The Power of Satisfiability Checking

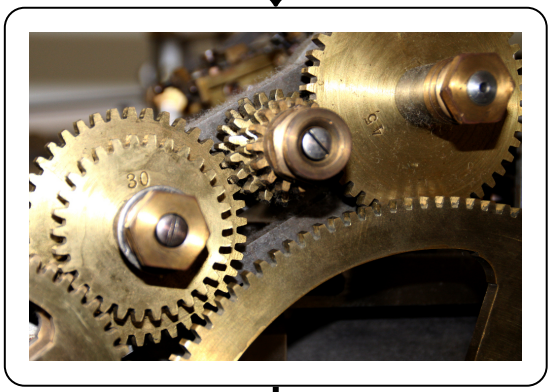
Erika Ábrahám

RWTH Aachen University, Germany

ECSS 2016
Informatics Driving the Digital World
Budapest, Hungary, October 24-26, 2016

What is this talk about?

Quantifier-free logical formula

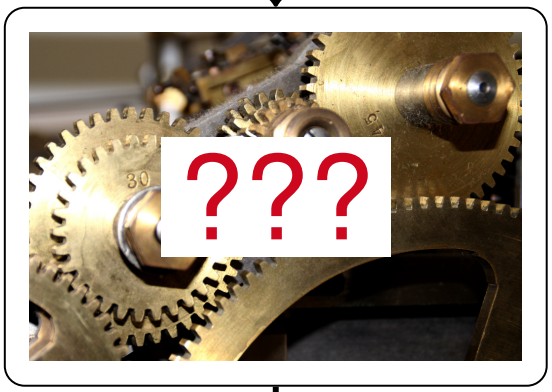


Solver

Satisfiability of the input formula

What is this talk about?

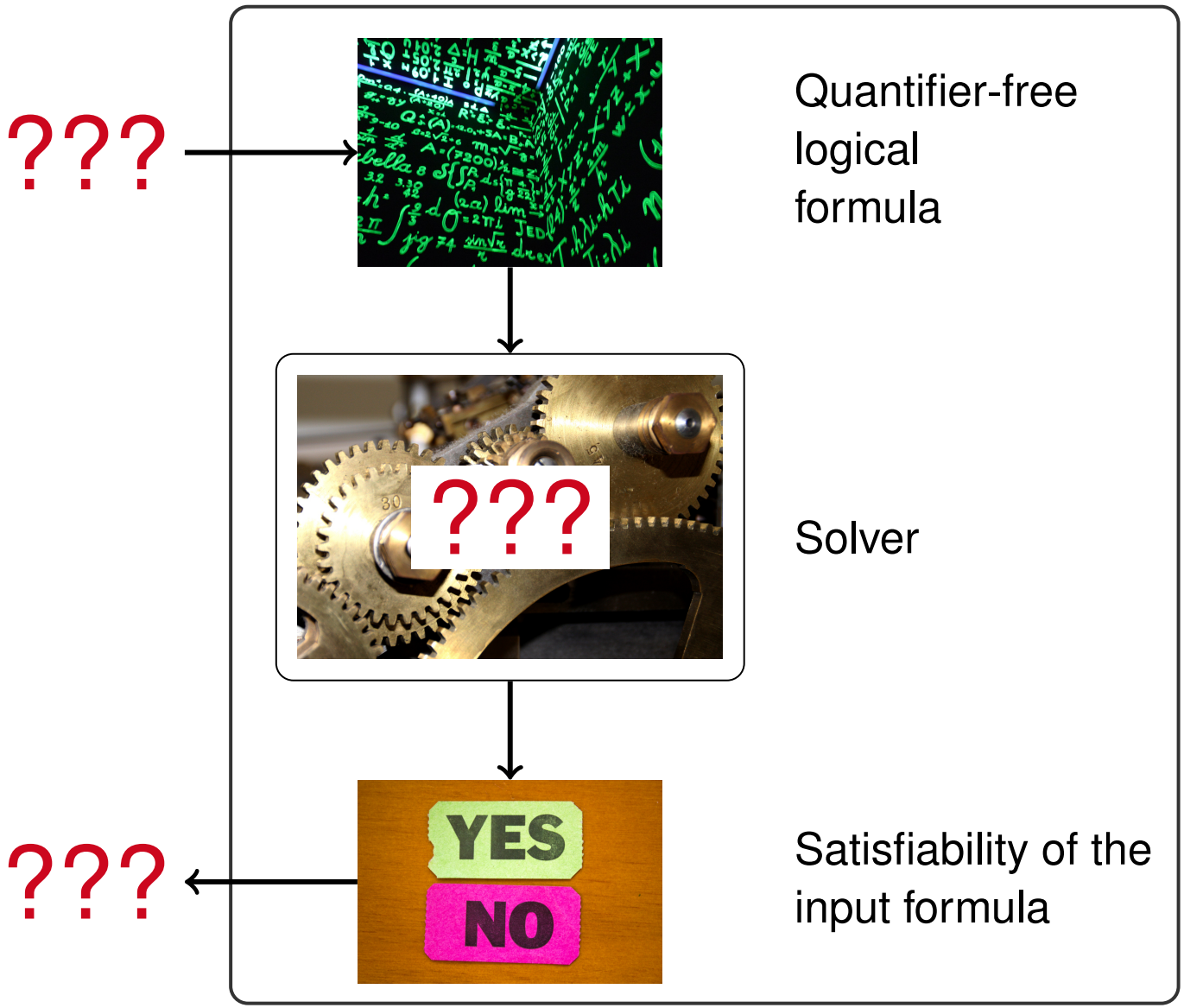
Quantifier-free logical formula



Solver

Satisfiability of the input formula

What is this talk about?



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:

$$a = \mathit{true}, \quad b = \mathit{false}, \quad c = \mathit{true}$$

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:

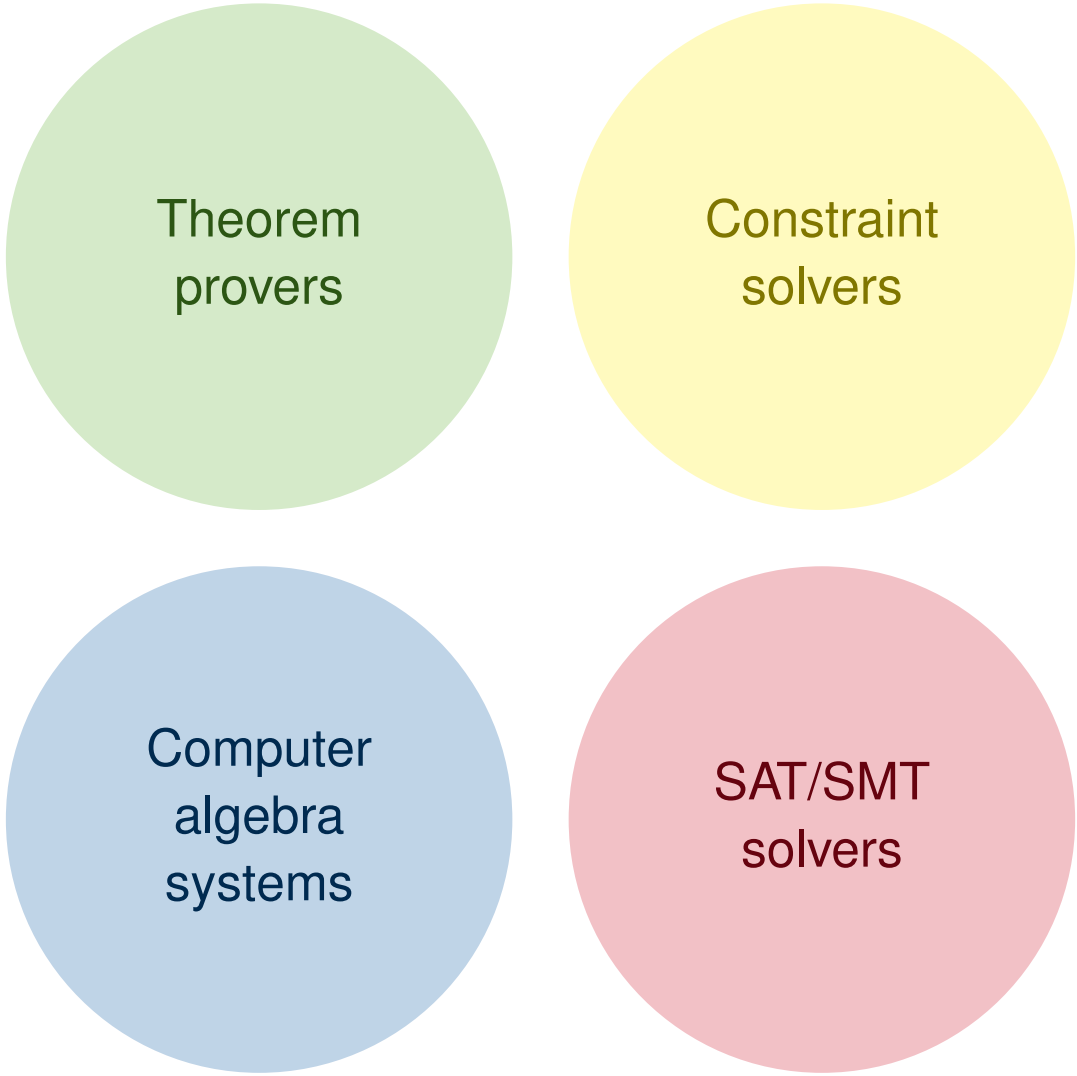
$$(x - 2y > 0 \vee x^2 - 2 = 0) \wedge x^4 y + 2x^2 - 4 > 0$$

Satisfying assignment:

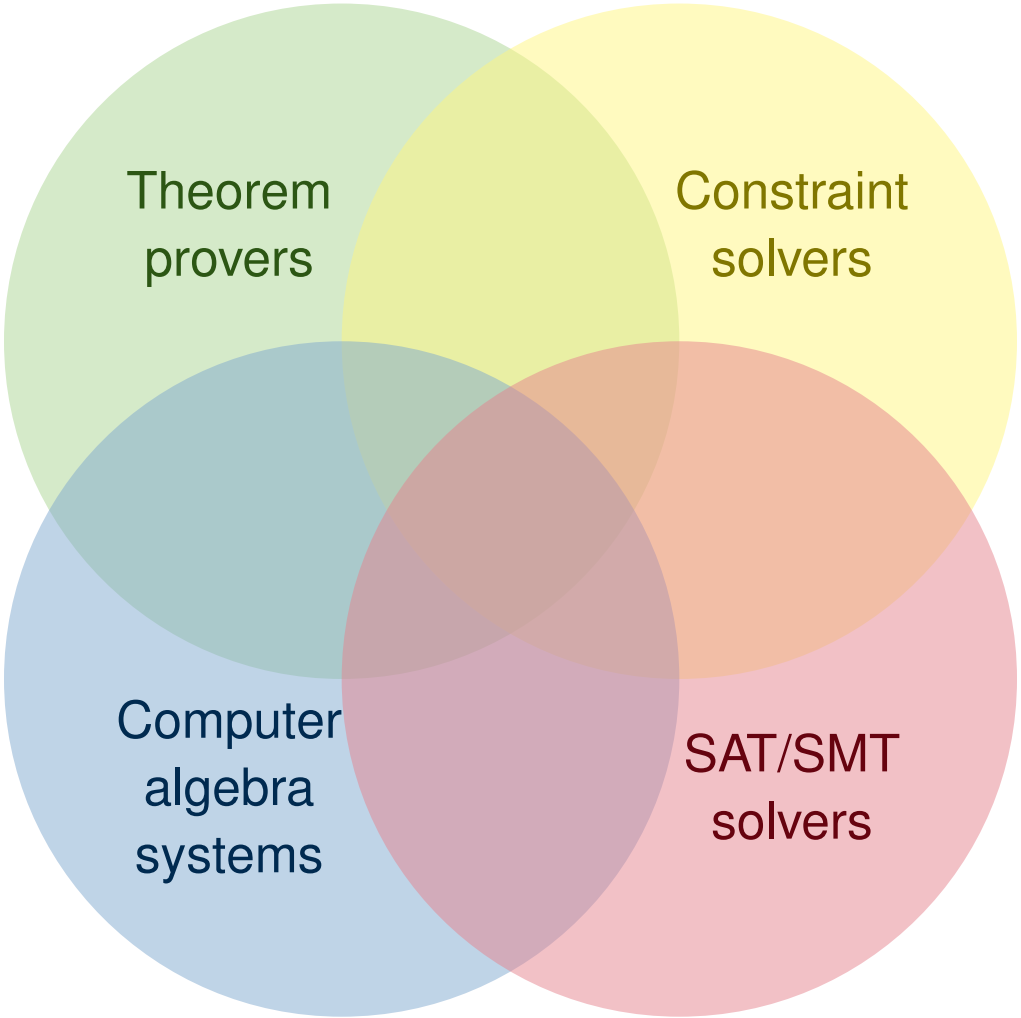
$$x = \sqrt{2}, \quad y = 2$$

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

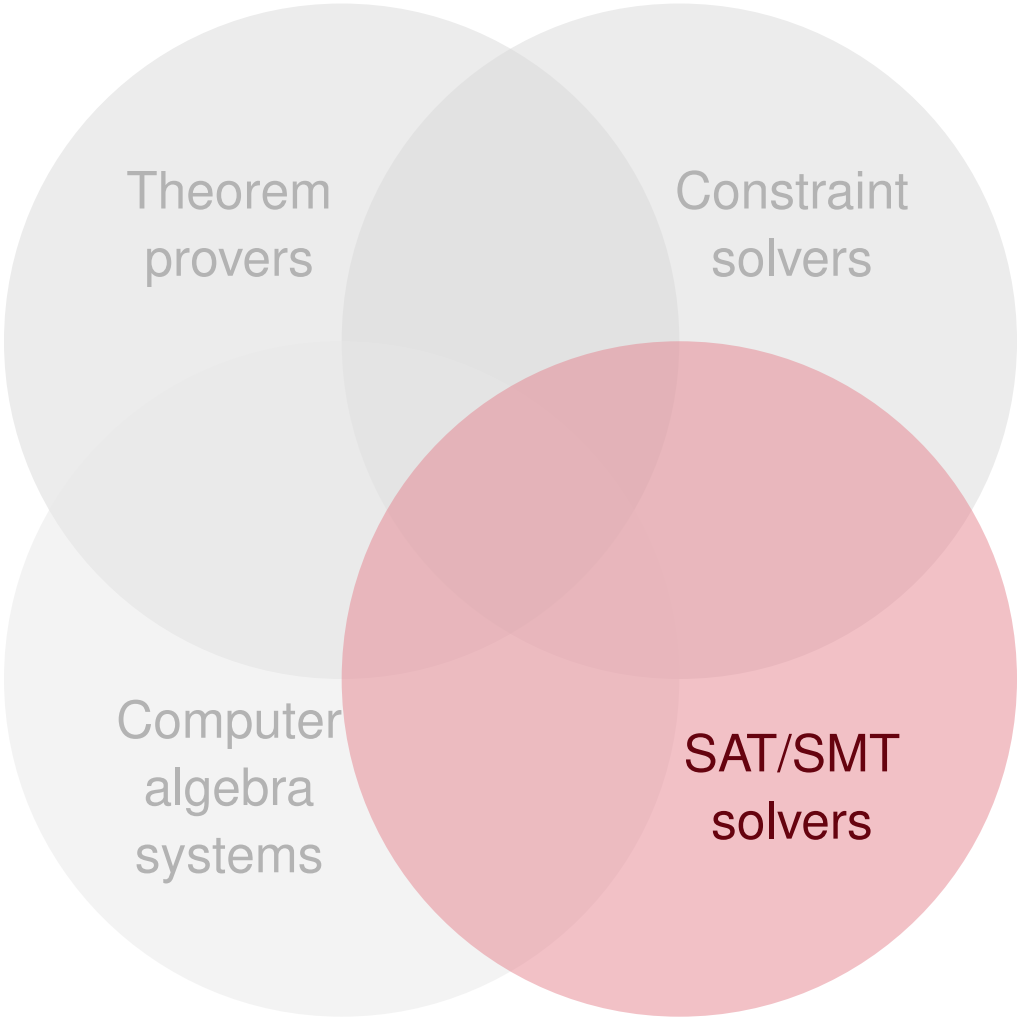
Some technologies for checking the satisfiability of logical formulas



Some technologies for checking the satisfiability of logical formulas



Some technologies for checking the satisfiability of logical formulas



Part I:
Some historical notes

Part II:
One of the major ingredients of success:
Strategic combinations of decision procedures
...in SAT solving...
...in SMT solving...
...in theory solving.

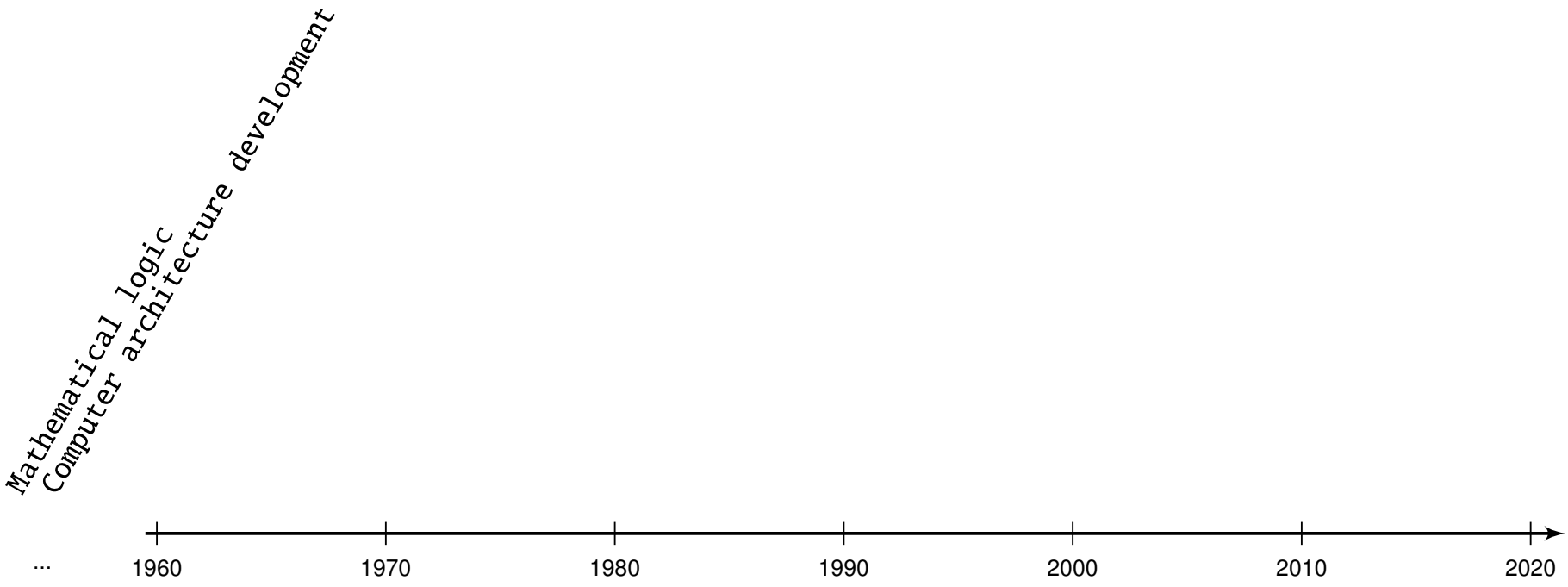
Part III:
How satisfiability checking drives the digital world

Part I:
Some historical notes

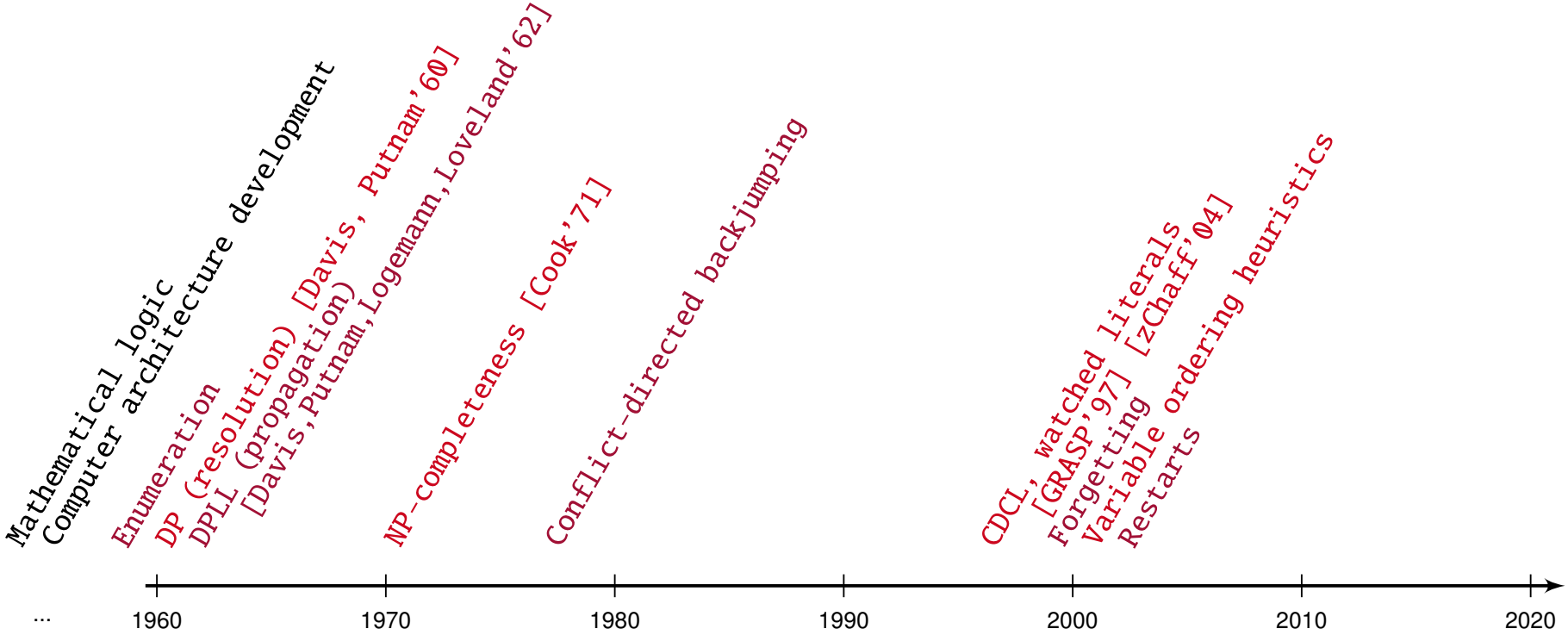
Part II:
One of the major ingredients of success:
Strategic combinations of decision procedures
...in SAT solving...
...in SMT solving...
...in theory solving.

Part III:
How satisfiability checking drives the digital world

Somemilestones in SAT/SMT development (incomplete!)

