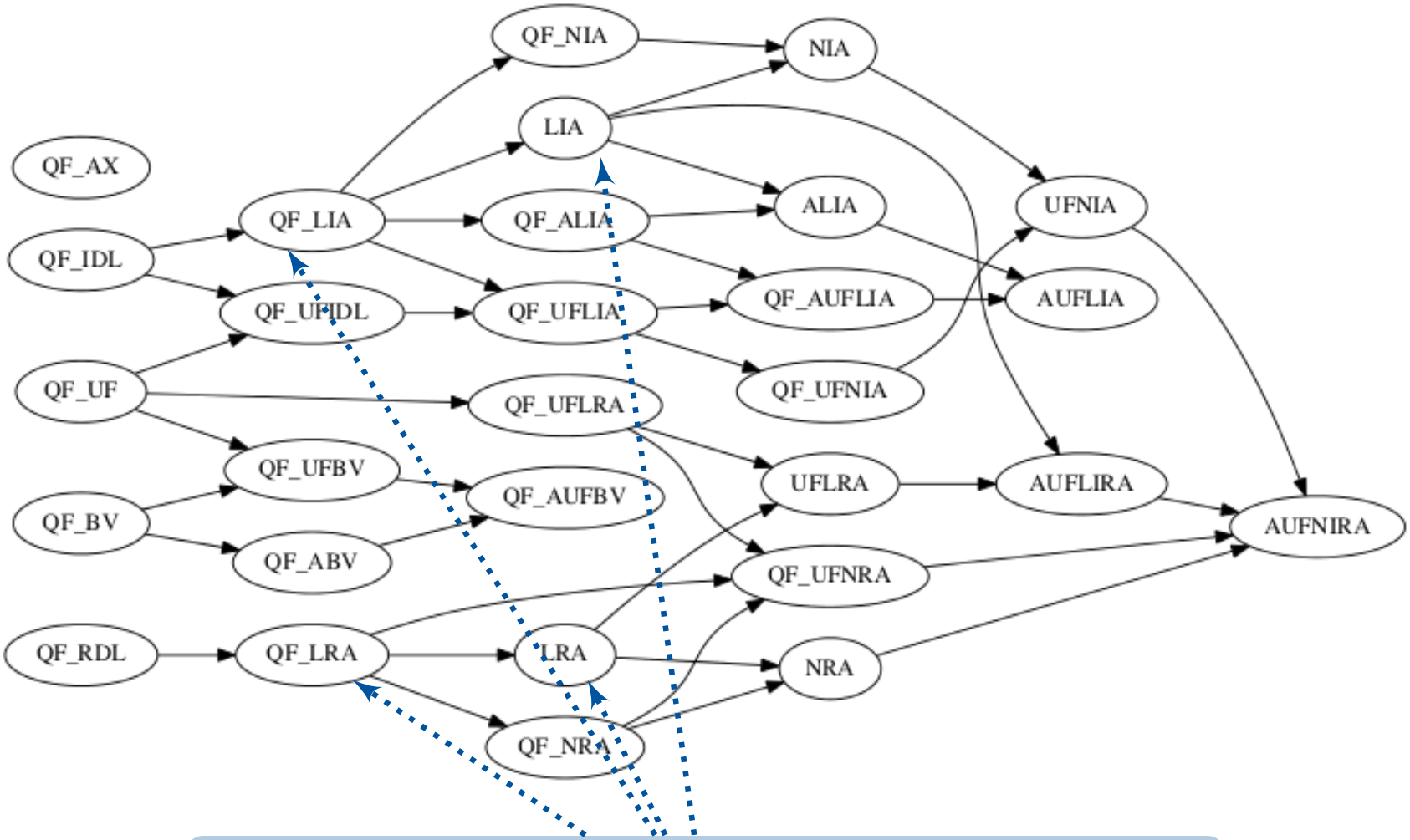


Quantifier-free integer/rational difference logic

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

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