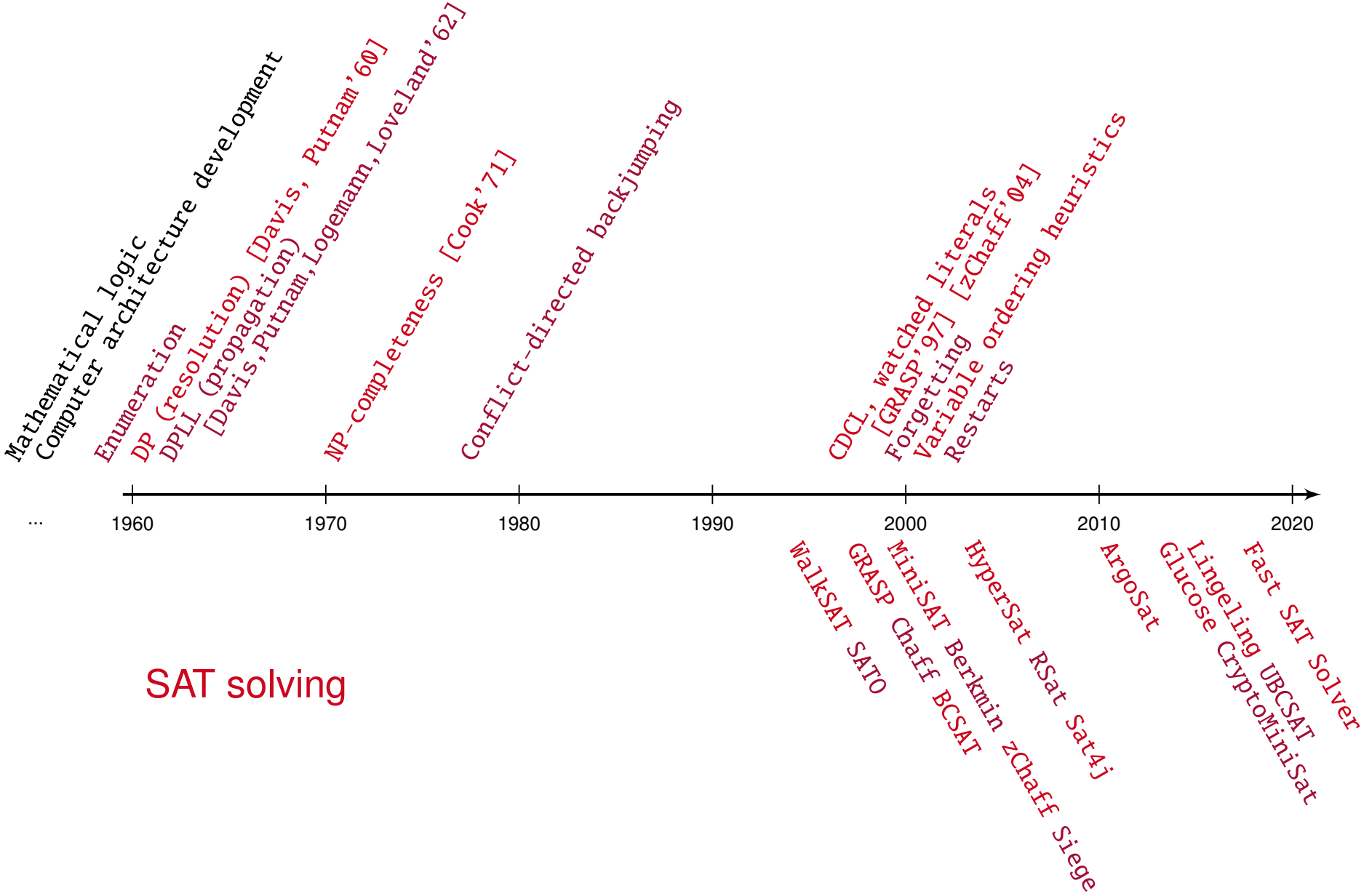


Somemilestones in SAT/SMT development (incomplete!)



SAT solving

Somemilestones in SAT/SMT development (incomplete!)



SAT solving

Satisfiability checking for propositional logic

SAT-solving community support:

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

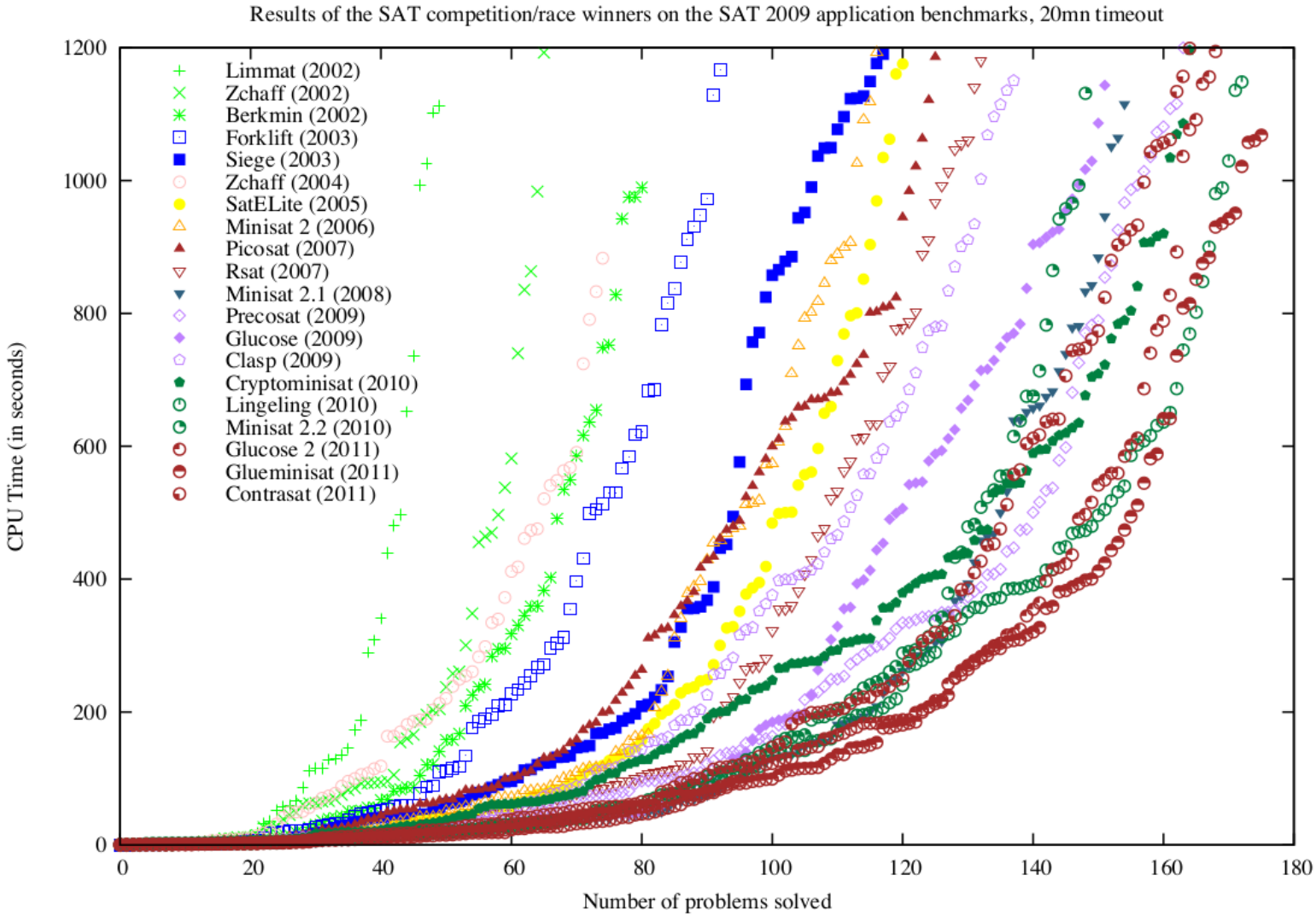
  2016 SAT Competition: 6 tracks, 29 solvers in the main track.

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

An impression of the SAT solver development

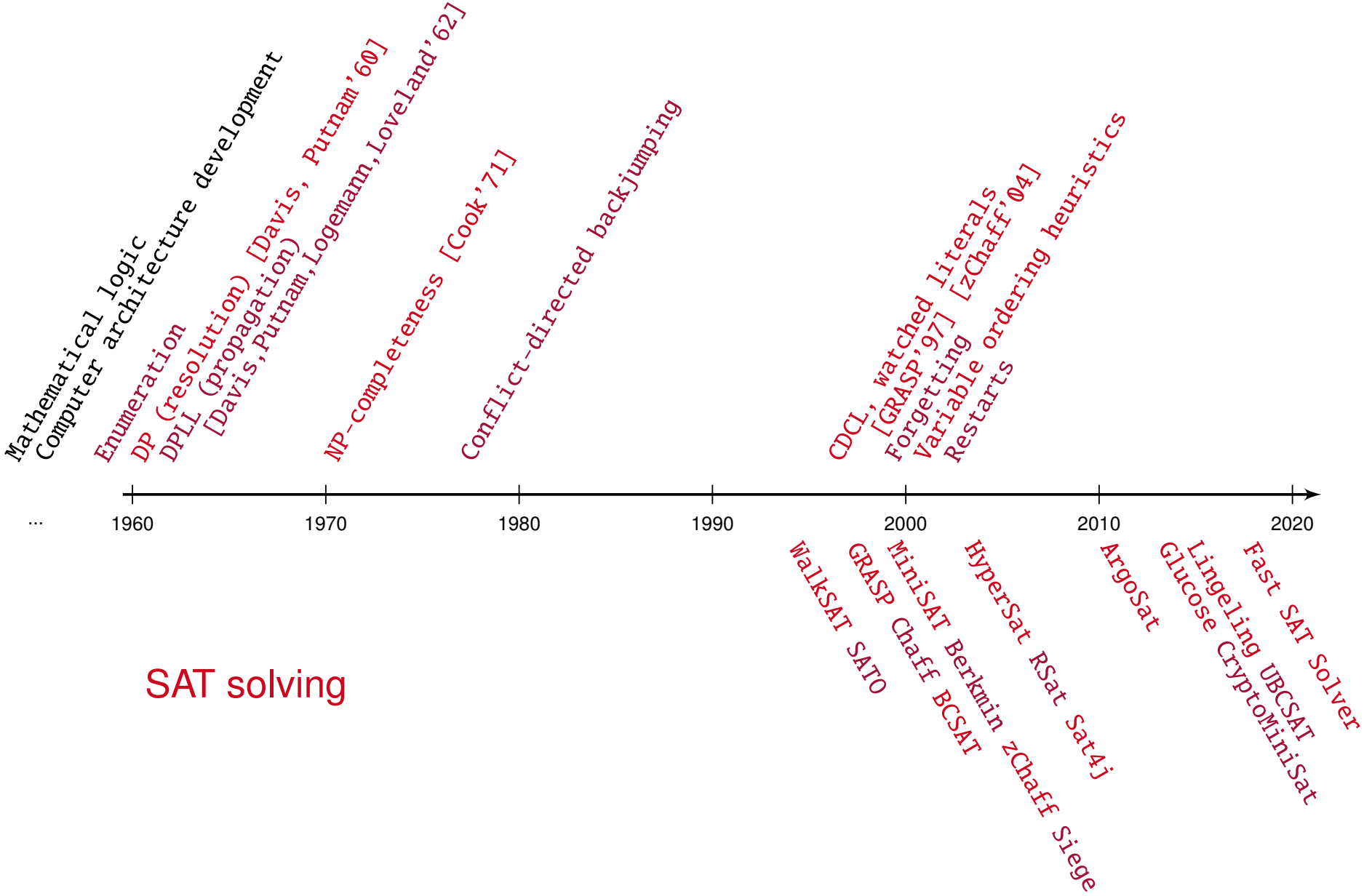
Today, practical problems with millions of variables are solvable!

An impression of the SAT solver development



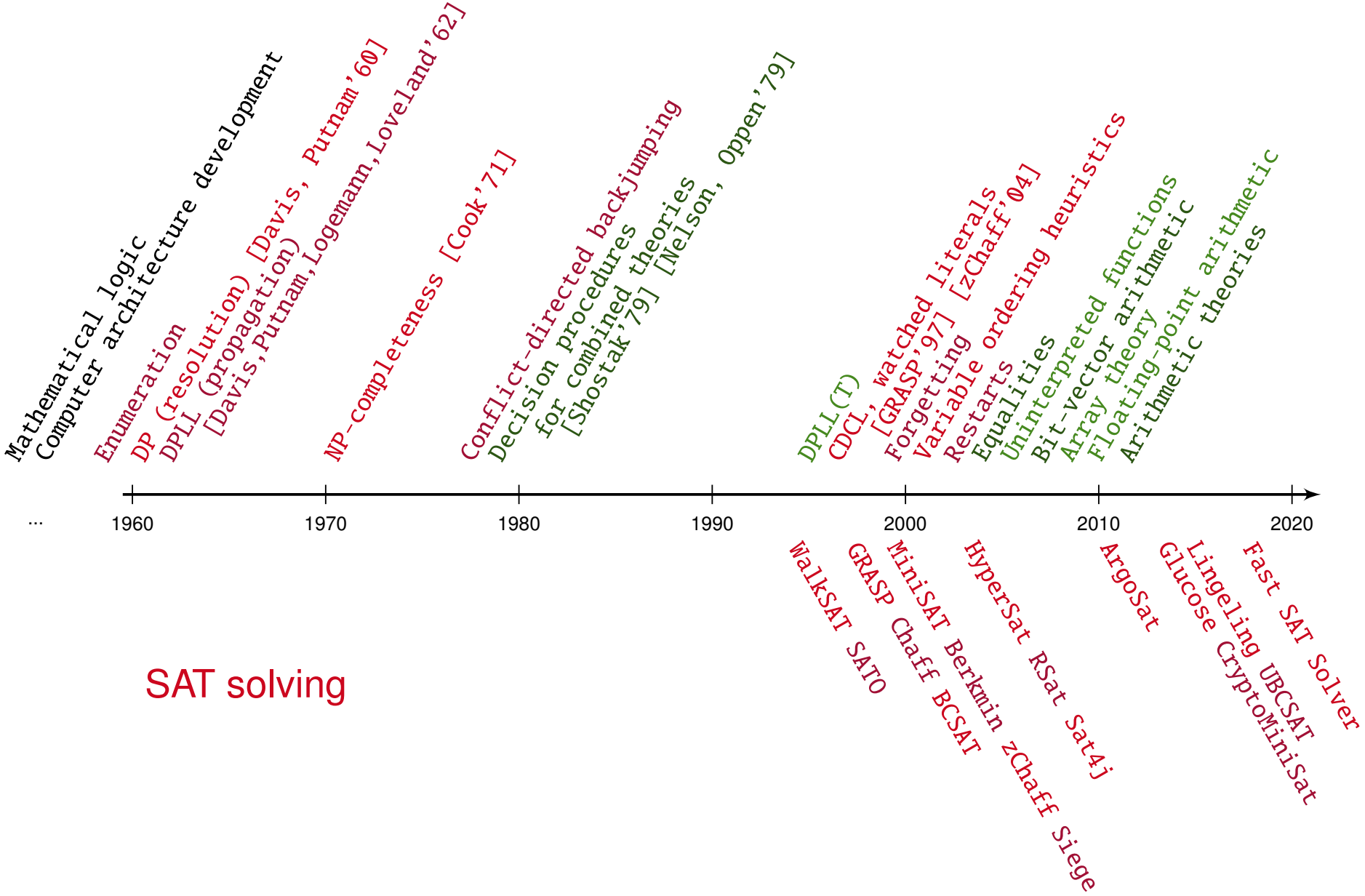
Source: Jarvisalo, Le Berre, Roussel, Simon. *The International SAT Solver Competitions*. AI Magazine, 2012.

Somemilestones in SAT/SMT development (incomplete!)

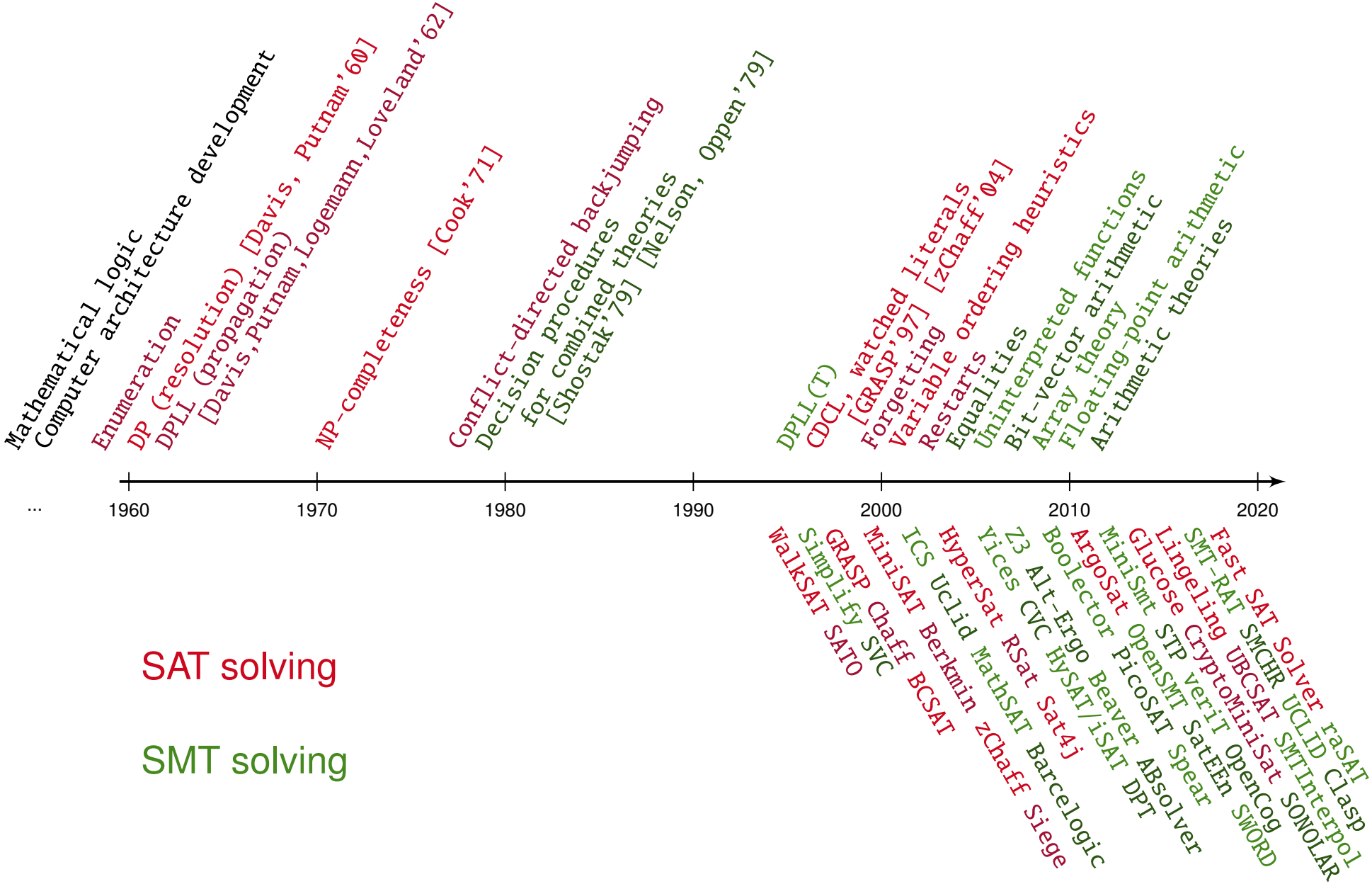


SAT solving

Somemilestones in SAT/SMT development (incomplete!)



Somemilestones in SAT/SMT development (incomplete!)



SAT solving

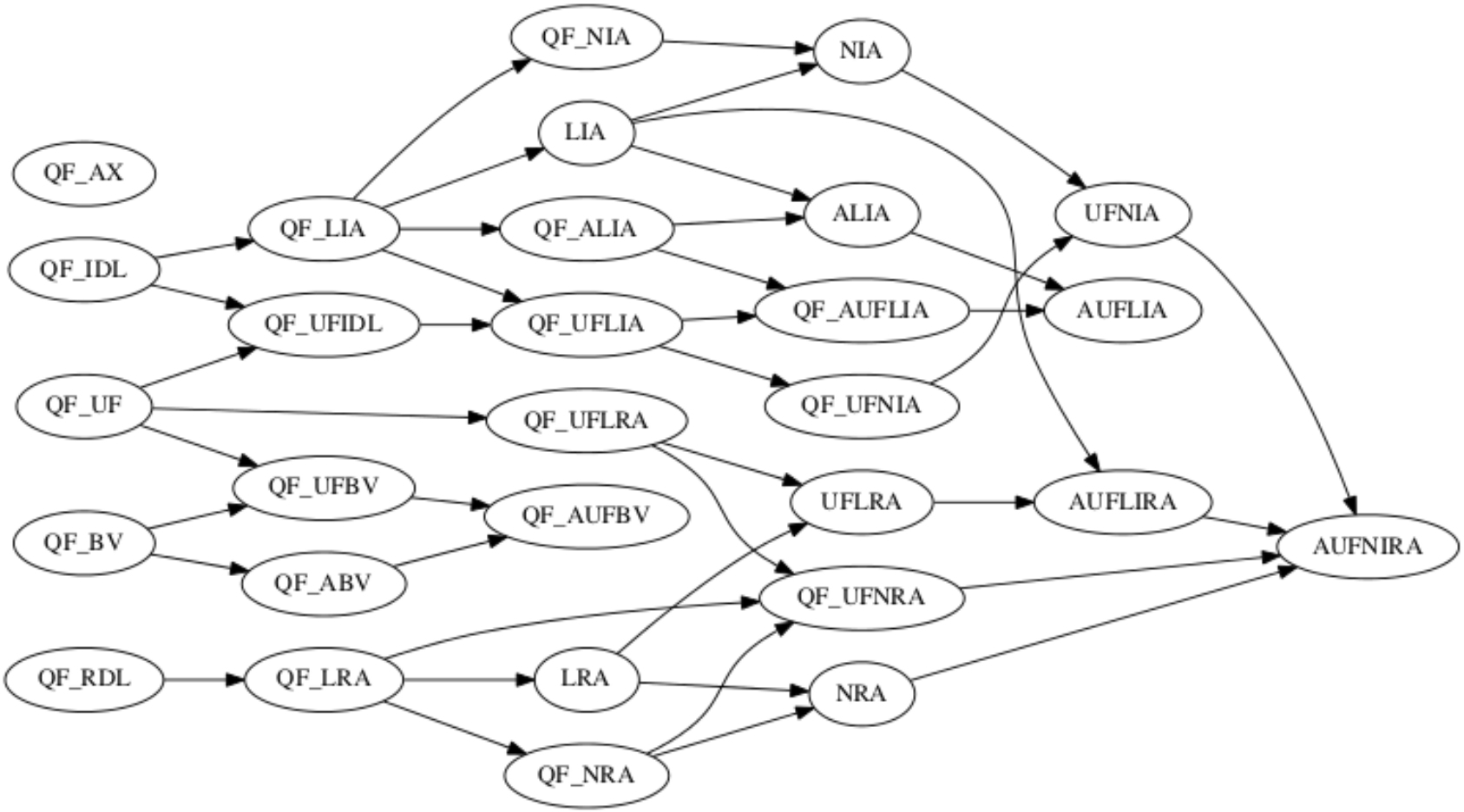
SMT solving

Satisfiability modulo theories solving

Active SMT community:

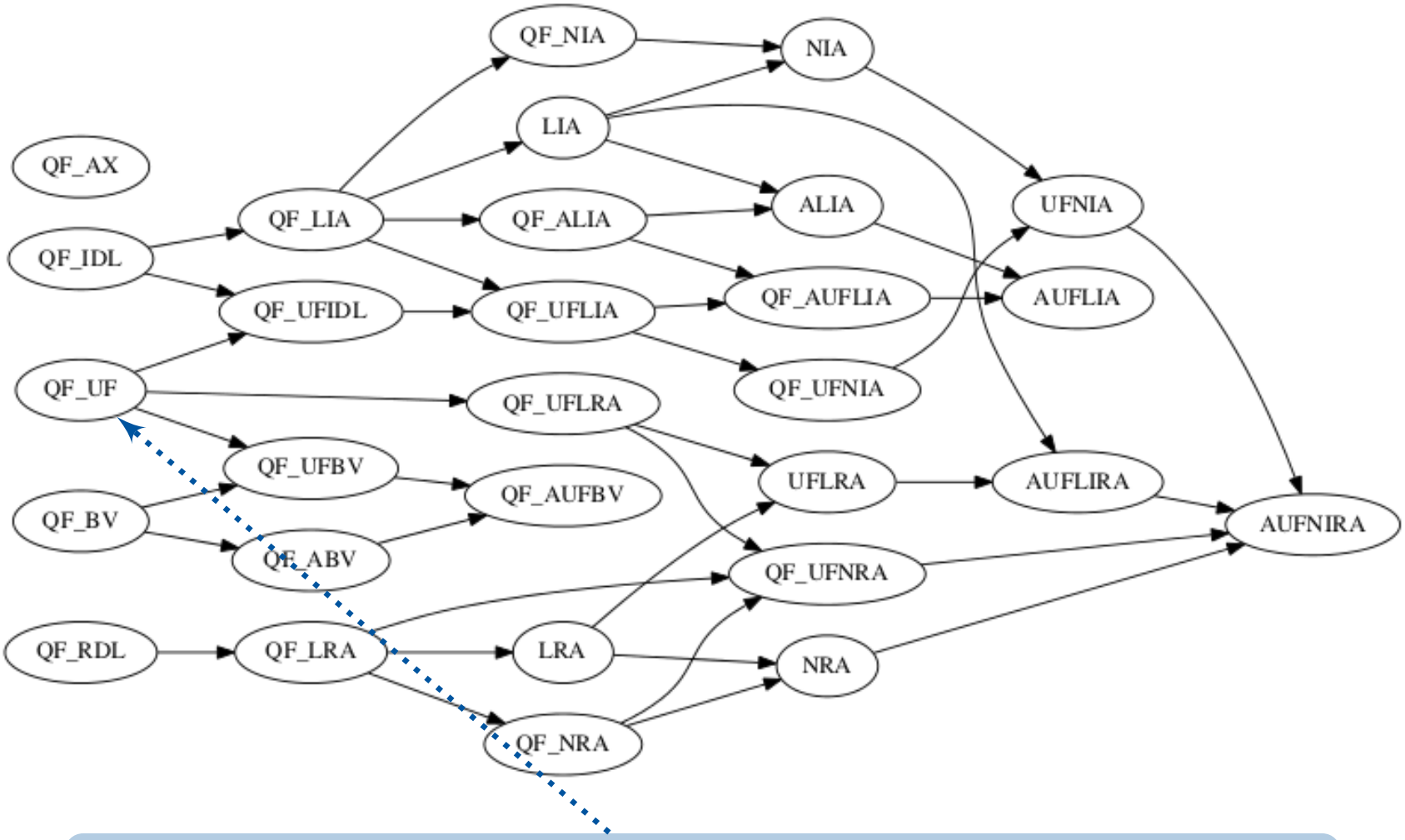
- SMT-LIB as standard input language since 2004.
- Competitions since 2005.
- SMT-COMP 2016 competition:
  - 4 tracks, 41 logical categories.
  - QF linear real arithmetic: $7 + 2$ solvers, 1626 benchmarks.
  - QF linear integer arithmetic: $6 + 2$ solvers, 5839 benchmarks.
  - QF non-linear real arithmetic: $5 + 1$ 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