SMT-LIB theories

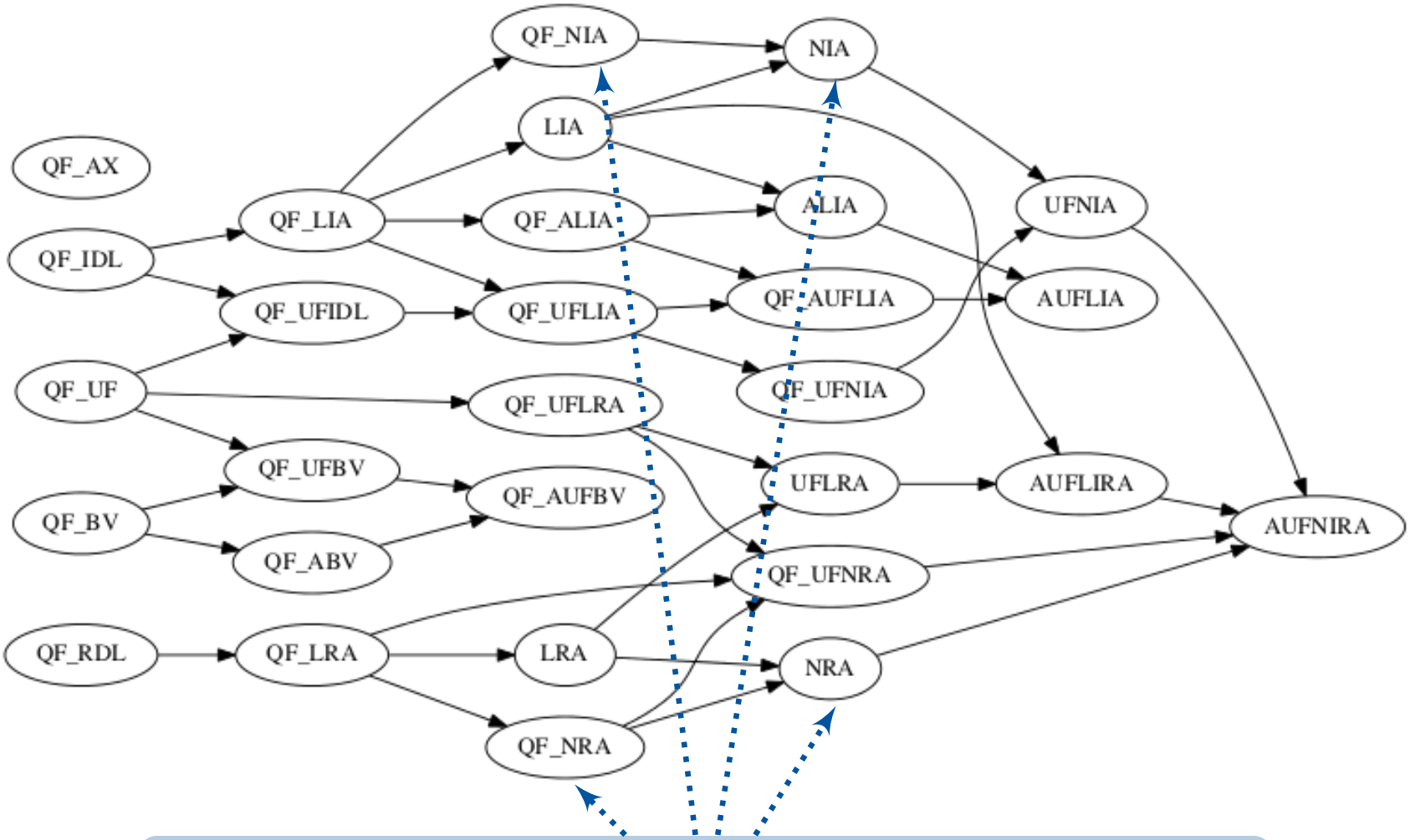


(Quantifier-free) real/integer linear arithmetic

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

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

SMT-LIB theories

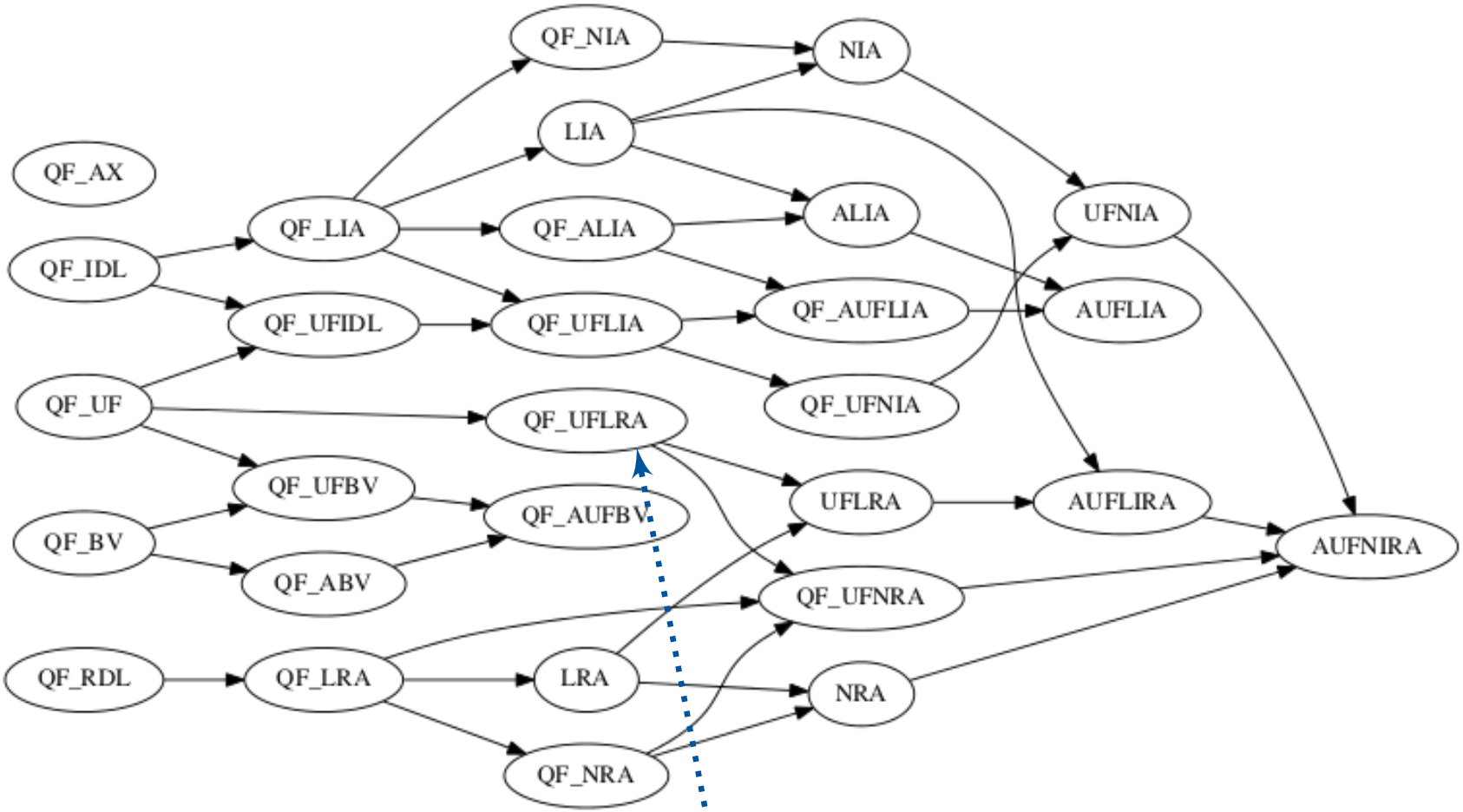


(Quantifier-free) real/integer non-linear arithmetic

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