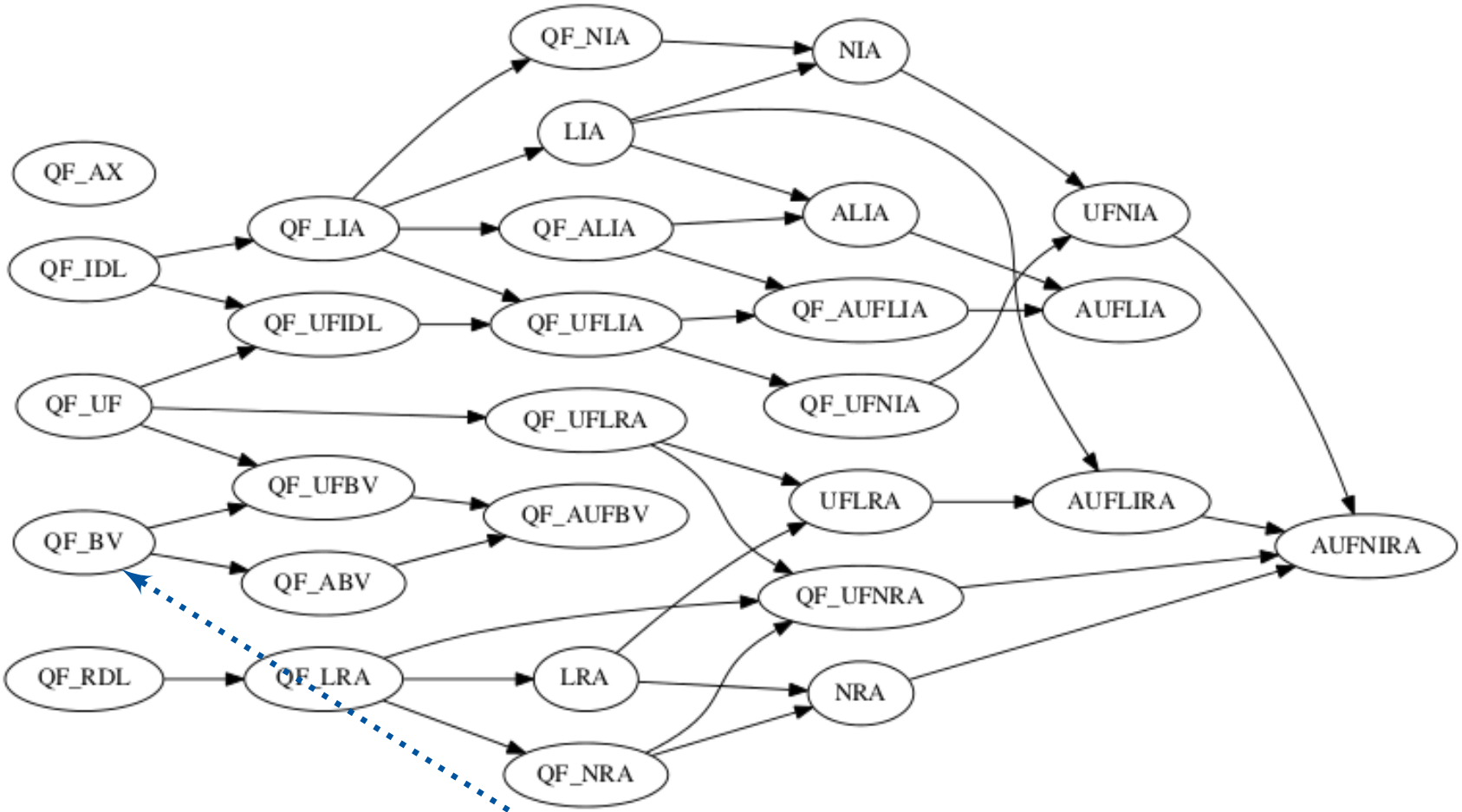


Quantifier-free equality logic with uninterpreted functions

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

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

SMT-LIB theories

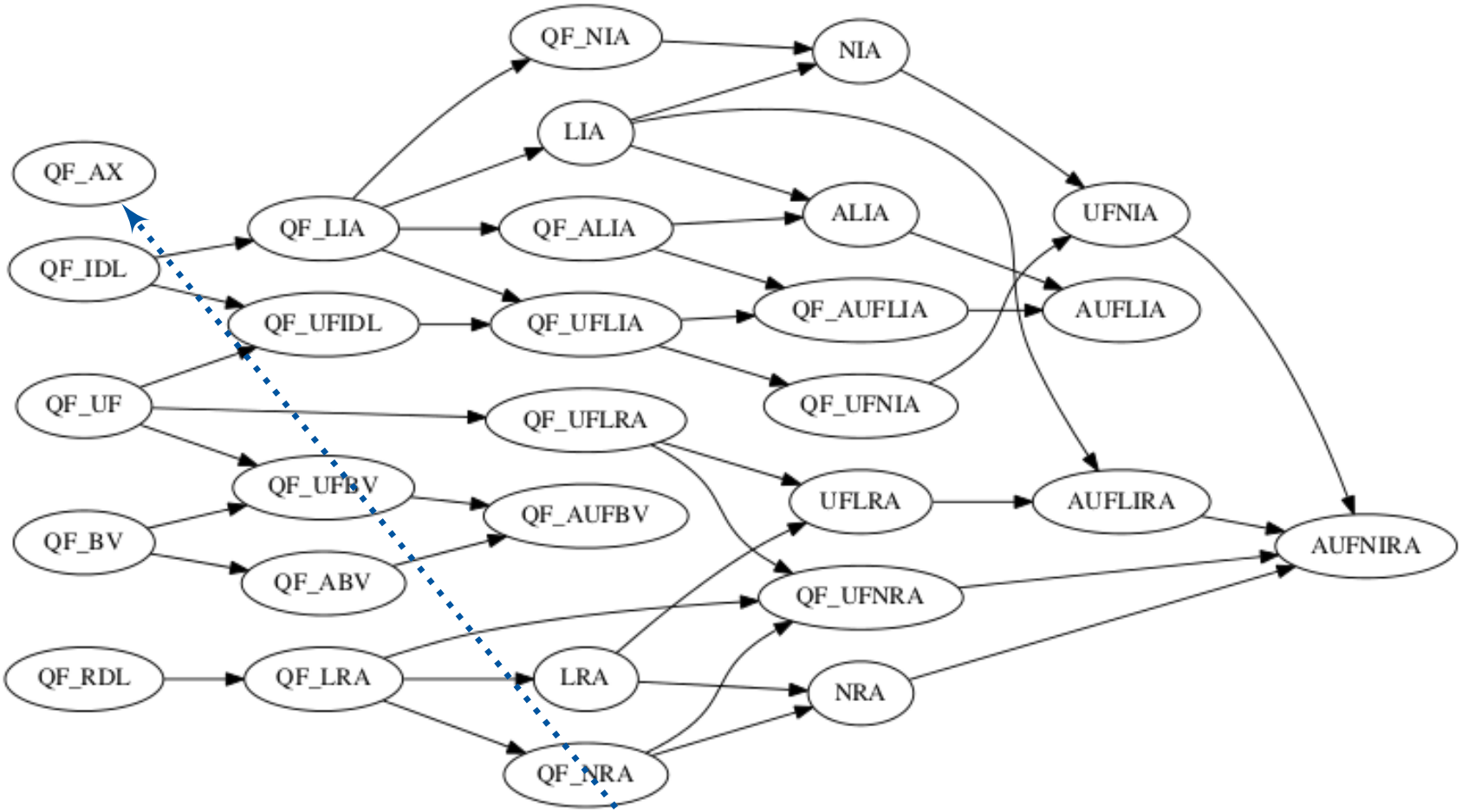


Quantifier-free bit-vector arithmetic

$$(a|b) \leq (a\&b)$$

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

SMT-LIB theories

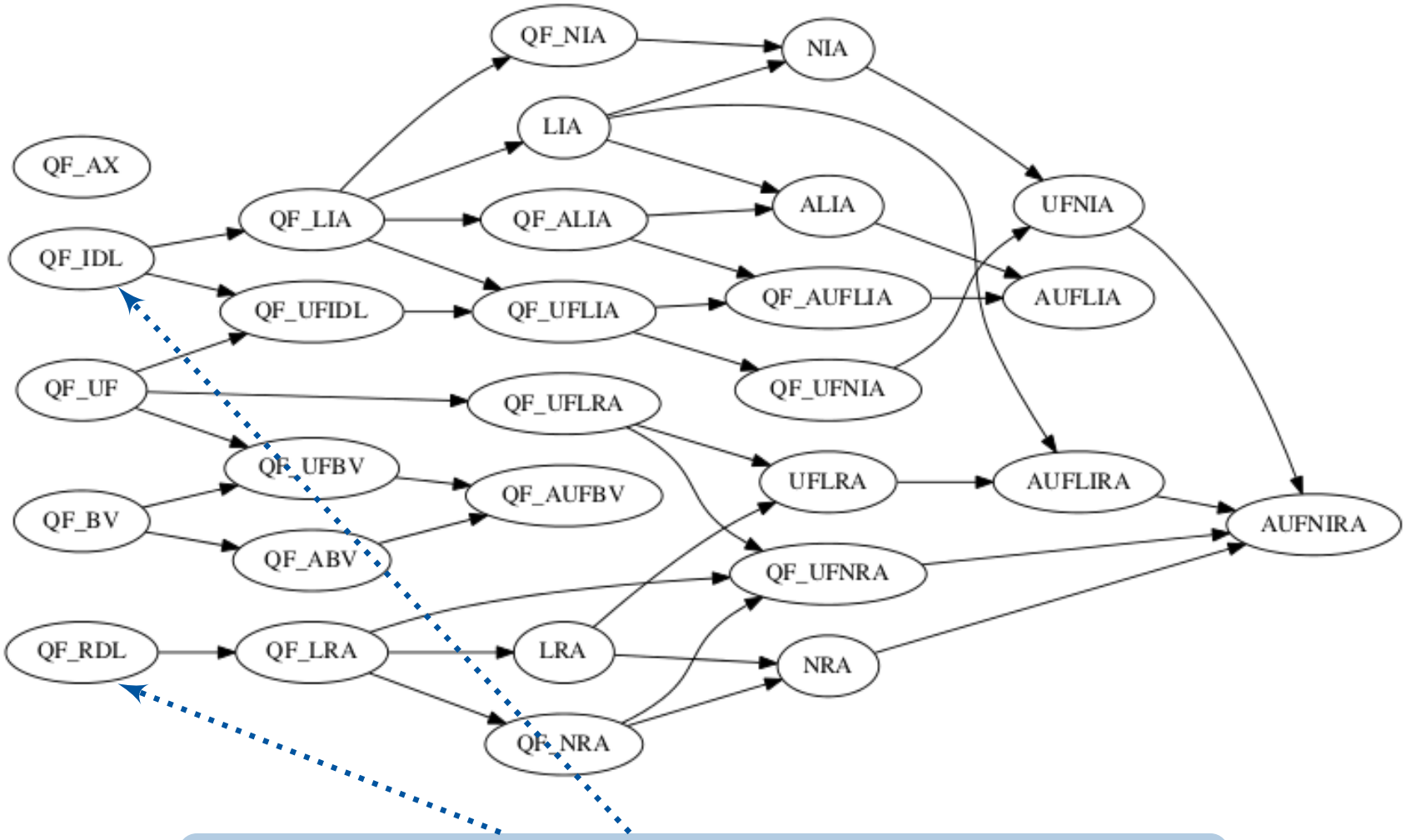


Quantifier-free array theory

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

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

SMT-LIB theories

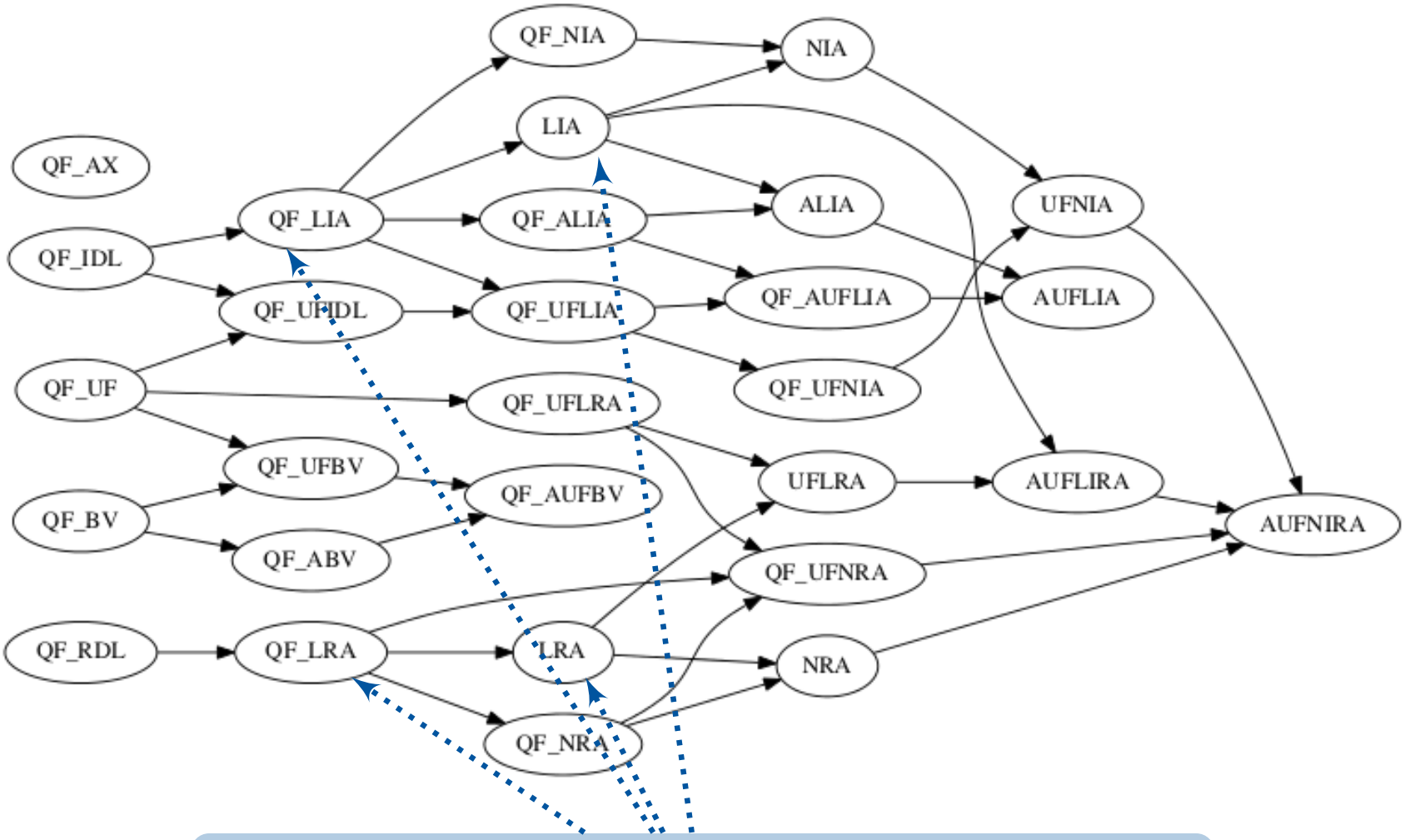


Quantifier-free integer/rational difference logic

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

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

SMT-LIB theories

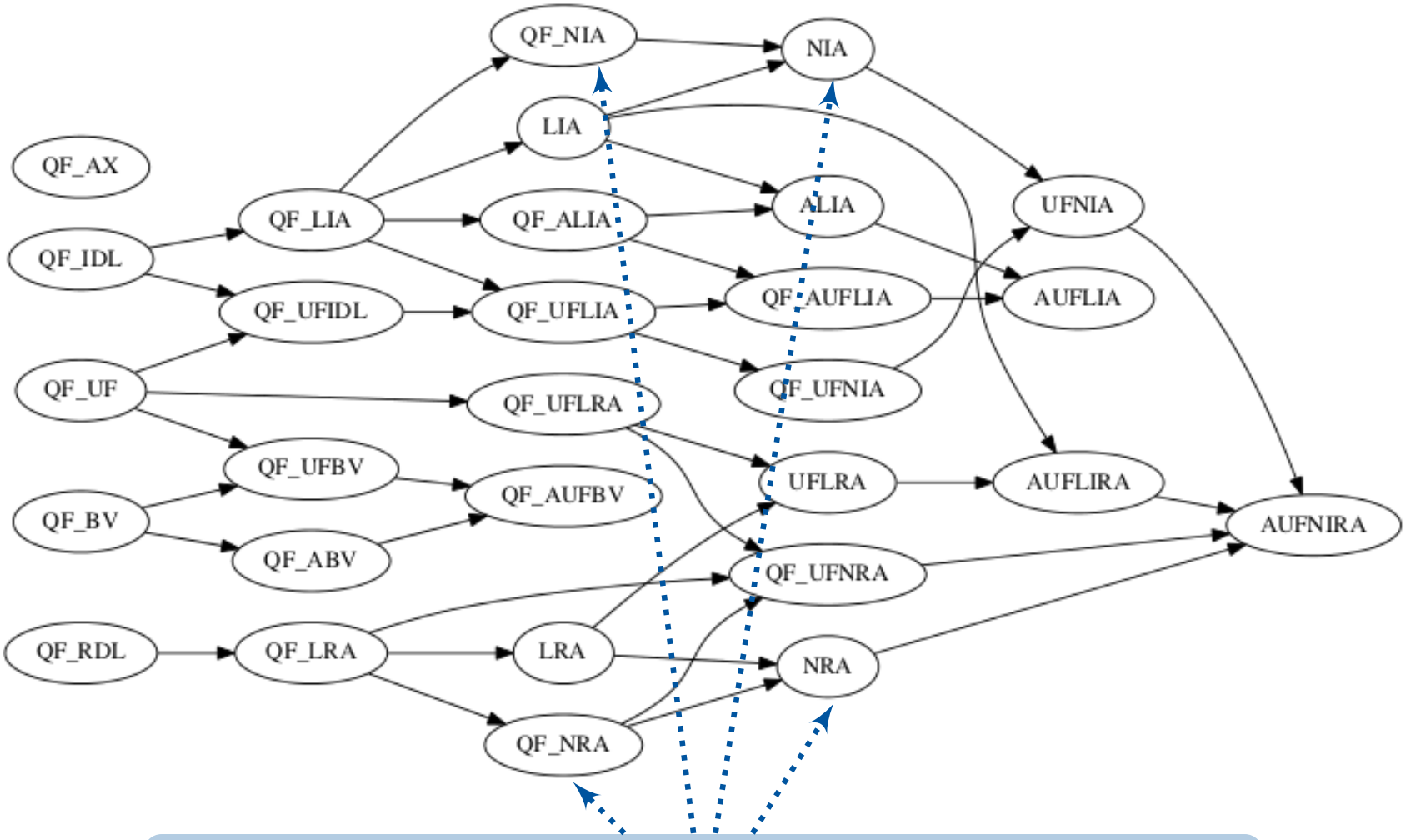


(Quantifier-free) real/integer linear arithmetic

$$3x + 7y = 8$$

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

SMT-LIB theories

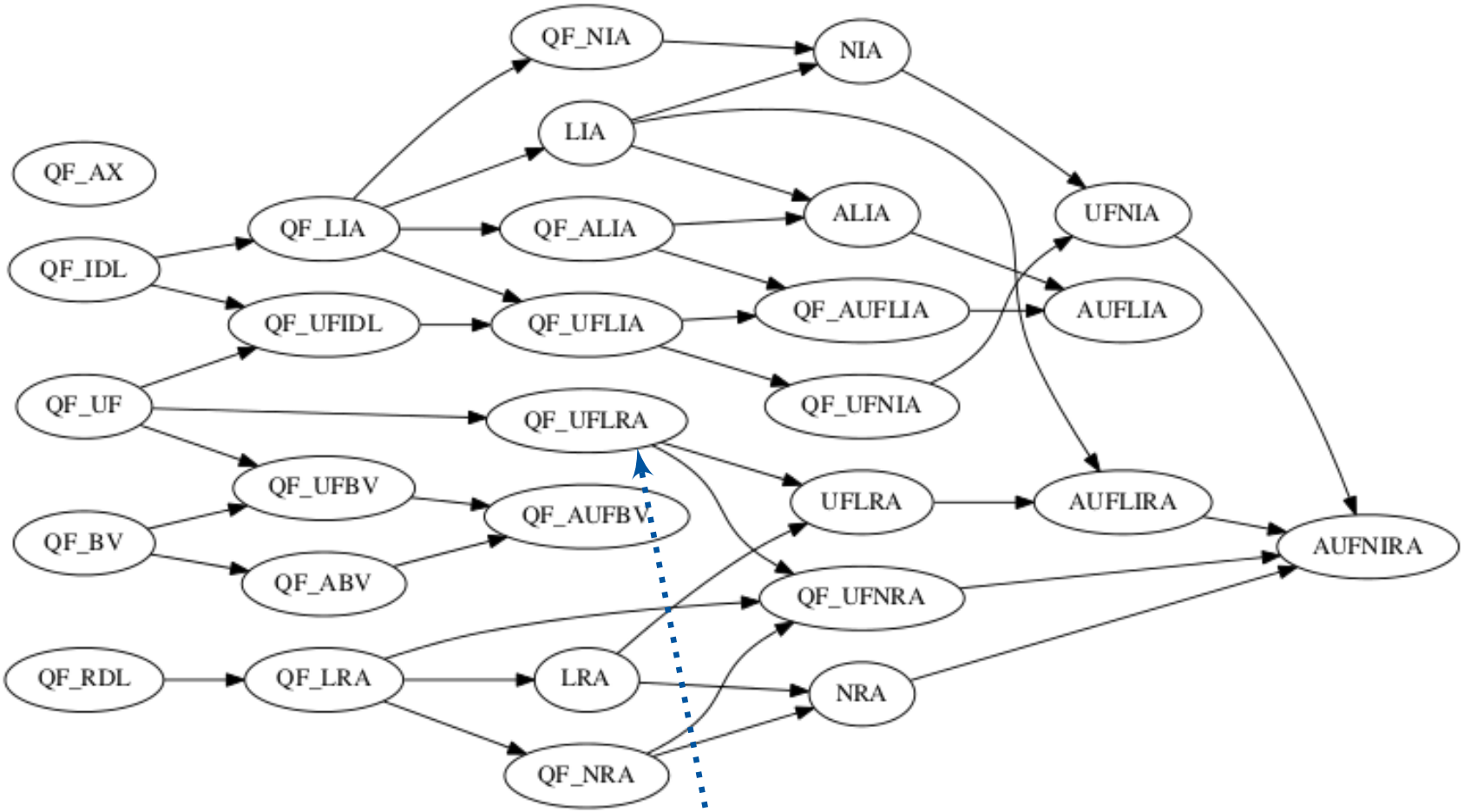


(Quantifier-free) real/integer non-linear arithmetic

$$x^2 + 2xy + y^2 \geq 0$$

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

SMT-LIB theories

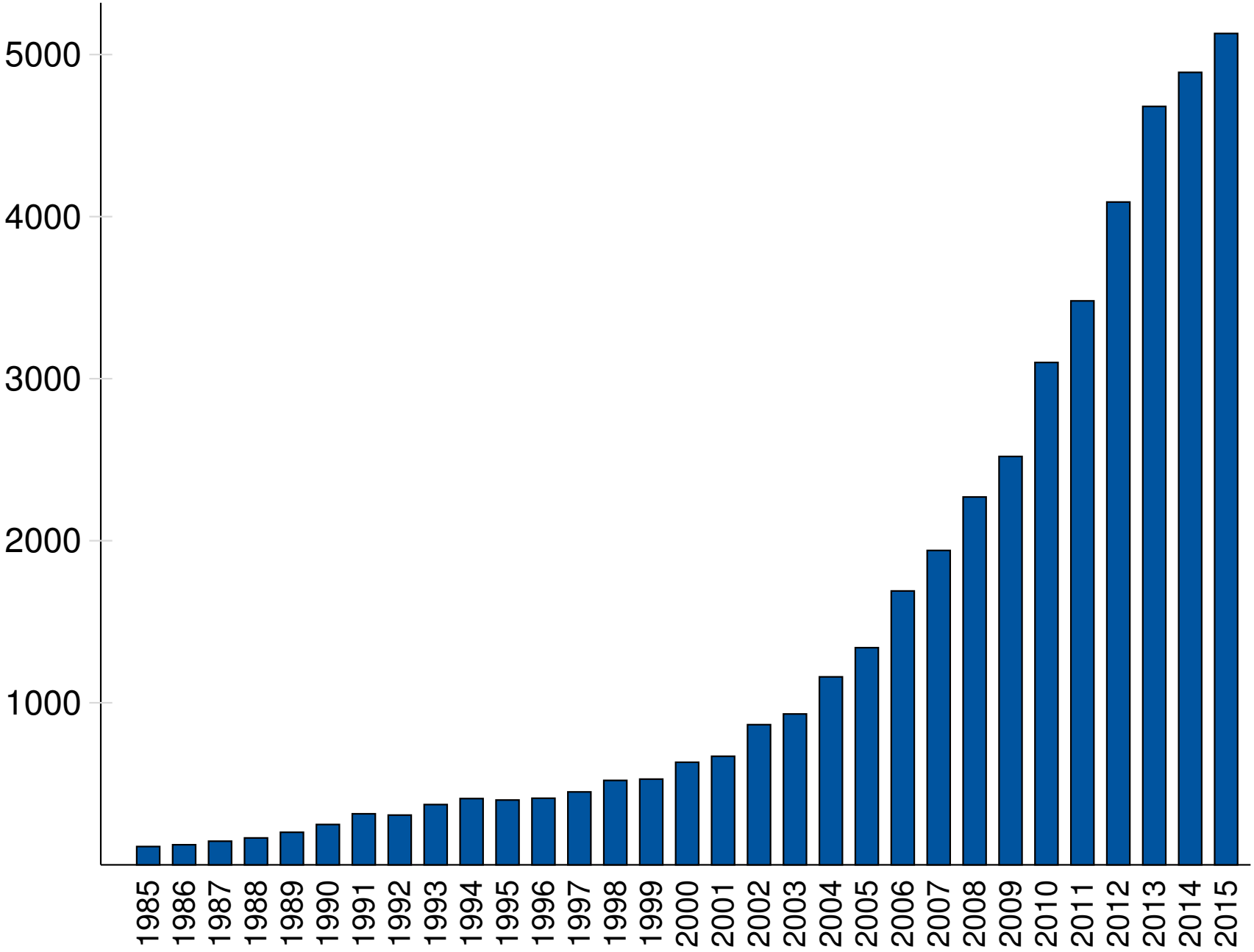


Combined theories

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

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

Google Scholar search for “SAT modulo theories”

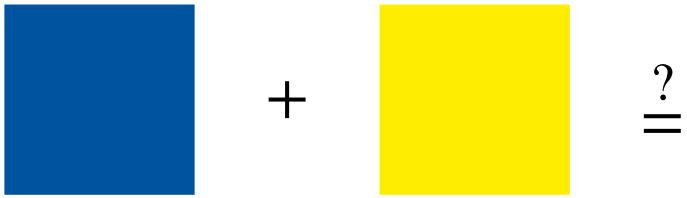


Part I:
Some historical notes

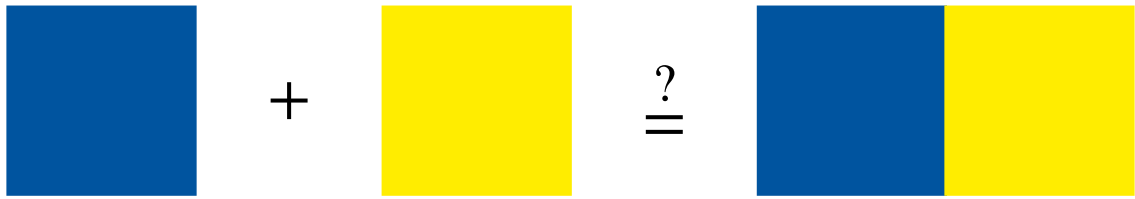
Part II:
One of the major ingredients of success:
Strategic combinations of decision procedures
...in SAT solving...
...in SMT solving...
...in theory solving.

Part III:
How satisfiability checking drives the digital world

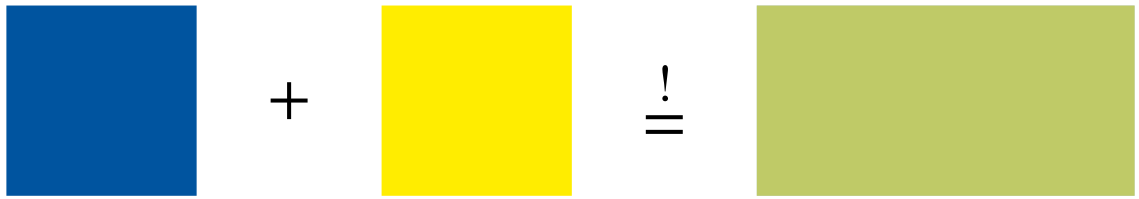
What does “strategic combination” mean?



What does “strategic combination” mean?



What does “strategic combination” mean?



Part I:
Some historical notes

Part II:
One of the major ingredients of success:
Strategic combinations of decision procedures
...in SAT solving...
...in SMT solving...
...in theory solving.

Part III:
How satisfiability checking drives the digital world

DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

$$
\begin{array}{lllllllll}
c_1: & (& \neg a & \vee & b & & &)\wedge \\
c_2: & (& & & \neg b & \vee & \neg c &)\wedge \\
c_3: & (& & & \neg b & \vee & c &)\wedge
\end{array}
$$

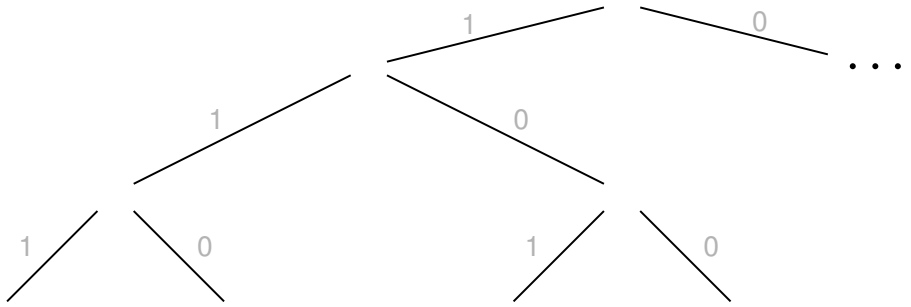
$\ldots$

DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration

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

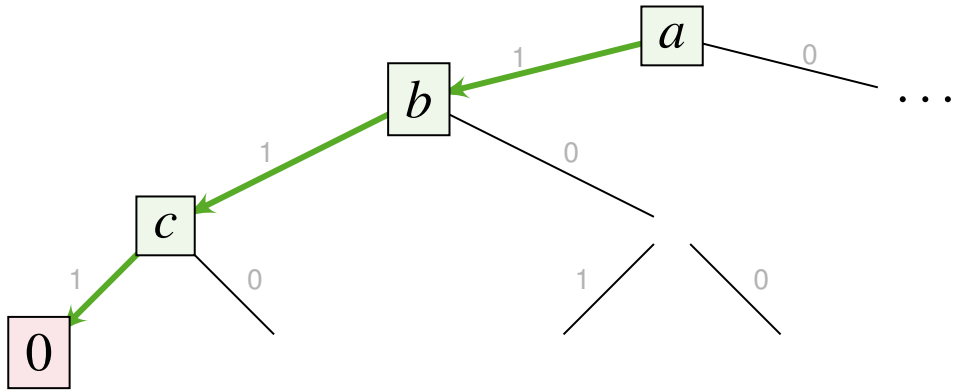


DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration

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

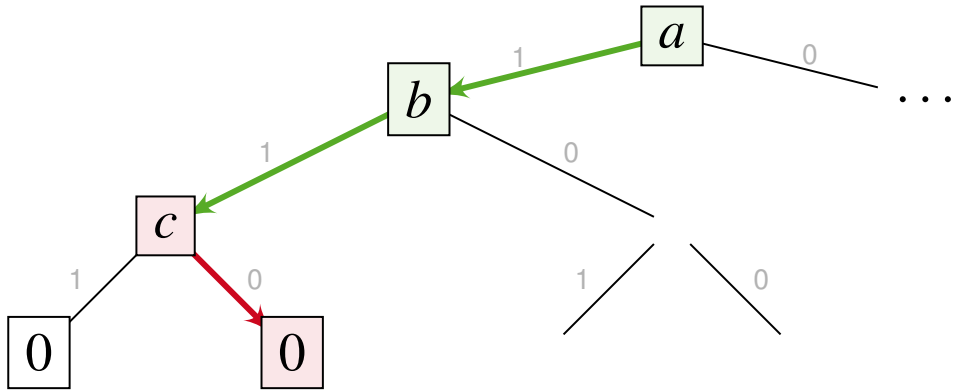


DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration

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

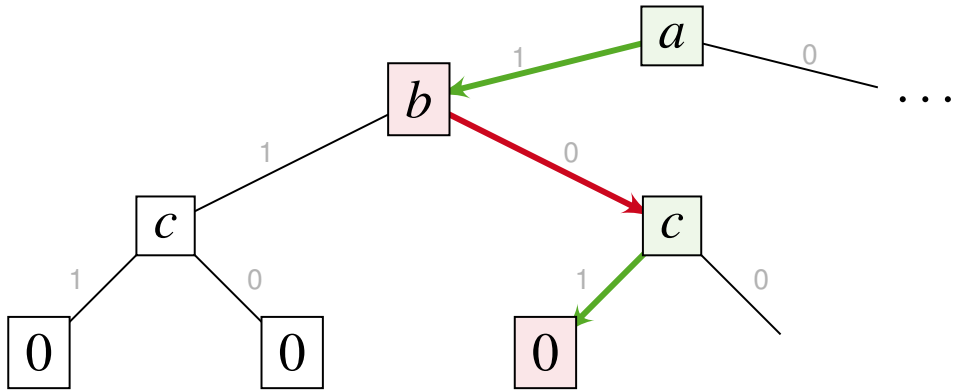


DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration

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

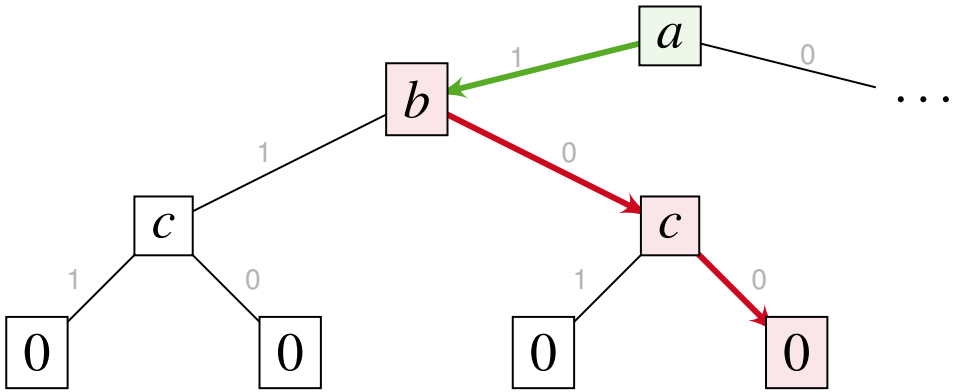


DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration

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

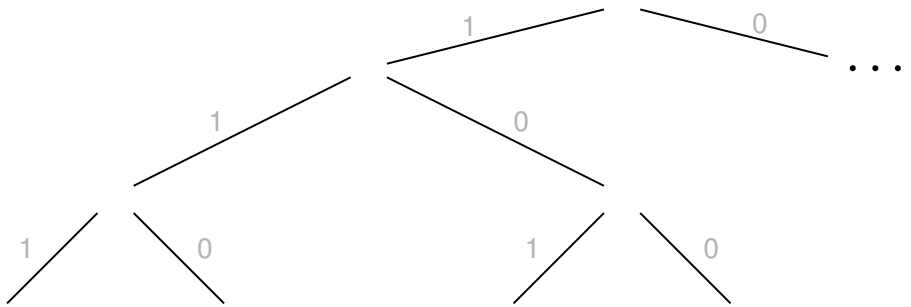


DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation

$$
\begin{array}{lllllll}
c_1: & (& \neg a & \vee & b & &)\wedge \\
c_2: & (& & & \neg b & \vee \neg c &)\wedge \\
c_3: & (& & & \neg b & \vee c &)\wedge \\
& \ldots & & & & &
\end{array}
$$

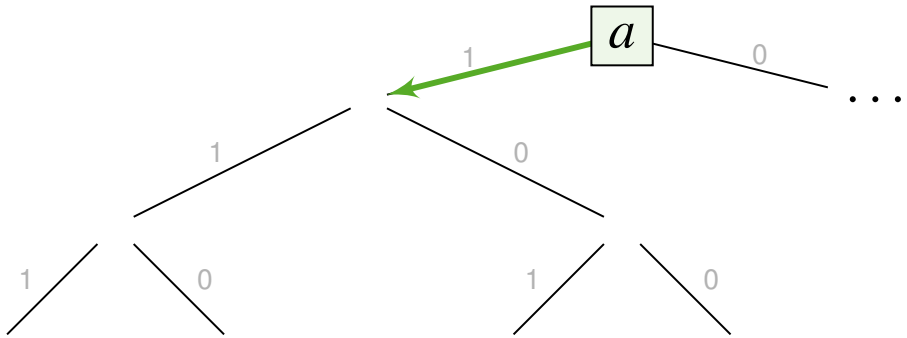


DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation

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

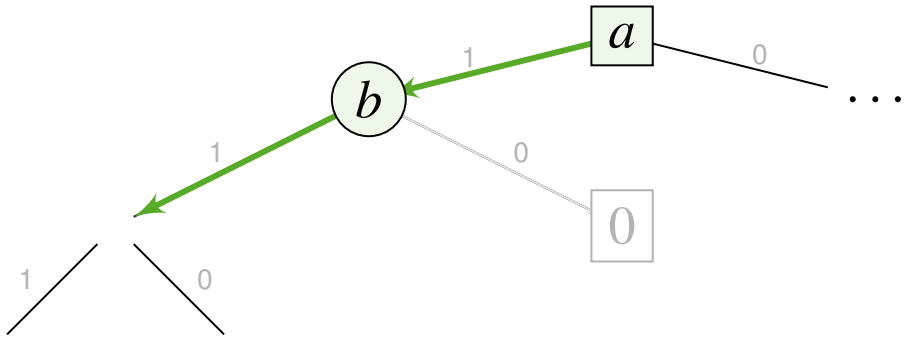


DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation

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

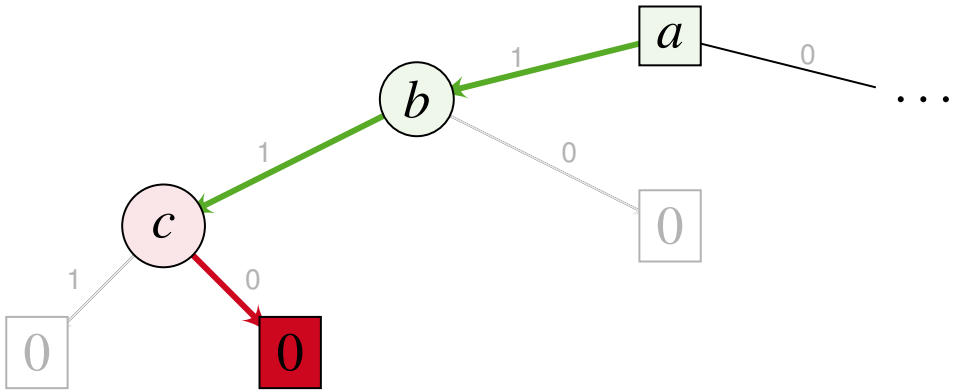


DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation

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

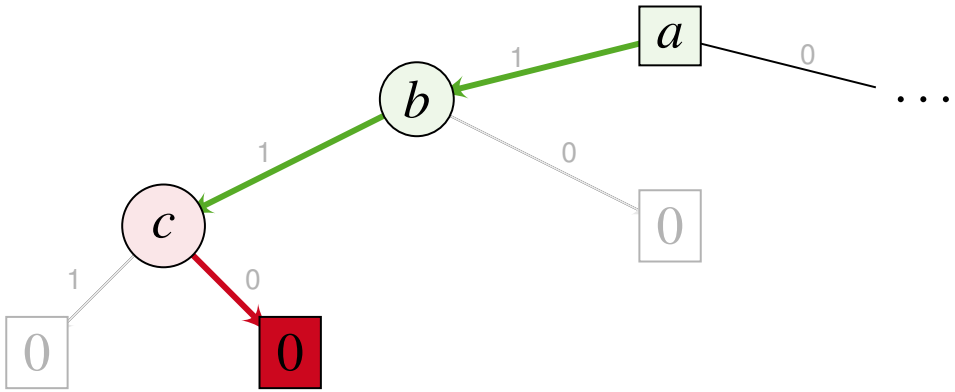


DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation + Resolution

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



Resolution

Assumption: conjunctive normal form (CNF)

Resolution

Assumption: conjunctive normal form (CNF)

Derivation rule form:

$$\frac{antecendent_1 \quad \ldots \quad antecendent_n}{consequent}\, Rule\_name$$

Resolution

Assumption: conjunctive normal form (CNF)

Derivation rule form:

$$\frac{\mathit{antecendent}_1 \quad \ldots \quad \mathit{antecendent}_n}{\mathit{consequent}}\ \mathit{Rule\_name}$$

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

Resolution

Assumption: conjunctive normal form (CNF)

Derivation rule form:

$$\frac{\mathit{antecedent}_1 \quad \ldots \quad \mathit{antecedent}_n}{\mathit{consequent}}\ \mathit{Rule\_name}$$

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

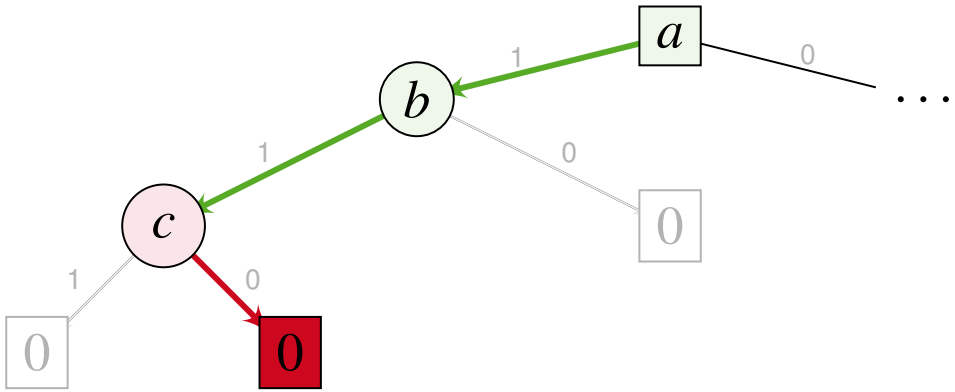
$$\exists x.\ C_x \wedge C_{\neg x} \wedge C \quad \leftrightarrow \quad \mathit{Resolvents}\,(C_x, C_{\neg x}) \wedge C$$

DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation + Resolution

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

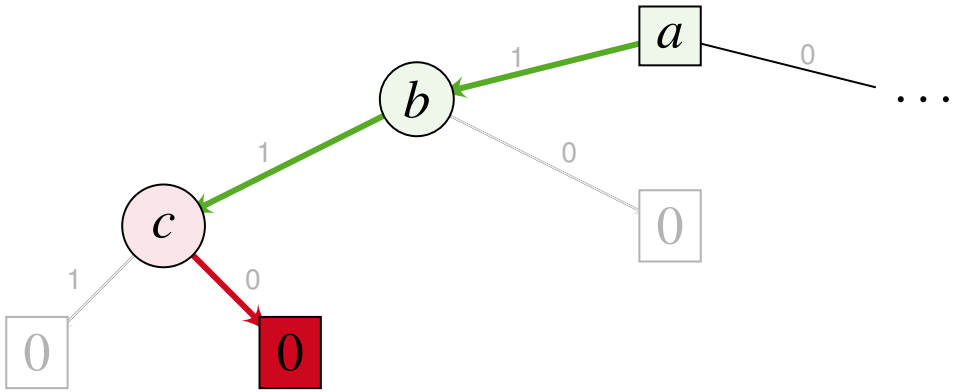


DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation + Resolution

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



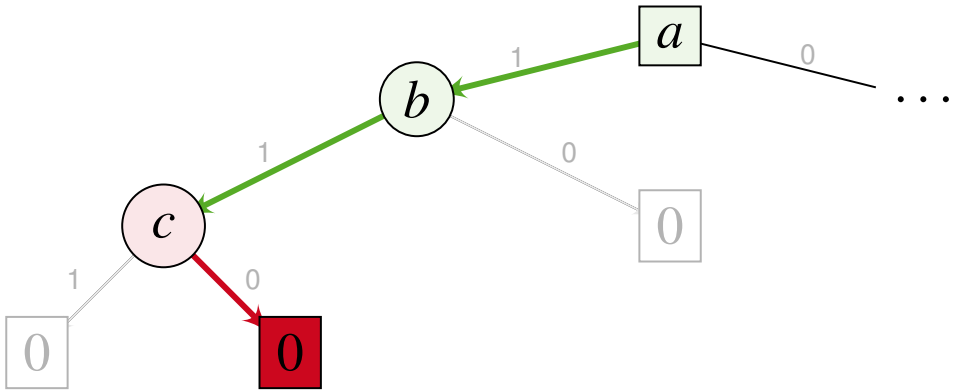
$$\frac{c_3 : (\neg b \vee \mathbf{c}) \quad c_2 : (\neg b \vee \neg \mathbf{c})}{c_4 : (\neg b)} \quad \textit{Resolution}$$

DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation + Resolution

$$
\begin{array}{lllllll}
c_1: & (\ \neg a & \vee & b & & &)\wedge \\
c_2: & (\ & & \neg b & \vee & \neg c &)\wedge \\
c_3: & (\ & & \neg b & \vee & c &)\wedge \\
c_4: & (\ & & \neg b & & &)\wedge \\
\multicolumn{7}{l}{\ldots}
\end{array}
$$



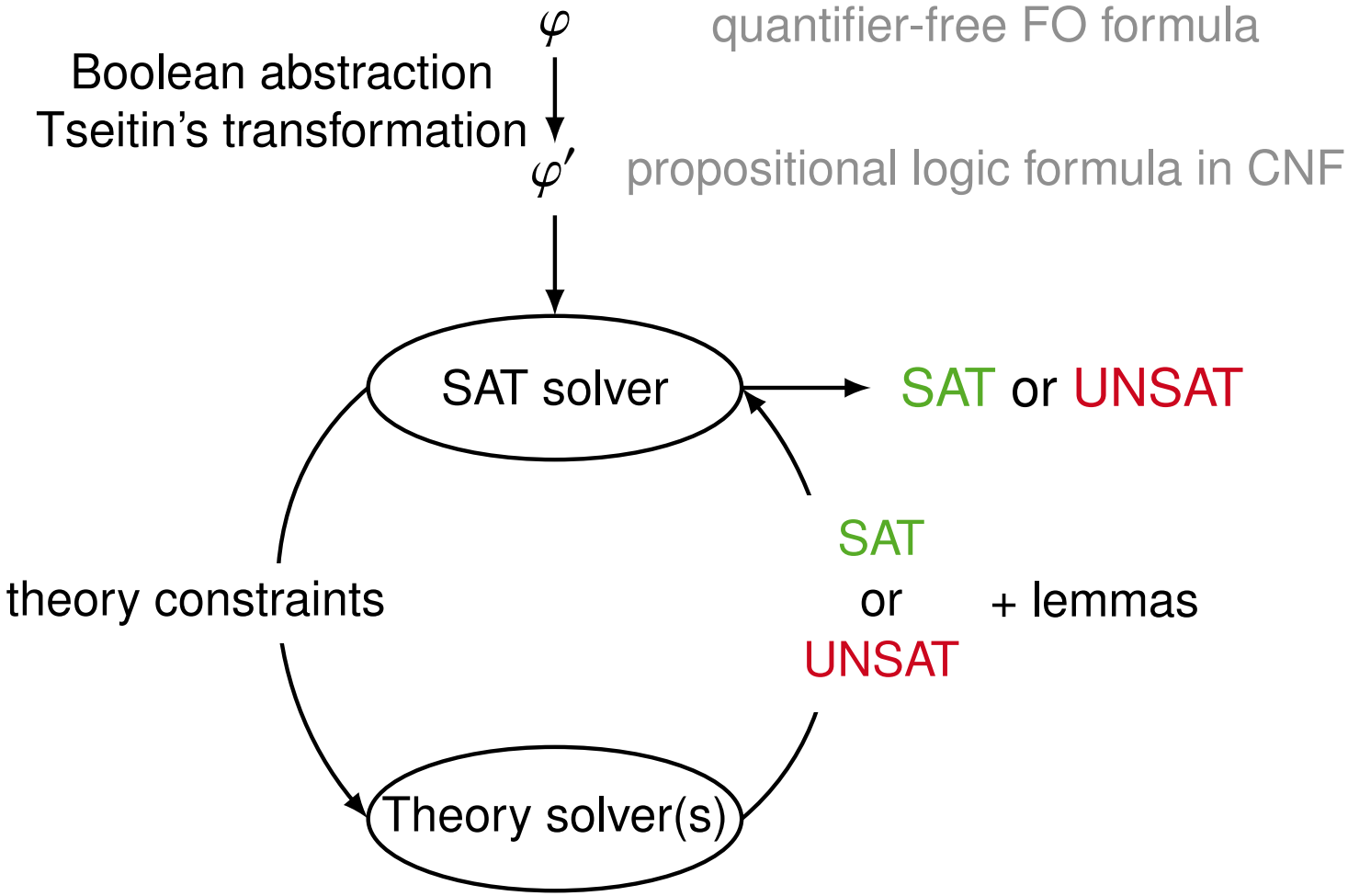
$$\frac{c_3 : (\neg b \vee \mathbf{c}) \quad c_2 : (\neg b \vee \neg \mathbf{c})}{c_4 : (\neg b)}$$ *Resolution*

Part I:
Some historical notes

Part II:
One of the major ingredients of success:
Strategic combinations of decision procedures
...in SAT solving...
...in SMT solving...
...in theory solving.

Part III:
How satisfiability checking drives the digital world

(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

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

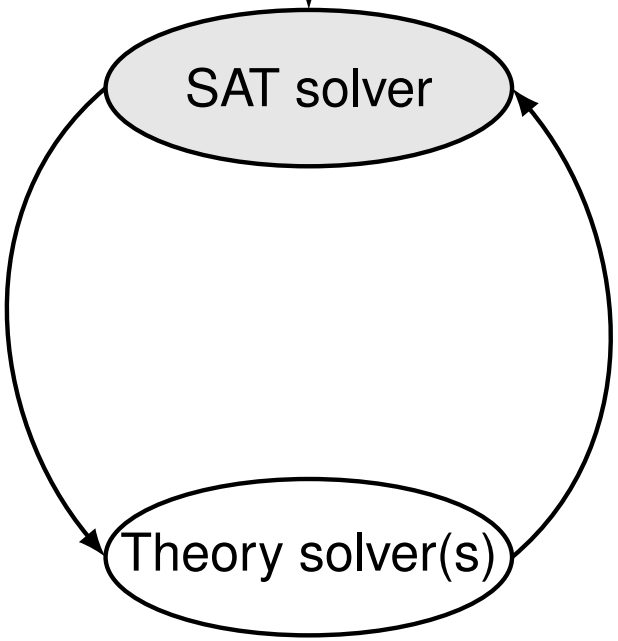
$$\downarrow$$

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

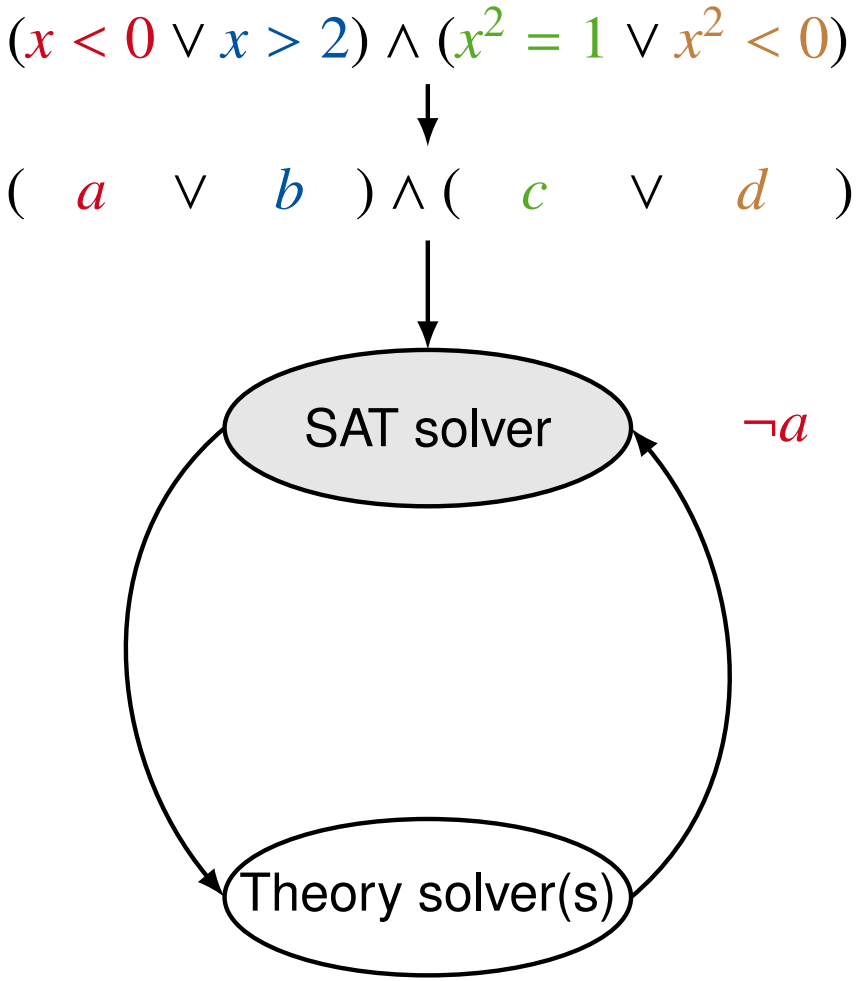
Less lazy SMT solving

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

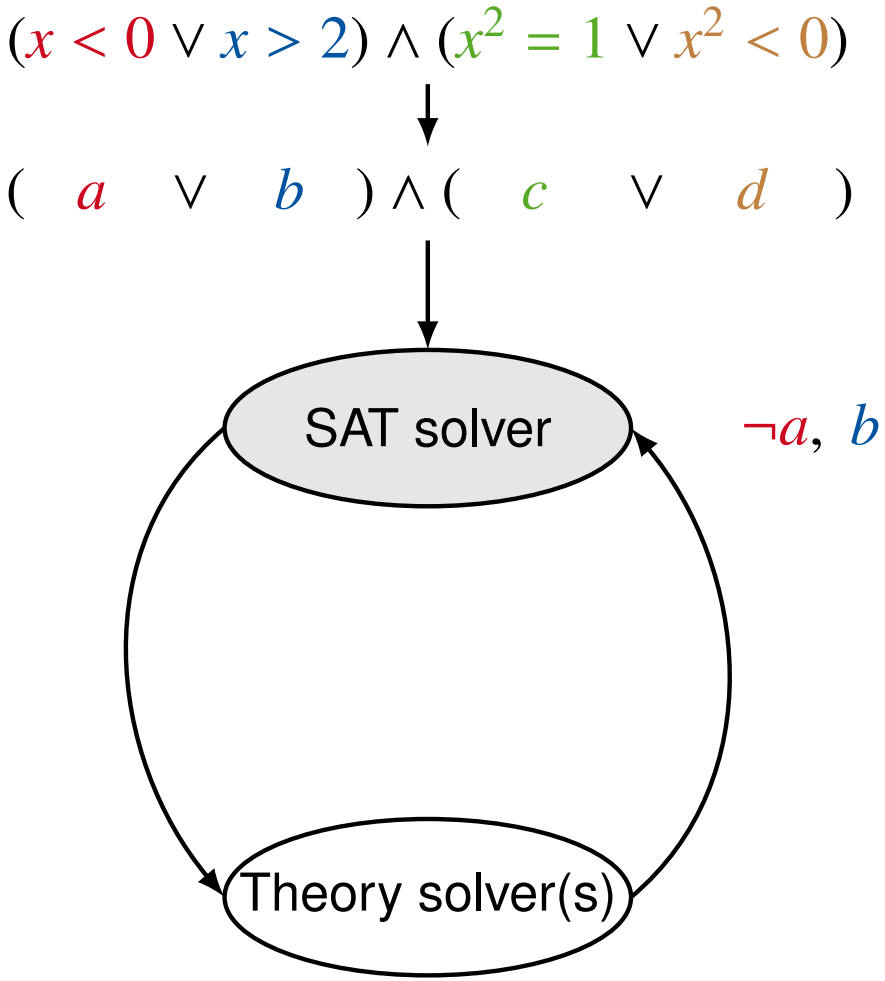
$$(\ a\ \vee\ b\ ) \wedge (\ c\ \vee\ d\ )$$