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

SMT-LIB theories

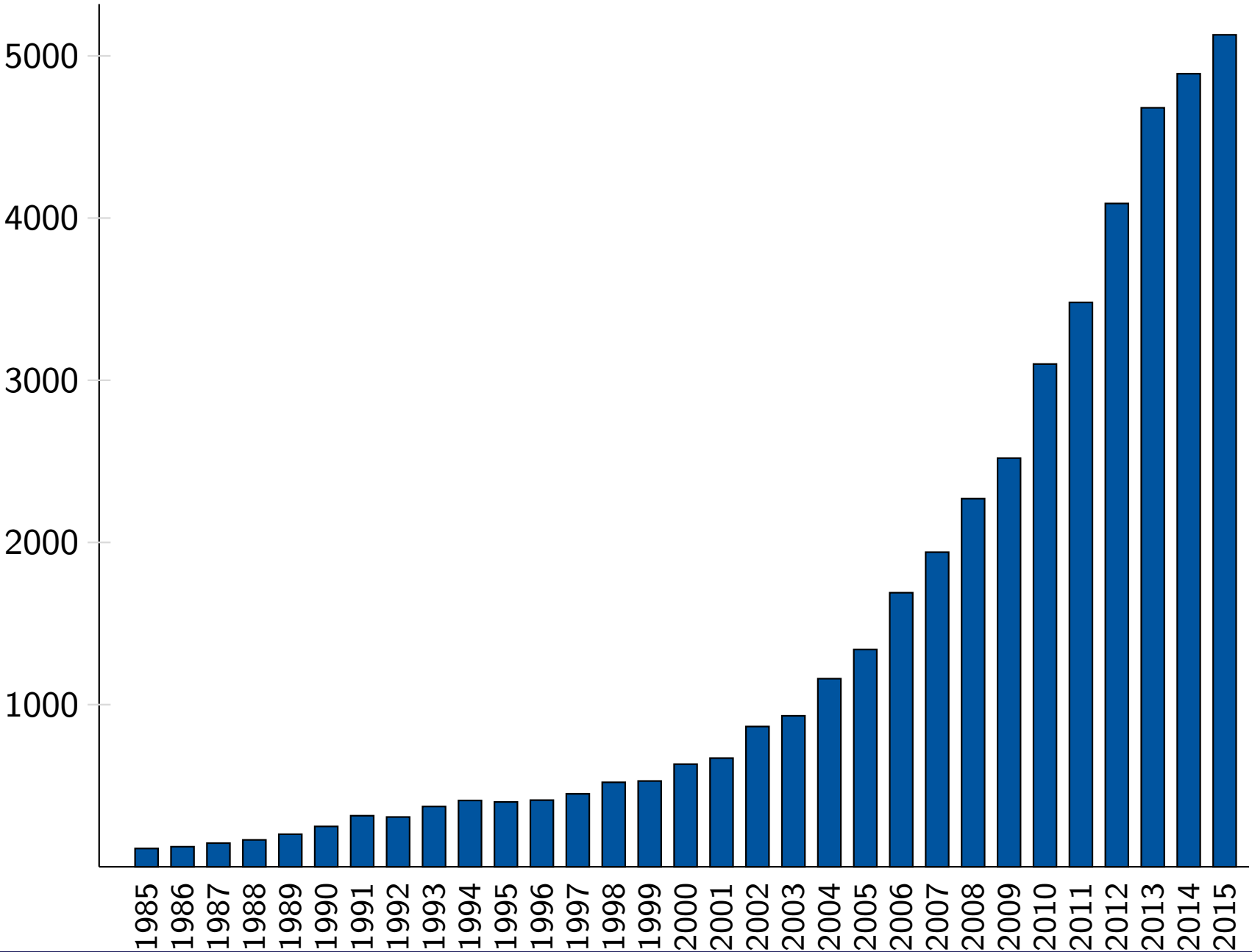


Combined theories

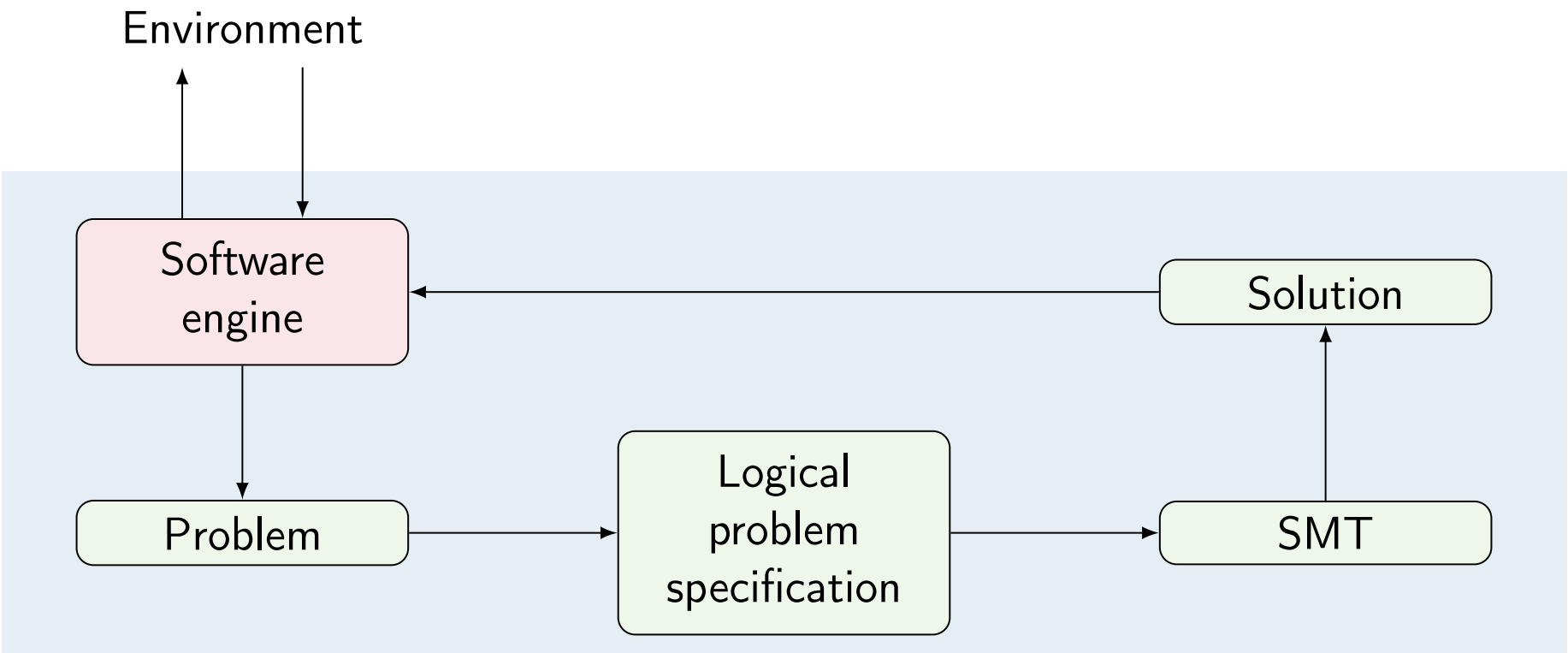