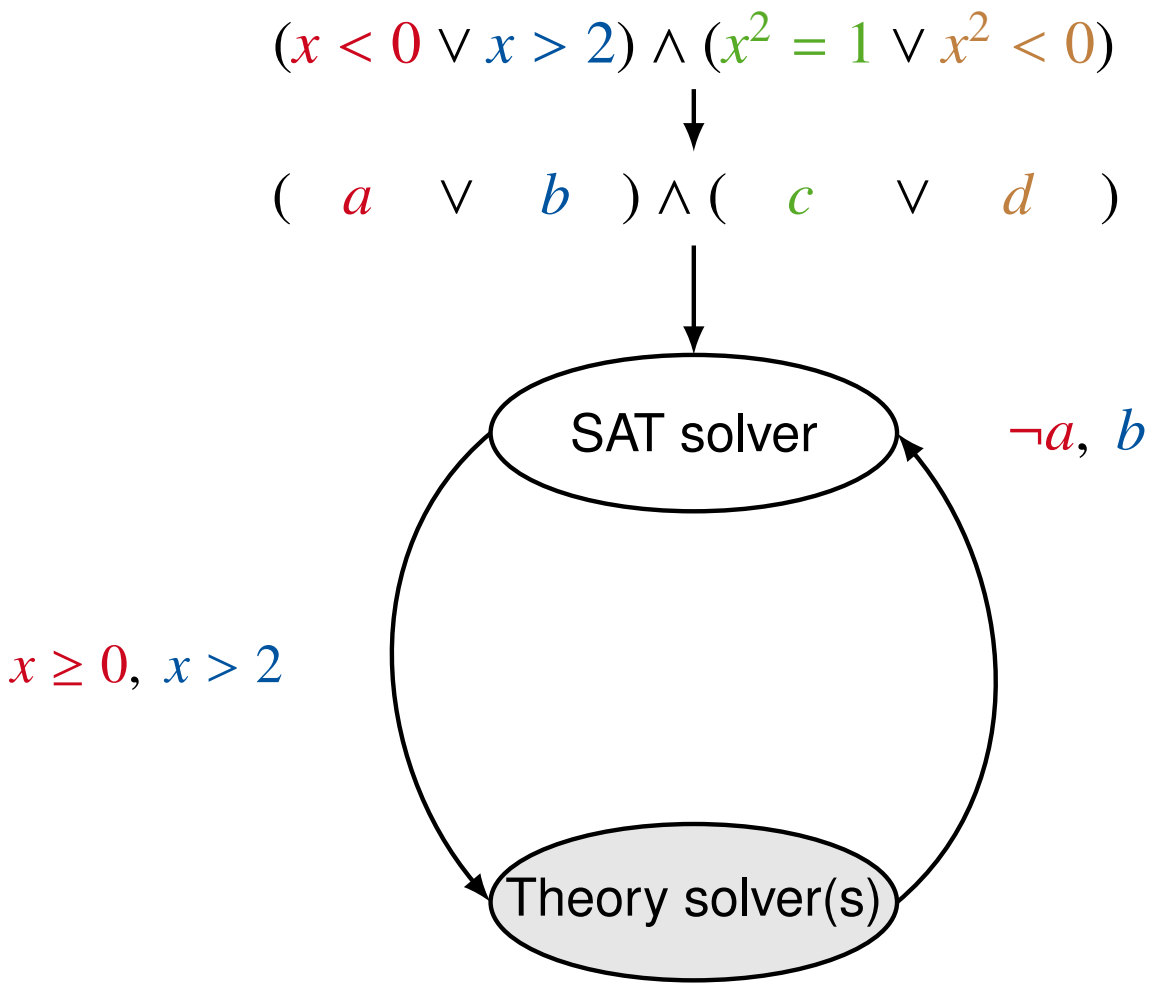
Less lazy SMT solving



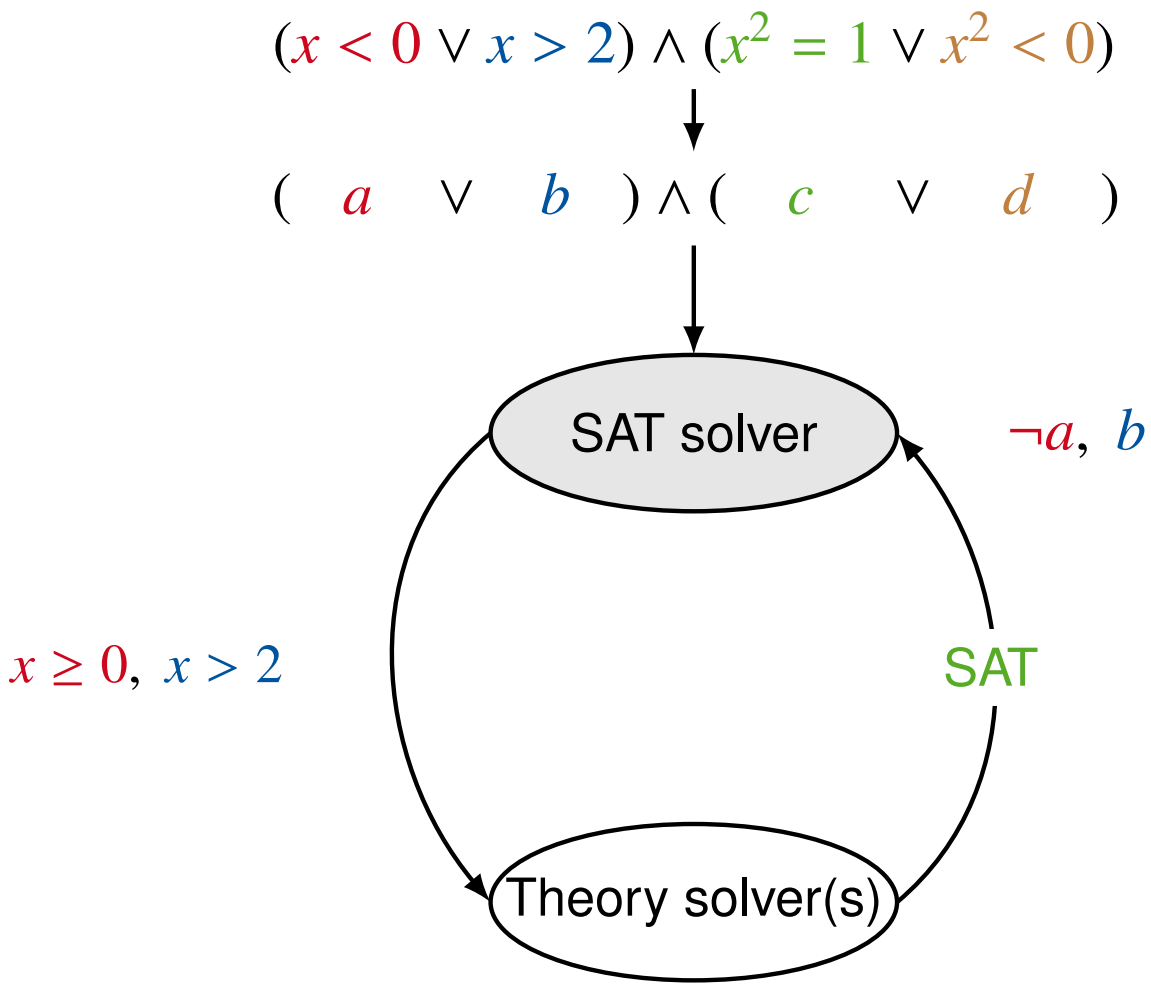
Less lazy SMT solving



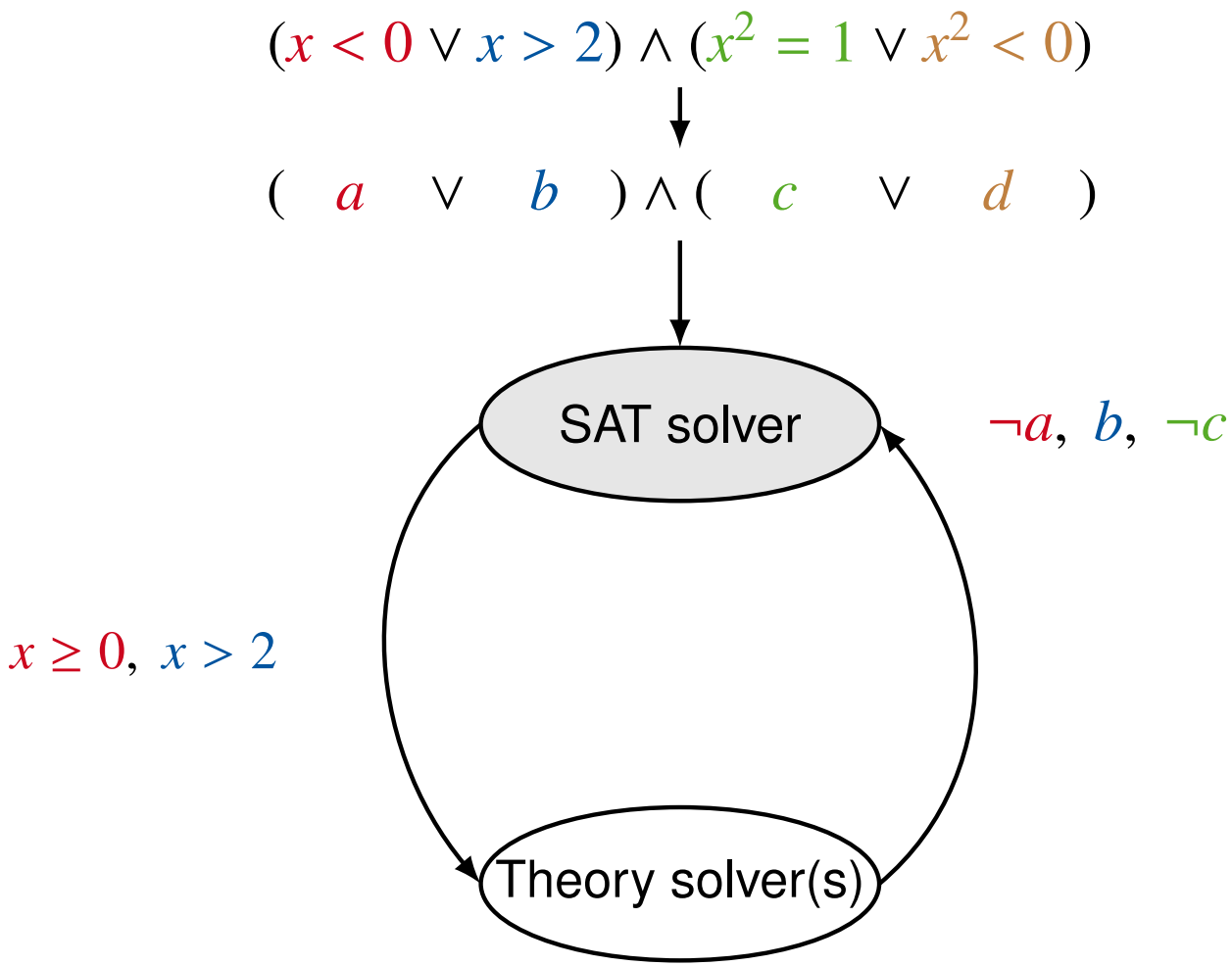
Less lazy SMT solving



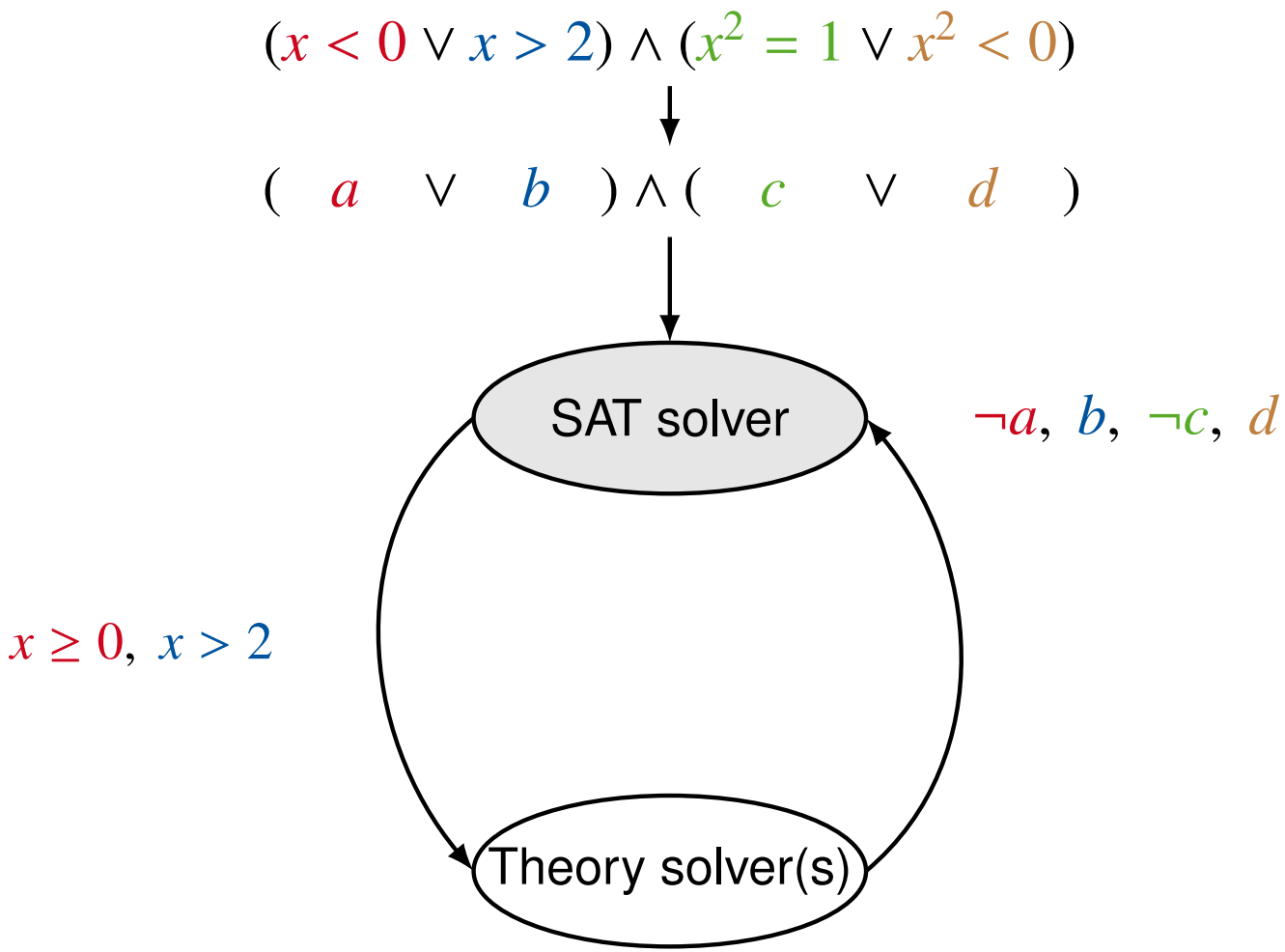
Less lazy SMT solving



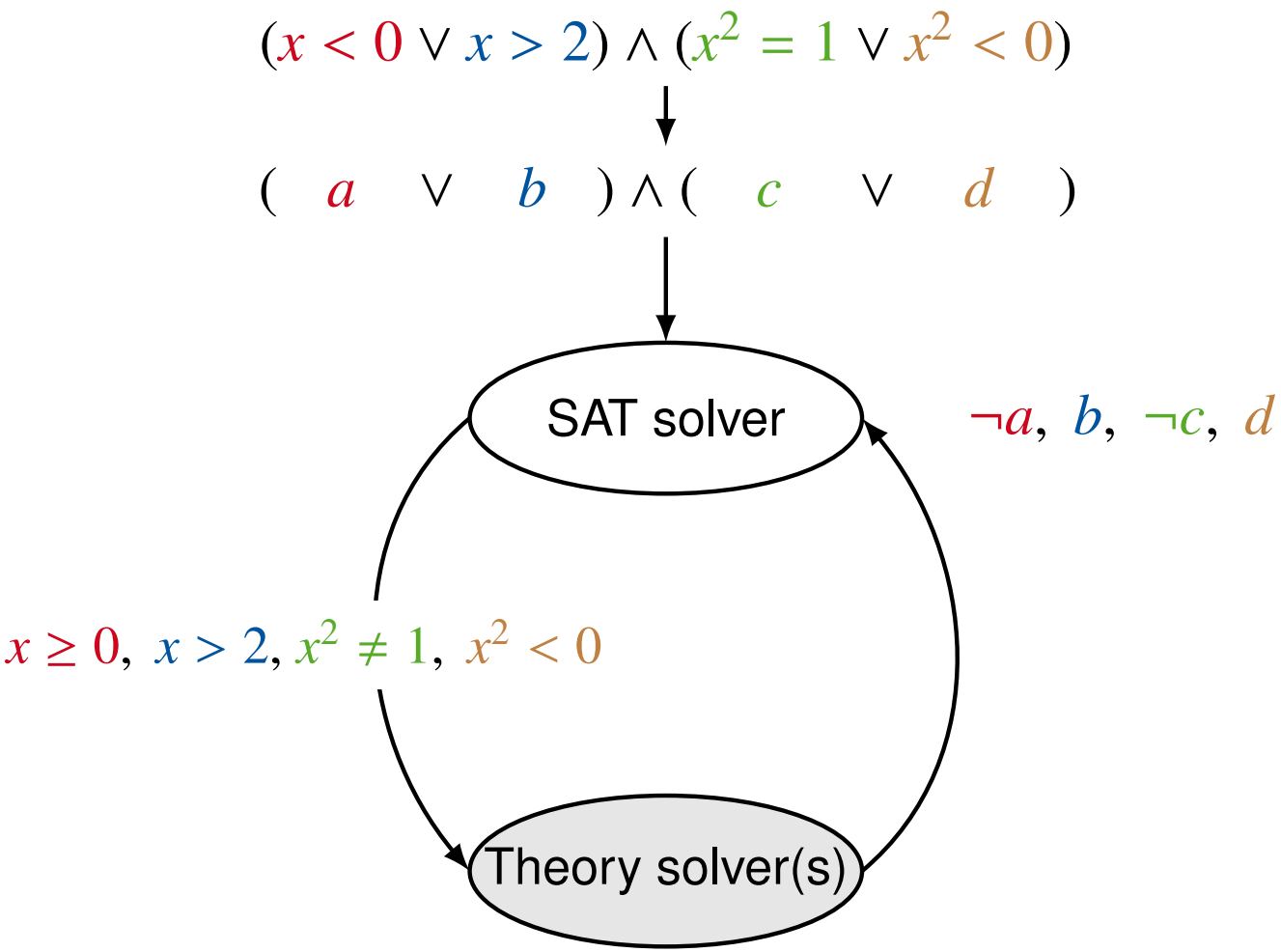
Less lazy SMT solving



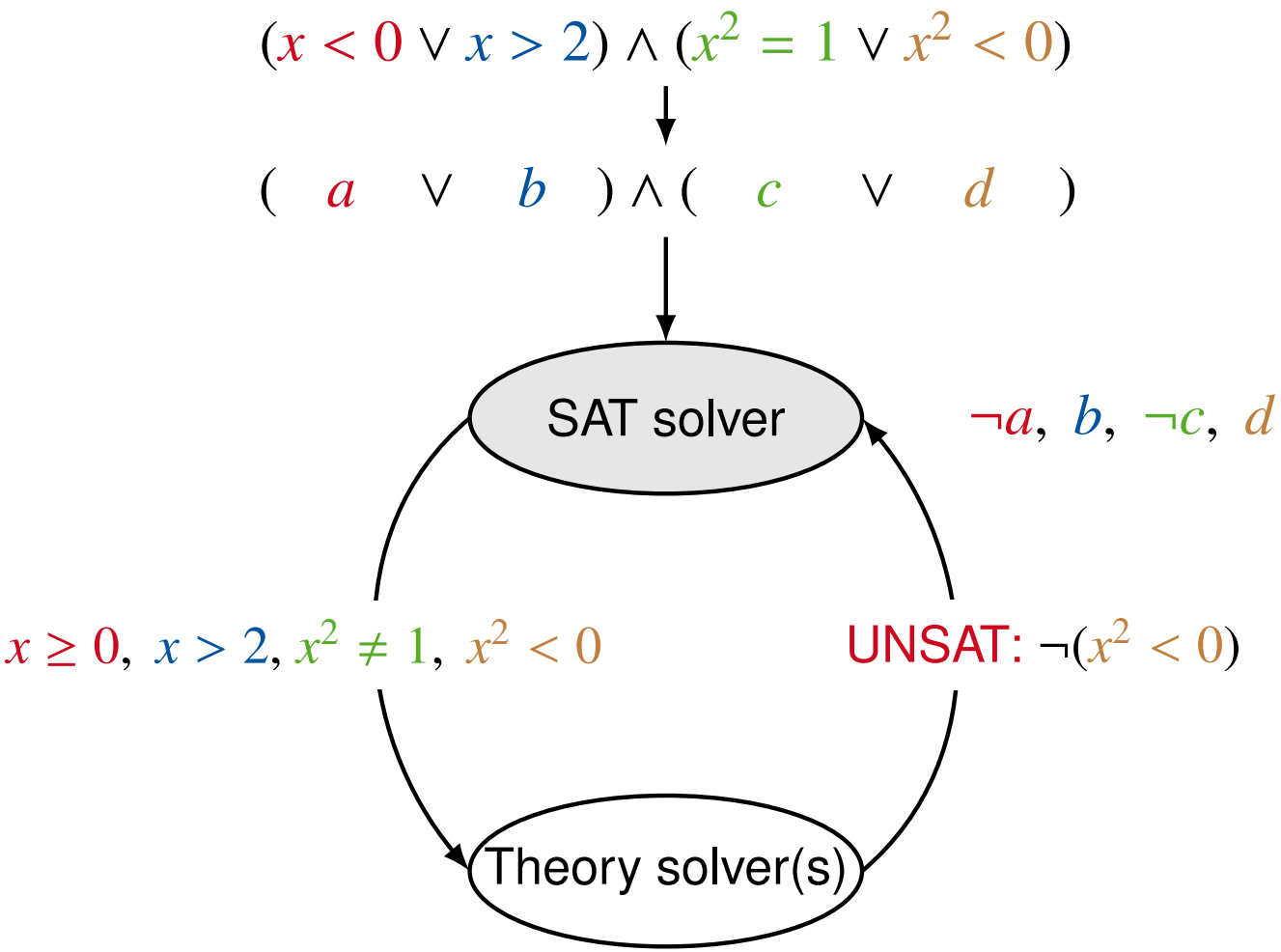
Less lazy SMT solving



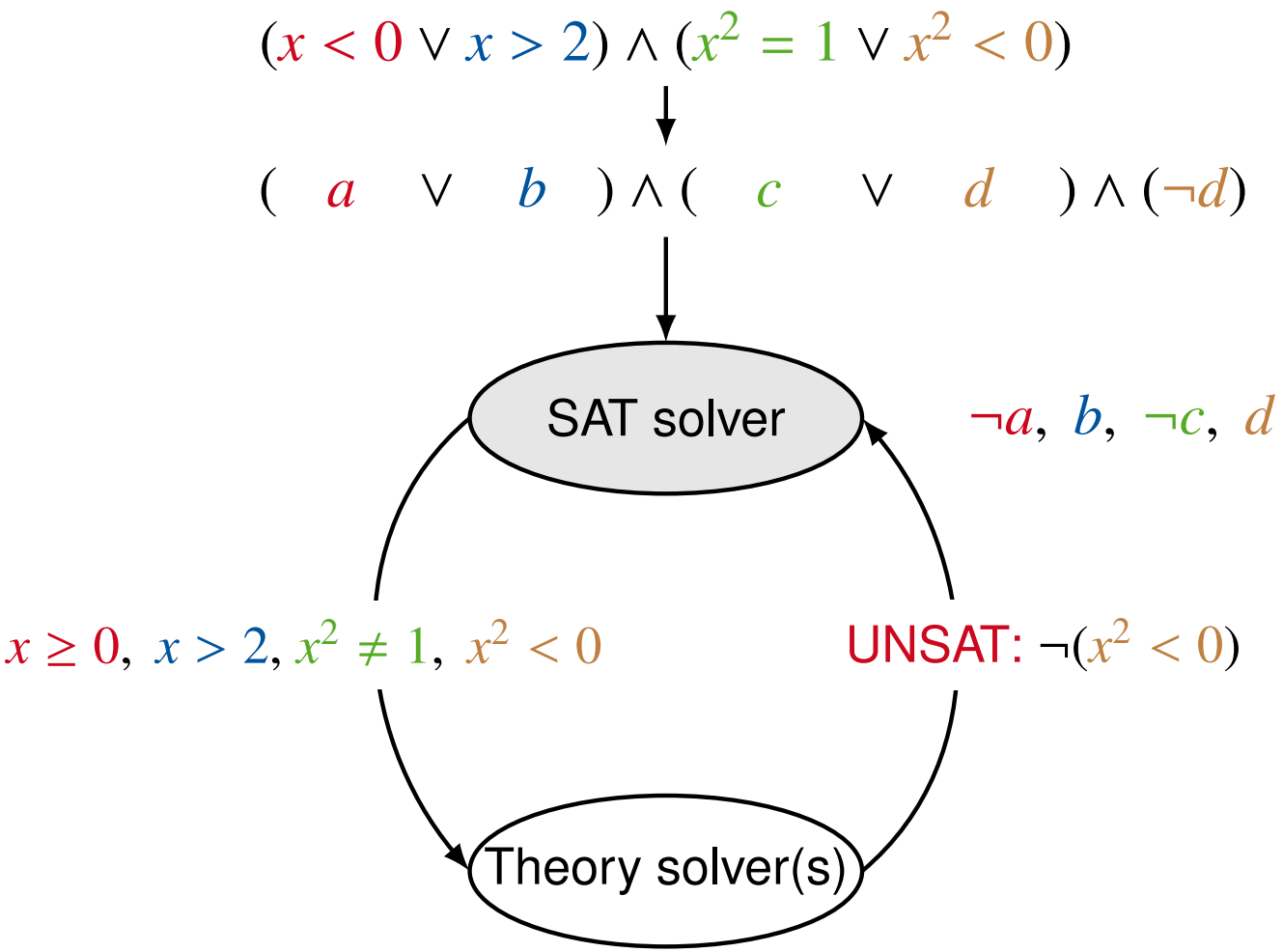
Less lazy SMT solving



Less lazy SMT solving



Less lazy SMT solving



Some theory solver candidates for arithmetic theories

Linear real arithmetic:

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

Linear integer arithmetic:

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

Non-linear real arithmetic:

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

Non-linear integer arithmetic:

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

Problem solved?

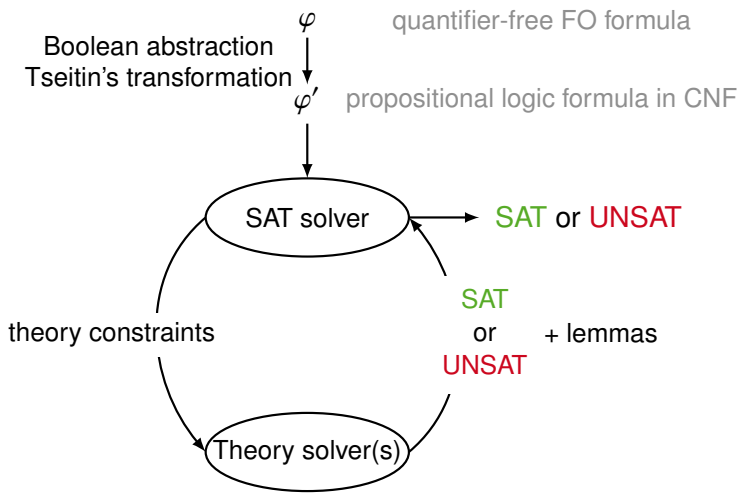
Can we use implementations of those methods out of the box?

Problem solved?

Can we use implementations of those methods out of the box?

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

- work incrementally,
- generate lemmas explaining inconsistencies, and
- be able to backtrack.

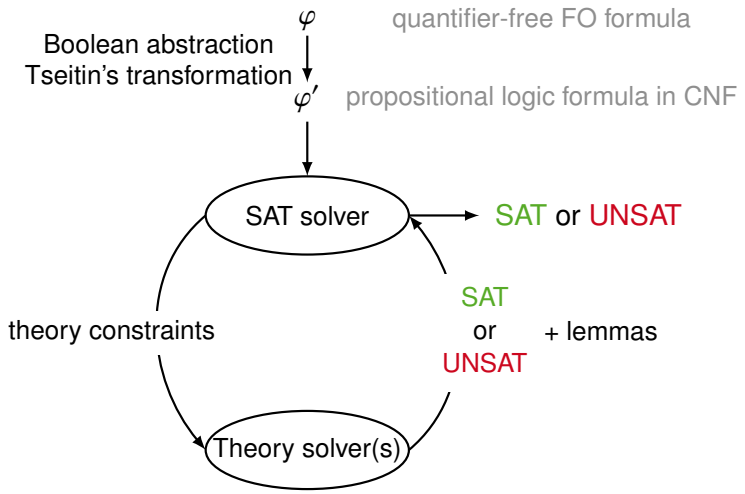


Problem solved?

Can we use implementations of those methods out of the box?

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.

Satisfiablility 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

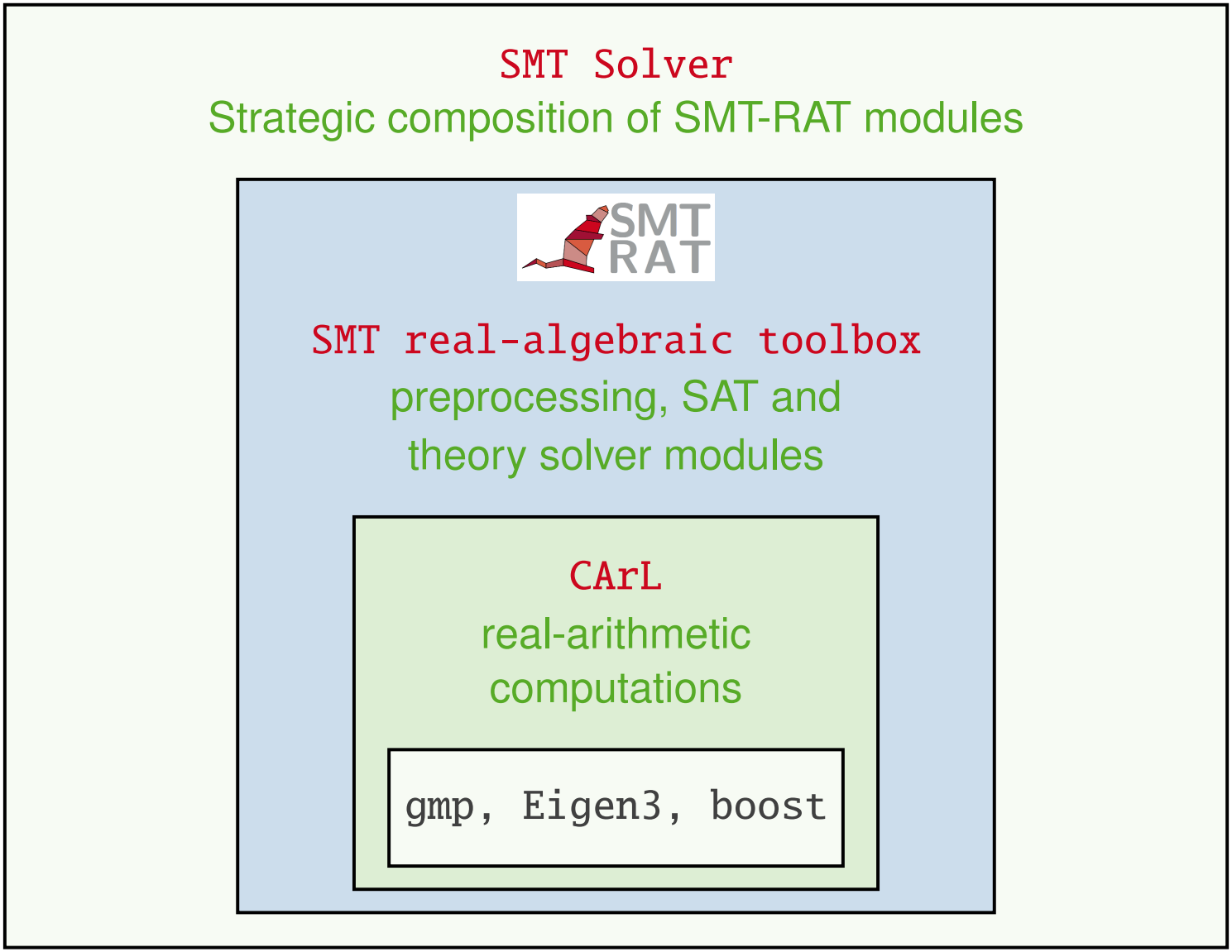
The use of advanced methods to solve practical and industrially relevant problems by computers has a long history. Whereas Symbolic Computation is concerned with the algorithmic determination of exact solutions to complex mathematical problems, more recent developments in the area of Satisfiability Checking 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 unifiying 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

Part I:
Some historical notes

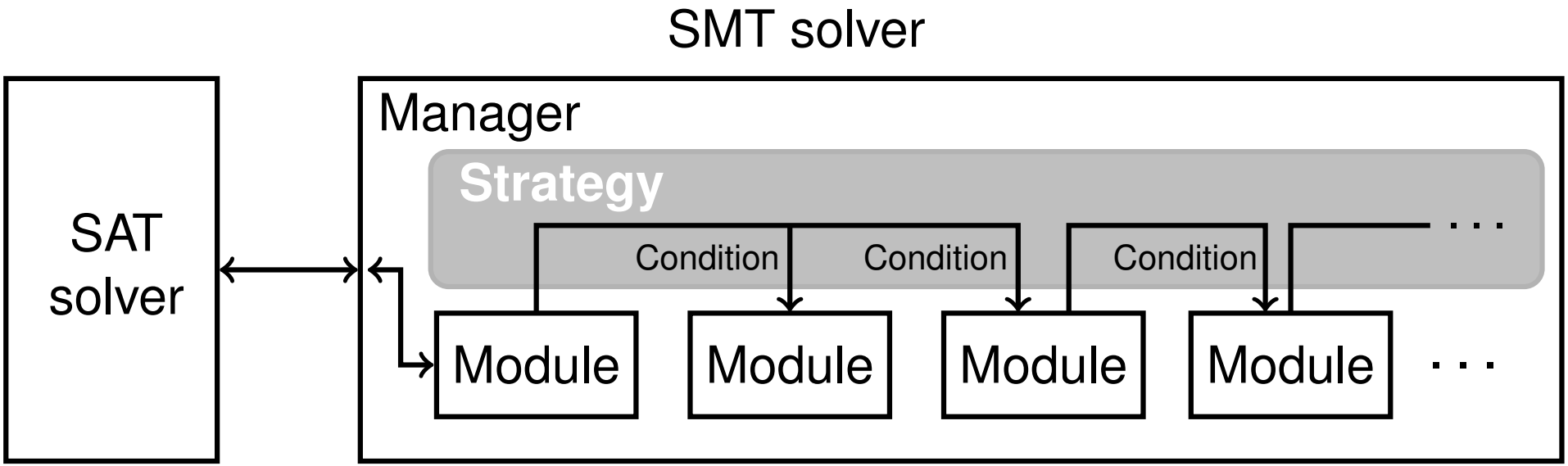
Part II:
One of the major ingredients of success:
Strategic combinations of decision procedures
...in SAT solving...
...in SMT solving...
...in theory solving.

Part III:
How satisfiability checking drives the digital world

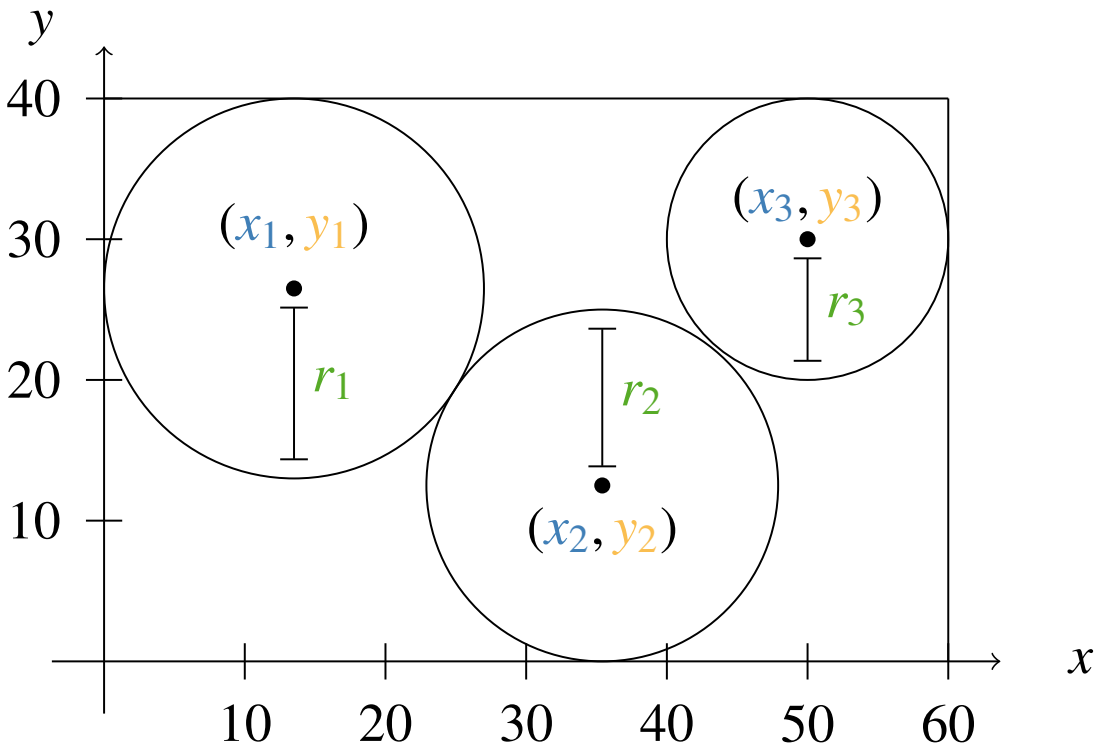
The SMT-RAT library



Strategic composition of solver modules in SMT-RAT



A pizza positioning example



$$\bigwedge_{i=1}^{3} (r_i = 13 \lor r_i = 10 \lor r_i = \tfrac{25}{2} \lor r_i = \tfrac{27}{2}) \quad \land \quad \bigwedge_{i=1}^{2} \bigwedge_{j=i+1}^{3} \neg(r_i = r_j) \quad \land$$

$$\bigwedge_{i=1}^{3} (x_i + r_i \leq 60 \land x_i - r_i \geq 0 \land y_i + r_i \leq 40 \land y_i - r_i \geq 0) \quad \land$$

$$\bigwedge_{i=1}^{2} \bigwedge_{j=i+1}^{3} (x_i - x_j)^2 + (y_i - y_j)^2 \geq (r_i + r_j)^2$$

A pizza positioning example

Z3, the currently most popular SMT solver, could not solve this problem within several minutes.

$$\bigwedge_{i=1}^{3} (r_i = 13 \lor r_i = 10 \lor r_i = \tfrac{25}{2} \lor r_i = \tfrac{27}{2}) \quad \land \quad \bigwedge_{i=1}^{2} \bigwedge_{j=i+1}^{3} \neg(r_i = r_j) \quad \land$$
$$\bigwedge_{i=1}^{3} (x_i + r_i \leq 60 \land x_i - r_i \geq 0 \land y_i + r_i \leq 40 \land y_i - r_i \geq 0) \quad \land$$
$$\bigwedge_{i=1}^{2} \bigwedge_{j=i+1}^{3} (x_i - x_j)^2 + (y_i - y_j)^2 \geq (r_i + r_j)^2$$

A pizza positioning example

Z3, the currently most popular SMT solver, could not solve this problem within several minutes.

Combining interval constraint propagation and virtual substitution using SMT-RAT finished in a seond!

$$\bigwedge_{i=1}^{3} (r_i = 13 \lor r_i = 10 \lor r_i = \tfrac{25}{2} \lor r_i = \tfrac{27}{2}) \quad \land \quad \bigwedge_{i=1}^{2} \bigwedge_{j=i+1}^{3} \neg(r_i = r_j) \quad \land$$
$$\bigwedge_{i=1}^{3} (x_i + r_i \leq 60 \land x_i - r_i \geq 0 \land y_i + r_i \leq 40 \land y_i - r_i \geq 0) \quad \land$$
$$\bigwedge_{i=1}^{2} \bigwedge_{j=i+1}^{3} (x_i - x_j)^2 + (y_i - y_j)^2 \geq (r_i + r_j)^2$$

Some other key issues

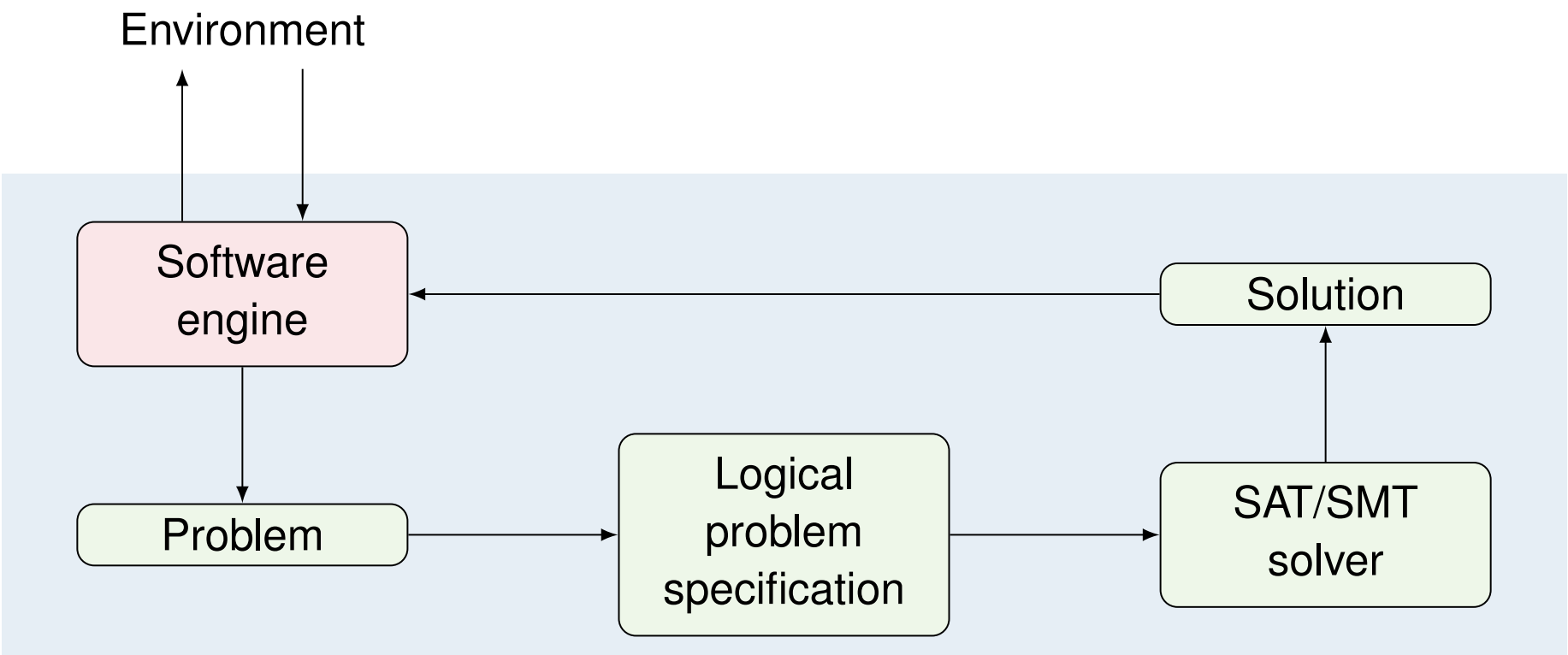
- **Heuristics!!!**
- Lemma generation
- Efficient data structures
- . . .

Part I:
Some historical notes

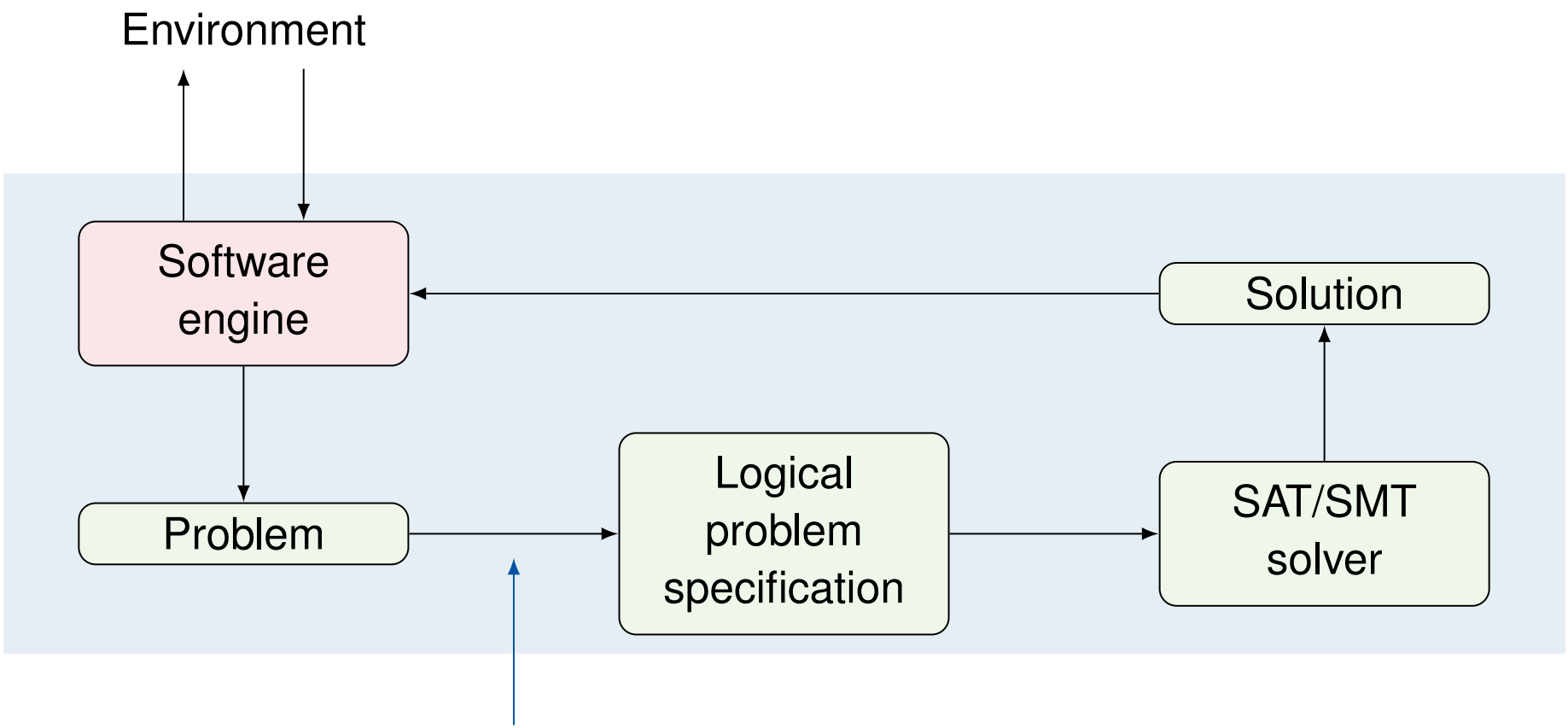
Part II:
One of the major ingredients of success:
Strategic combinations of decision procedures
...in SAT solving...
...in SMT solving...
...in theory solving.

Part III:
How satisfiability checking drives the digital world

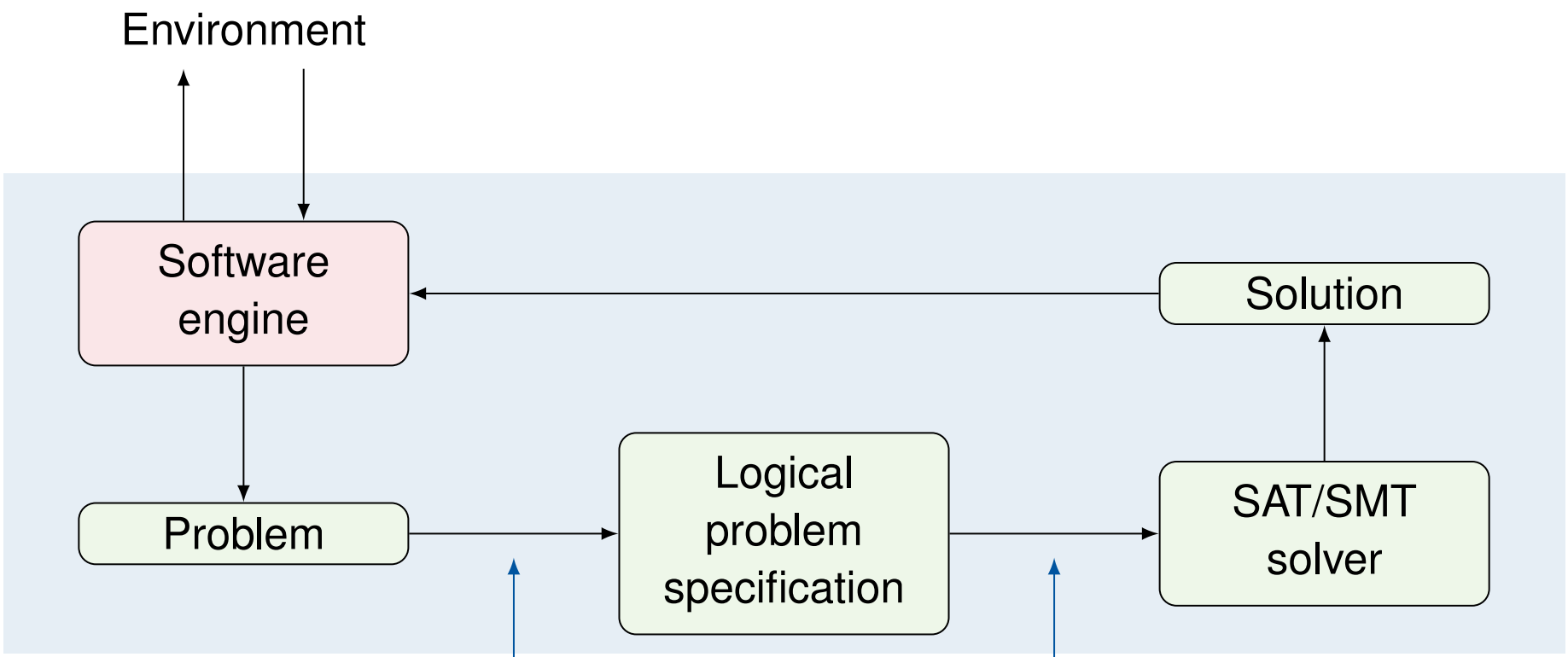
Embedding SAT/SMT solvers



Embedding SAT/SMT solvers



Embedding SAT/SMT solvers

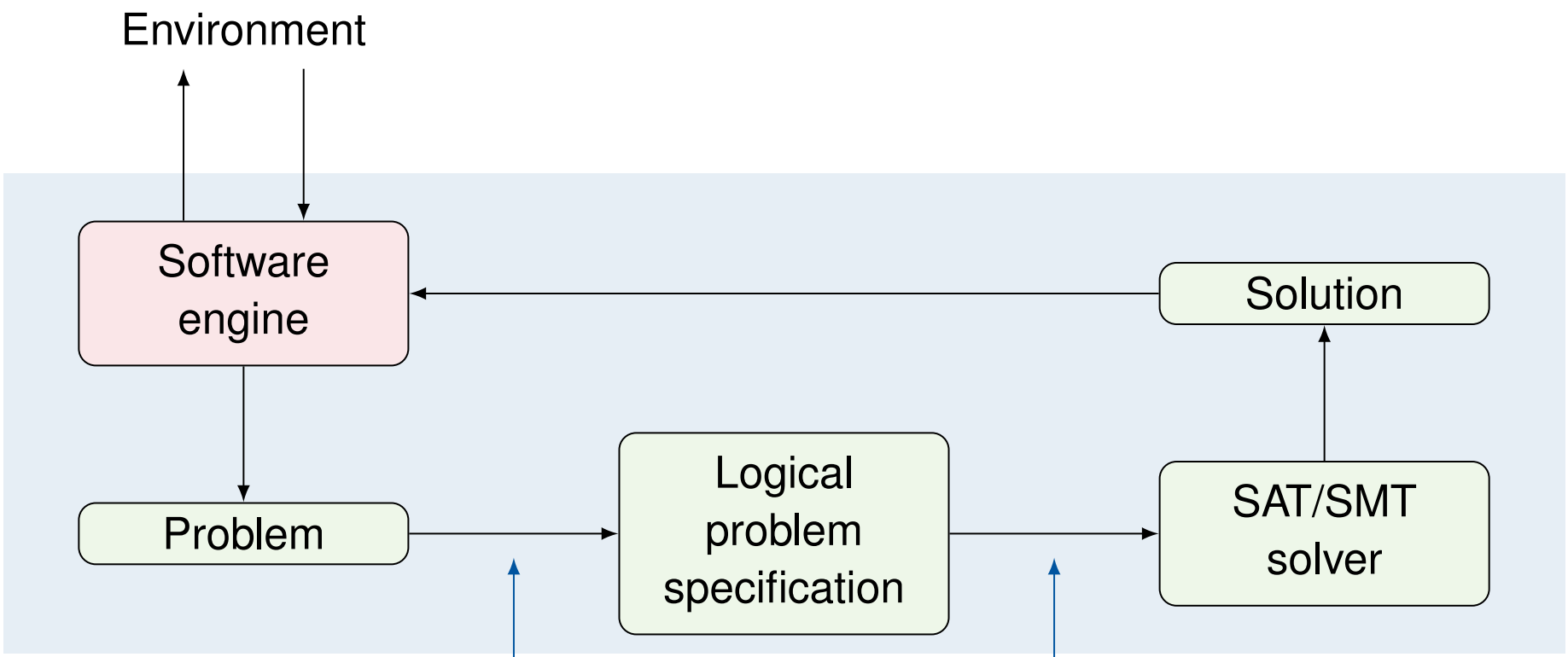


Encoding: SAT/SMT-LIB standard

elaborate encoding is extremely important!

standard input syntax → free solver choice

Embedding SAT/SMT solvers



Encoding: SAT/SMT-LIB standard

elaborate encoding is extremely important!

standard input syntax → free solver choice

In the following: applications of SMT solvers

Bounded model checking for C/C++

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/`

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/`

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.