$$2f(x) + 5y > 0 \wedge \neg(f(x) = y \vee x + 2y = 0)$$

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

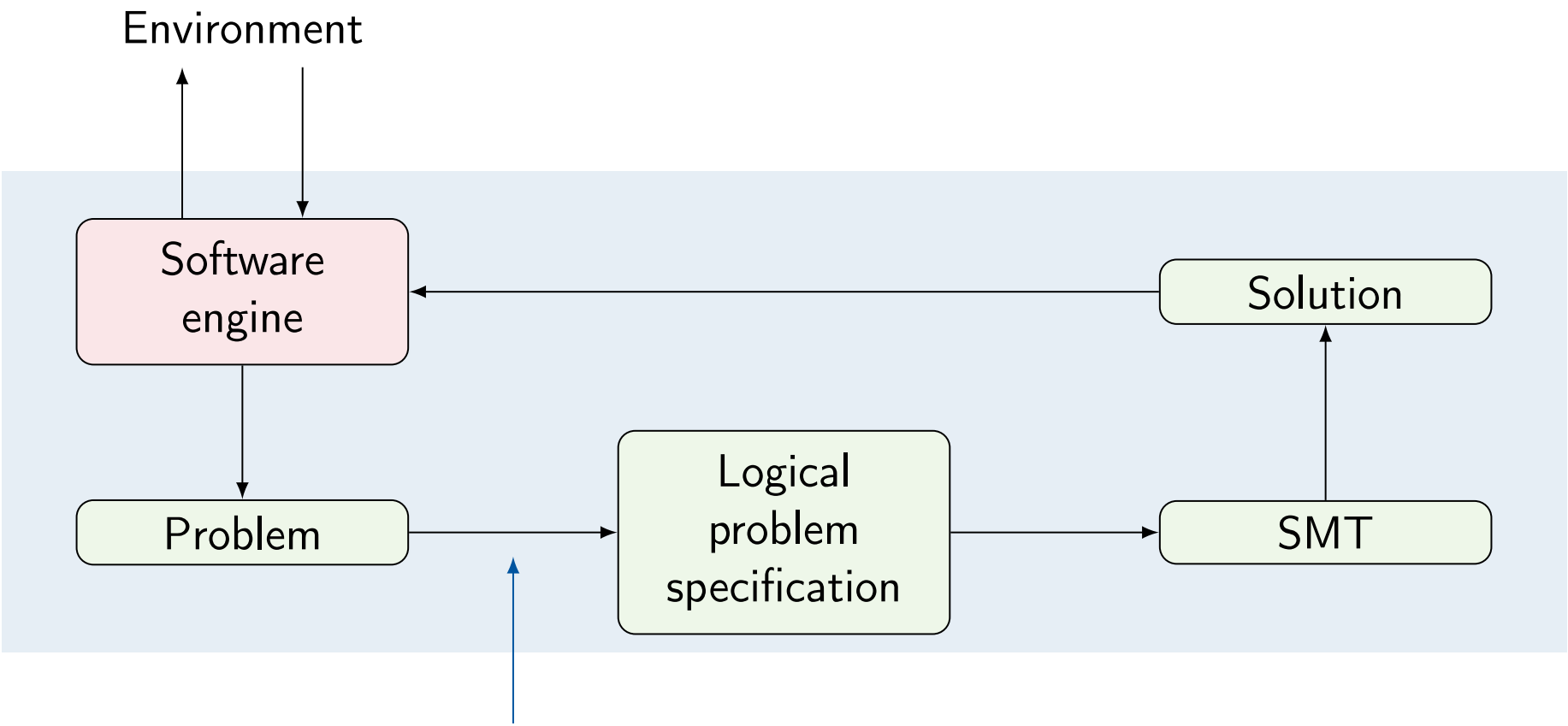
Google Scholar search for "SAT modulo theories"



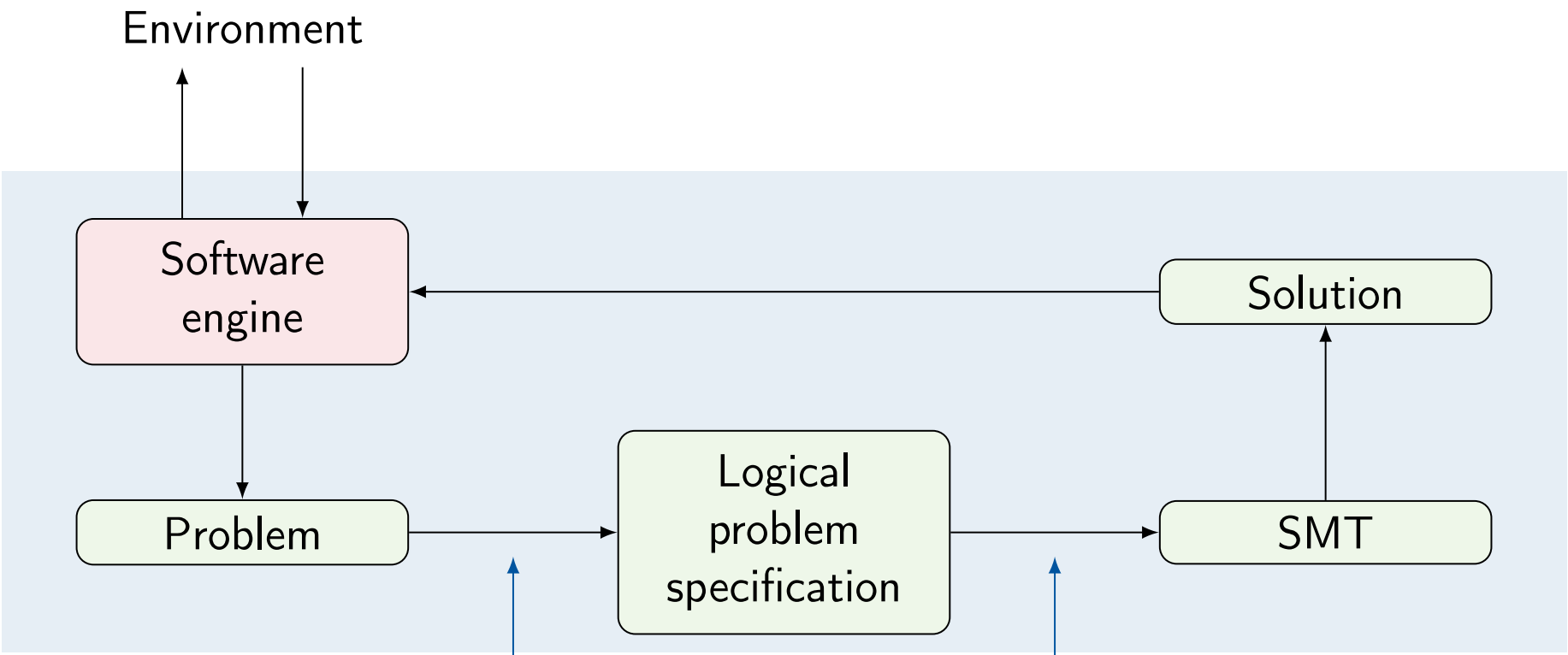
SAT/SMT embedding structure



SAT/SMT embedding structure



SAT/SMT embedding structure



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

standard input language → 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

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/`

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

Carnegie Mellon

CBMC Homepage

Bounded Model Checking for Software

Logical encoding of finite unsafe paths

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

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.

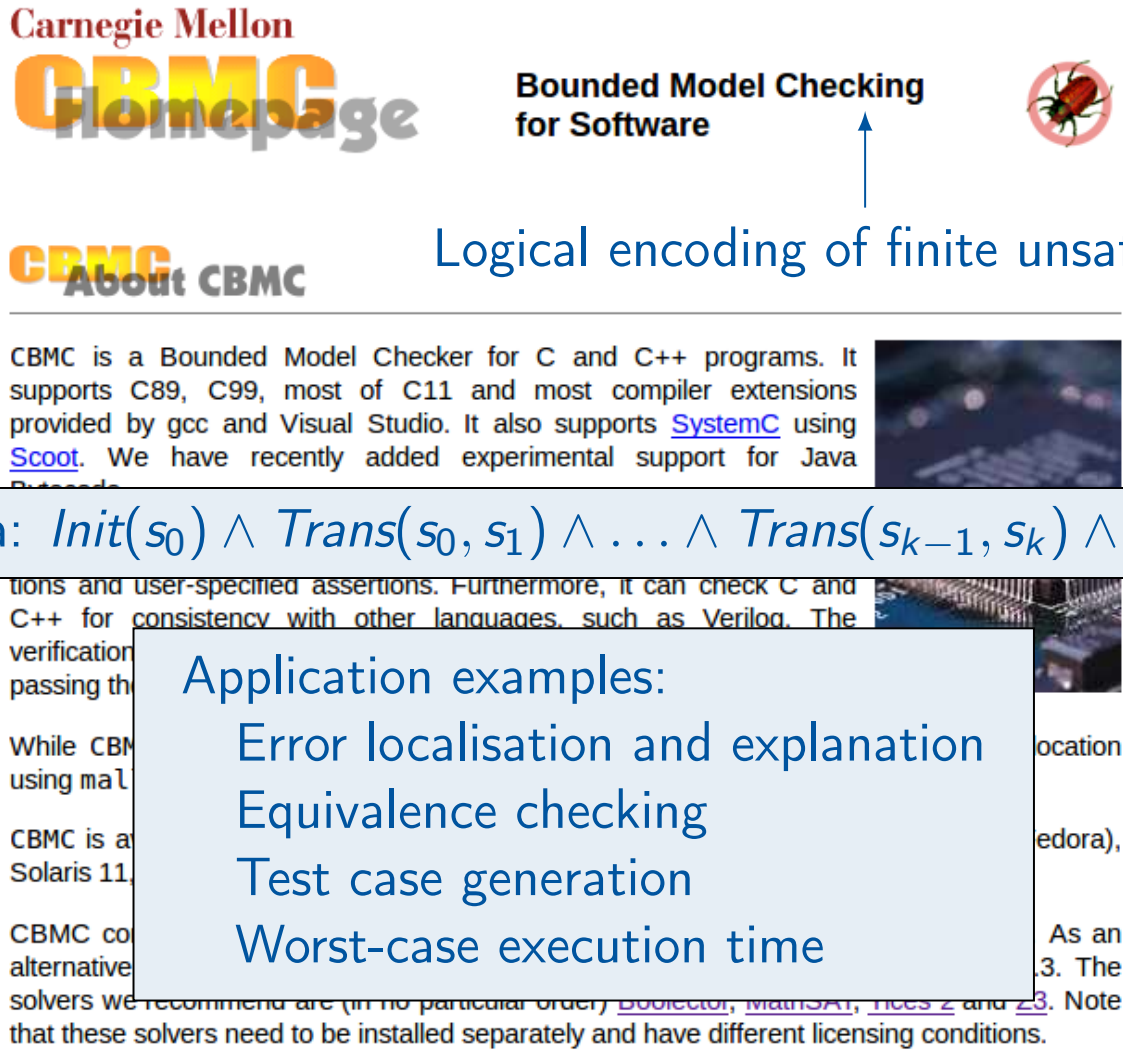
While CBMC is aimed for embedded software, it also supports dynamic memory allocation using `malloc` and `new`. For questions about CBMC, contact Daniel Kroening.

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.

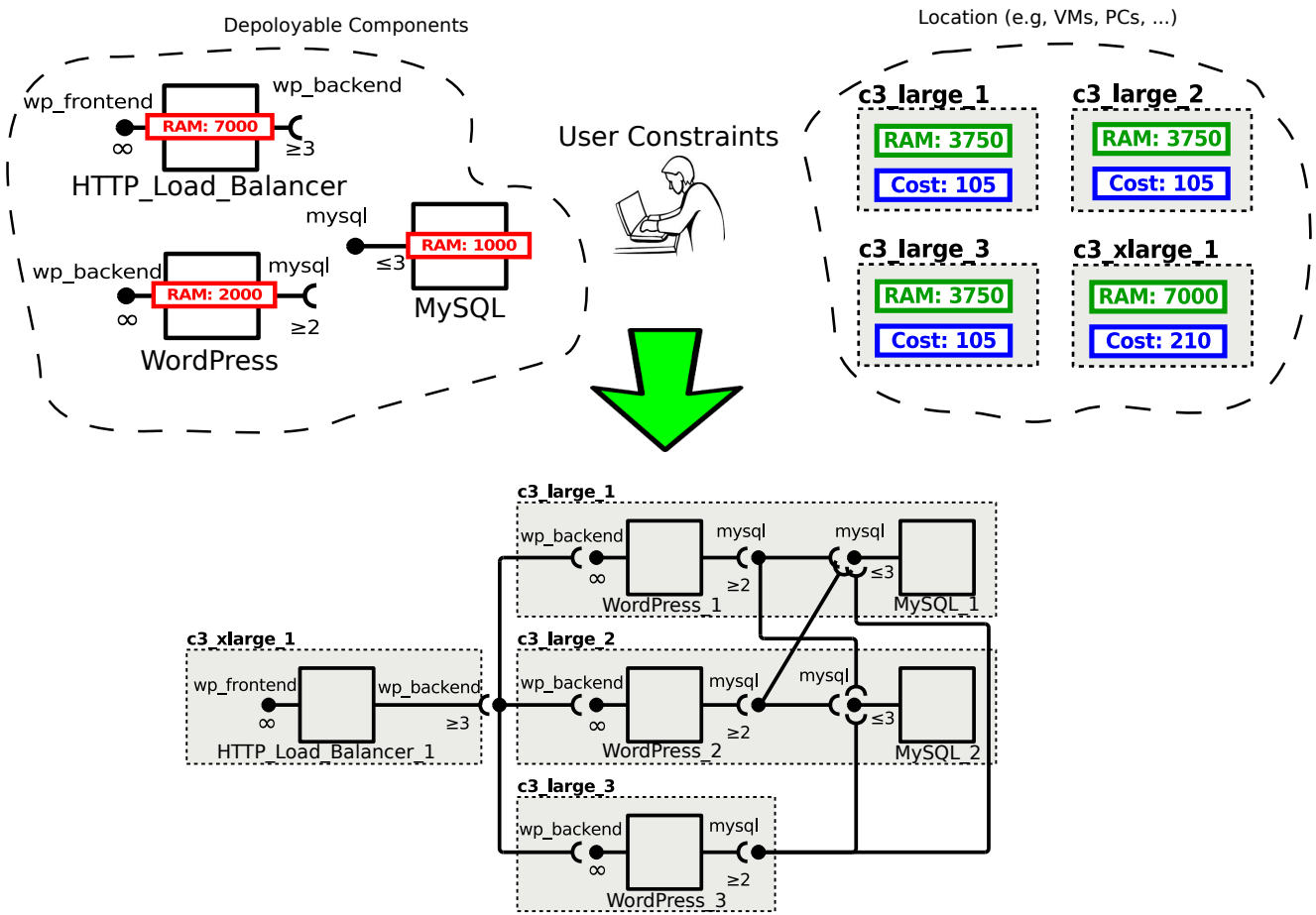
Source: D. Kroening. **CBMC home page.** `http://www.cprover.org/cbmc/`

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



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

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.