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

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.

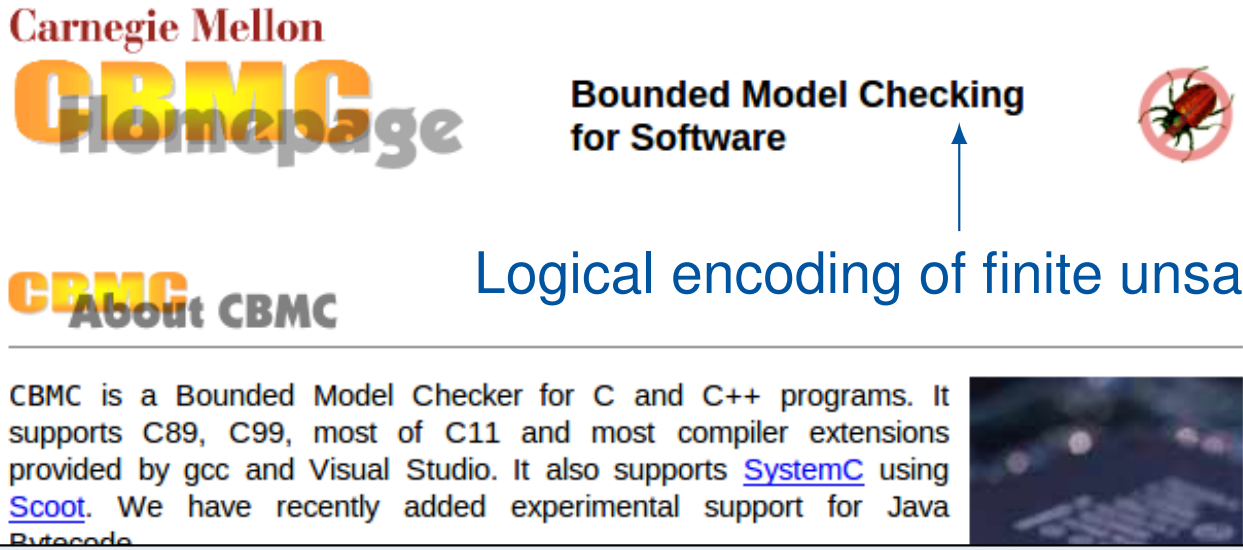
While CBMC is aimed for embedded software, it also supports dynamic memory allocation using `malloc` and `new`. For questions about CBMC, contact Daniel Kroening.

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.

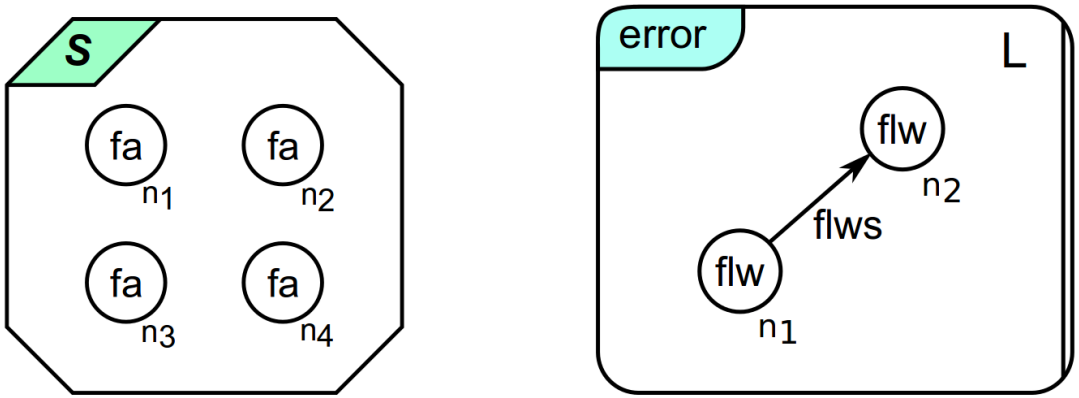
Source: D. Kroening. **CBMC home page.** `http://www.cprover.org/cbmc/`

Bounded model checking for C/C++



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

BMC for graph transformation systems



(a) Initial graph S (b) Forbidden pattern

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

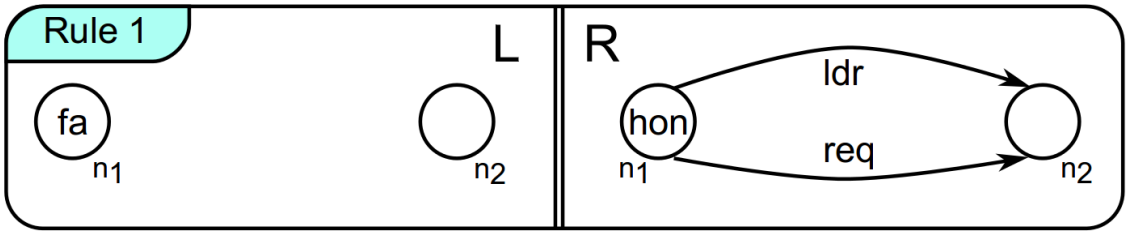


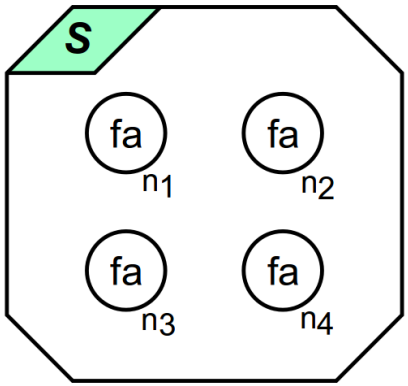
Fig. 2. Rule 1 of the car platooning GTS [1]

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

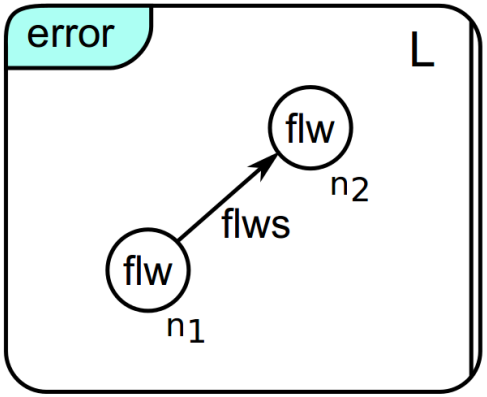
Bounded Model Checking of Graph Transformation Systems via SMT Solving.

In Proc. FMOODS/FORTE'13.

BMC for graph transformation systems



(a) Initial graph S



(b) Forbidden pattern

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

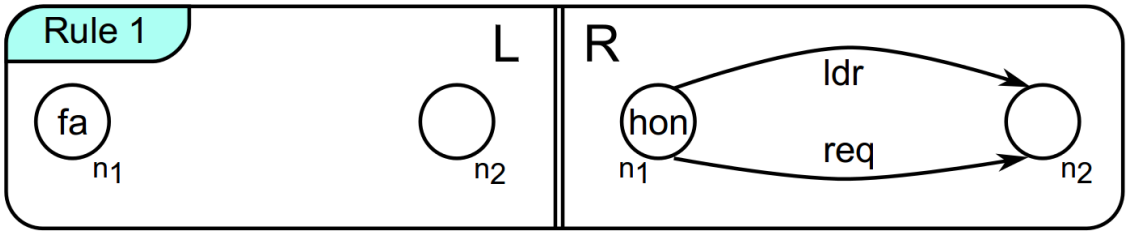


Fig. 2. Rule 1 of the car platooning GTS [1]

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.

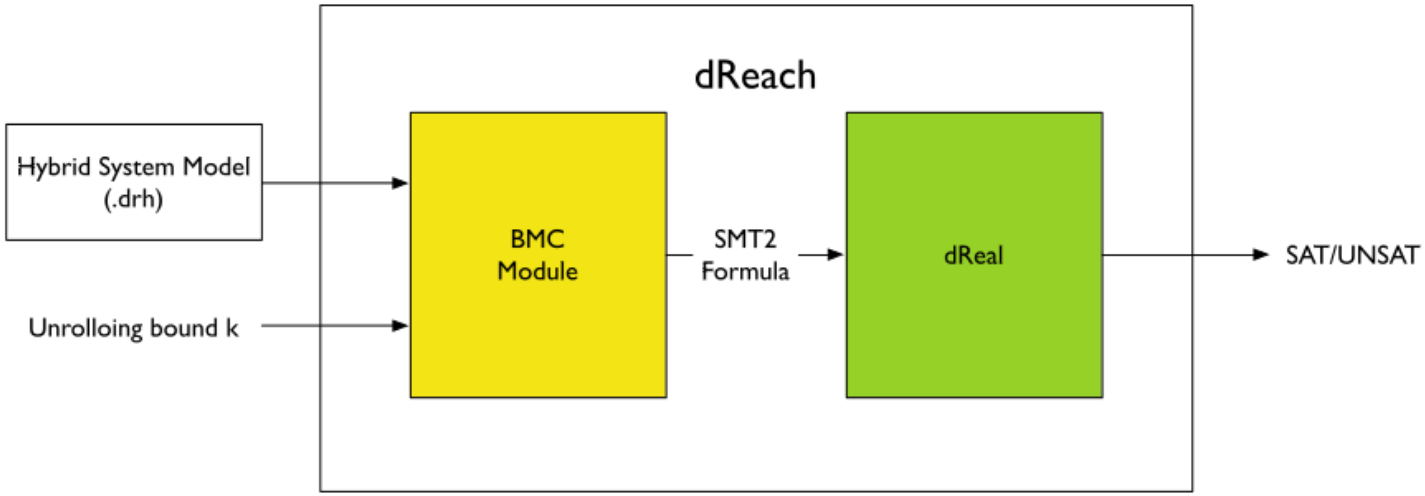
In Proc. FMOODS/FORTE'13.

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.



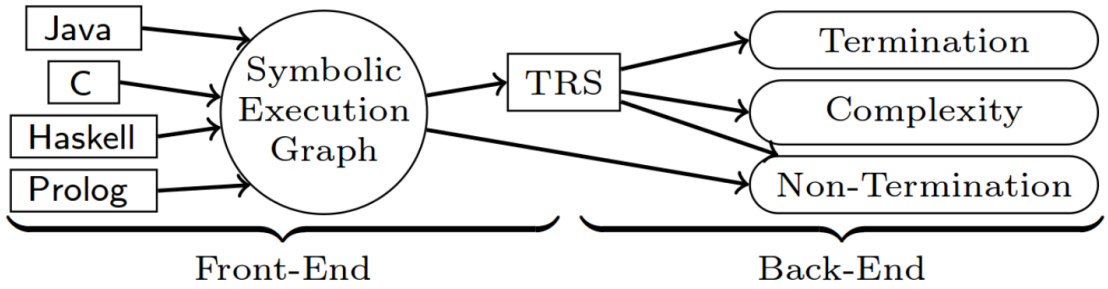
Source: D. Bryce, J. Sun, P. Zuliani, Q. Wang, S. Gao, F. Shmarov, S. Kong, W. Chen, Z. Tavares.

dReach home page. `http://dreal.github.io/dReach/`

Termination analysis for programs

AProVE

Automated Program Verification Environment



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

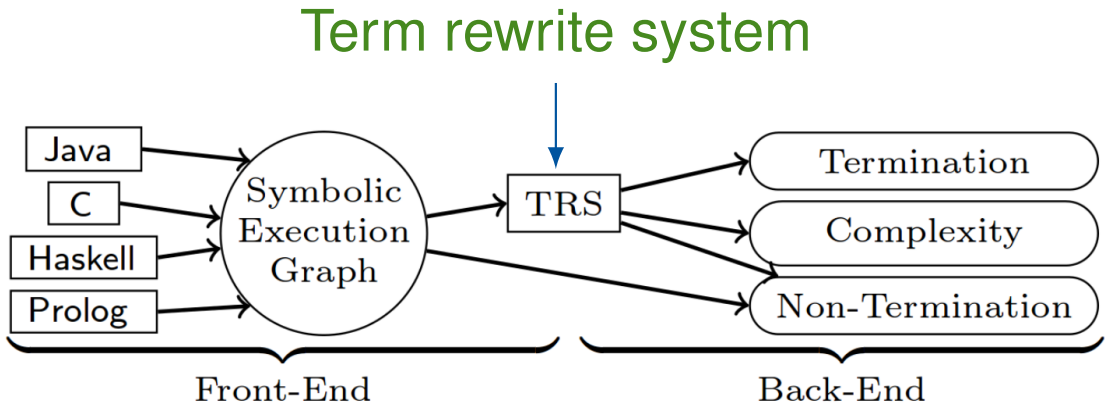
AProVE: Termination and memory safety of C programs (competition contribution).

In Proc. TACAS'15.

Termination analysis for programs

AProVE

Automated Program Verification Environment



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

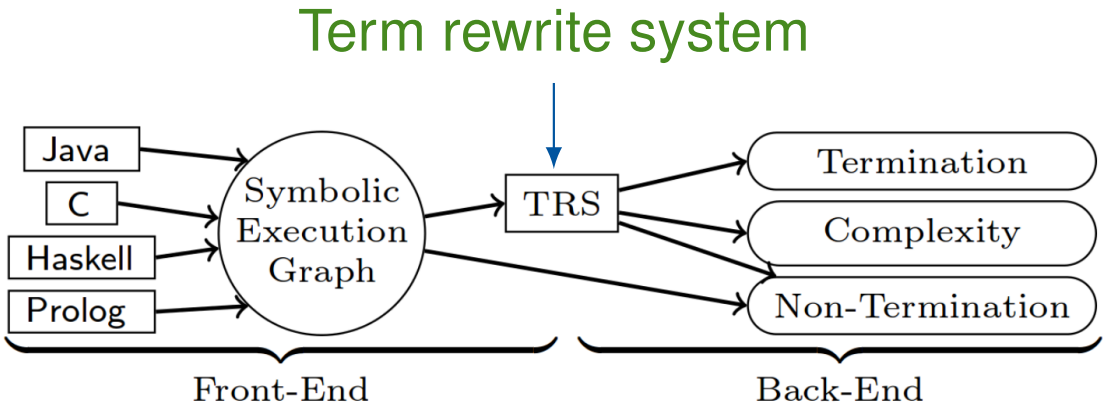
AProVE: Termination and memory safety of C programs (competition contribution).

In Proc. TACAS'15.

Termination analysis for programs

AProVE

Automated Program Verification Environment



Term rewrite system

↓

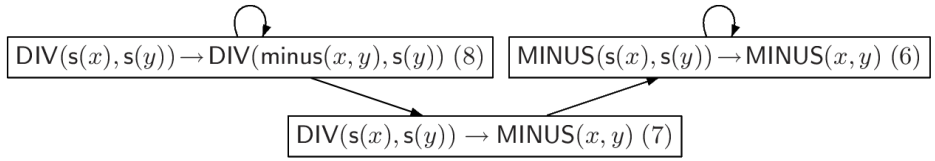
Dependency pairs

↓

Chains

$$\begin{aligned}
\text{minus}(x,0) &\to x && (1)\\
\text{minus}(0,\text{s}(y)) &\to 0 && (2)\\
\text{minus}(\text{s}(x),\text{s}(y)) &\to \text{minus}(x,y) && (3)\\
\text{div}(0,\text{s}(y)) &\to 0 && (4)\\
\text{div}(\text{s}(x),\text{s}(y)) &\to \text{s}(\text{div}(\text{minus}(x,y),\text{s}(y))) && (5)
\end{aligned}$$

$$\begin{aligned}
\text{MINUS}(\text{s}(x),\text{s}(y)) &\to \text{MINUS}(x,y) && (6)\\
\text{DIV}(\text{s}(x),\text{s}(y)) &\to \text{MINUS}(x,y) && (7)\\
\text{DIV}(\text{s}(x),\text{s}(y)) &\to \text{DIV}(\text{minus}(x,y),\text{s}(y)) && (8)
\end{aligned}$$



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

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

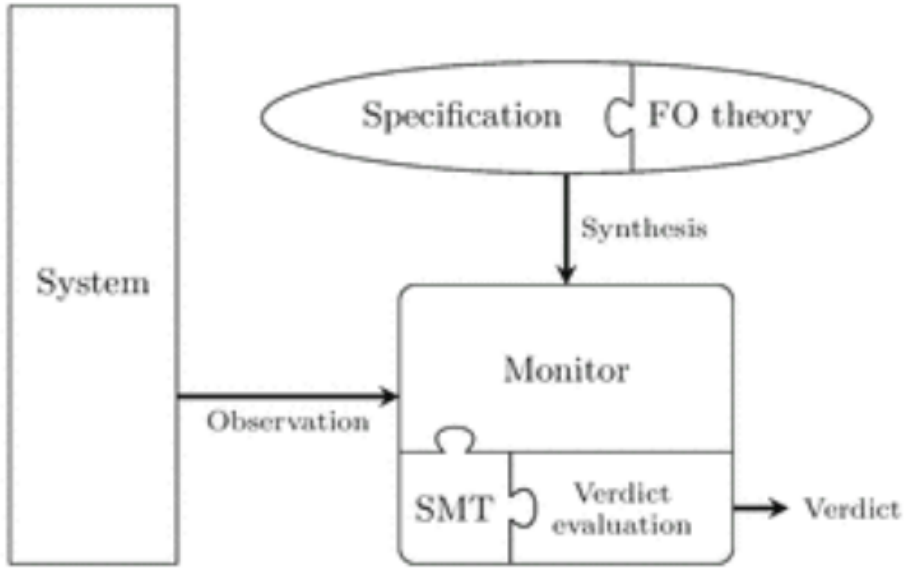


Fig. 1 Schematic overview of the monitoring approach

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

Monitoring modulo theories.

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

Planning

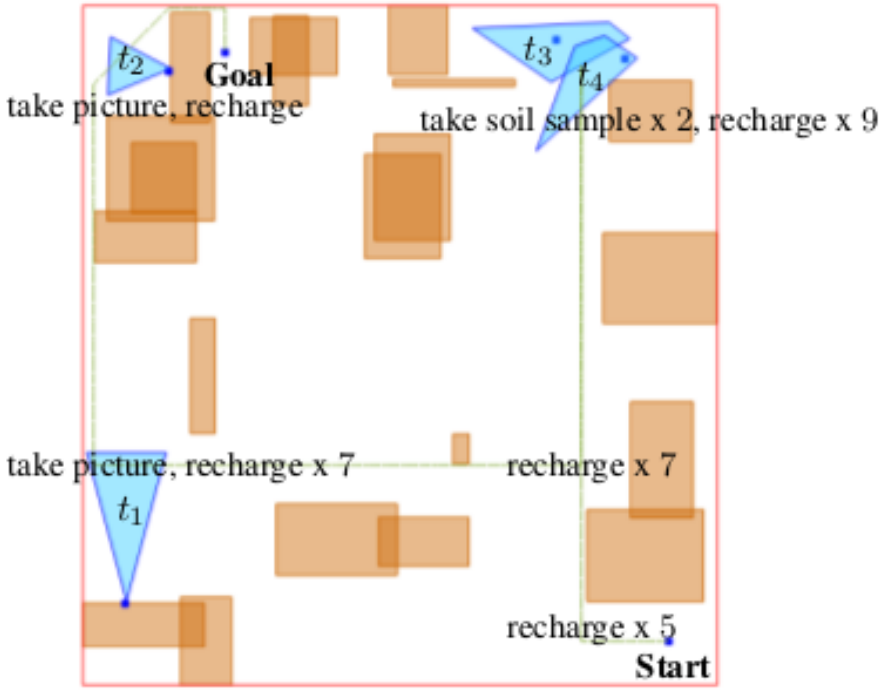


Figure 1: A GEOMETRIC ROVERS example instance, showing the starting and goal locations of the rover, areas where tasks can be performed (blue) and obstacles (orange) and a plan solving the task (green). The red box indicates the bounds of the environment.

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

Numeric planning with disjunctive global constraints via SMT.

In Proc. of ICASP'16.

Scheduling

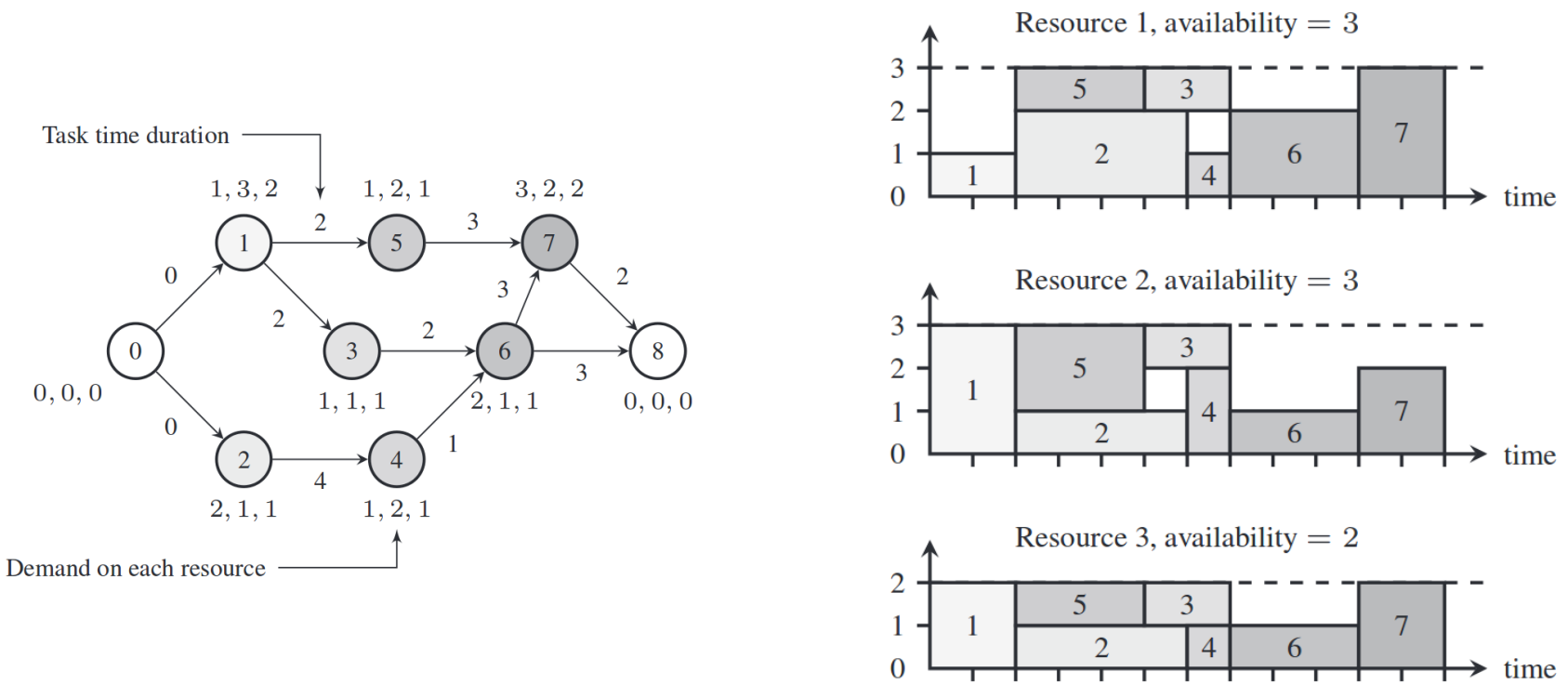


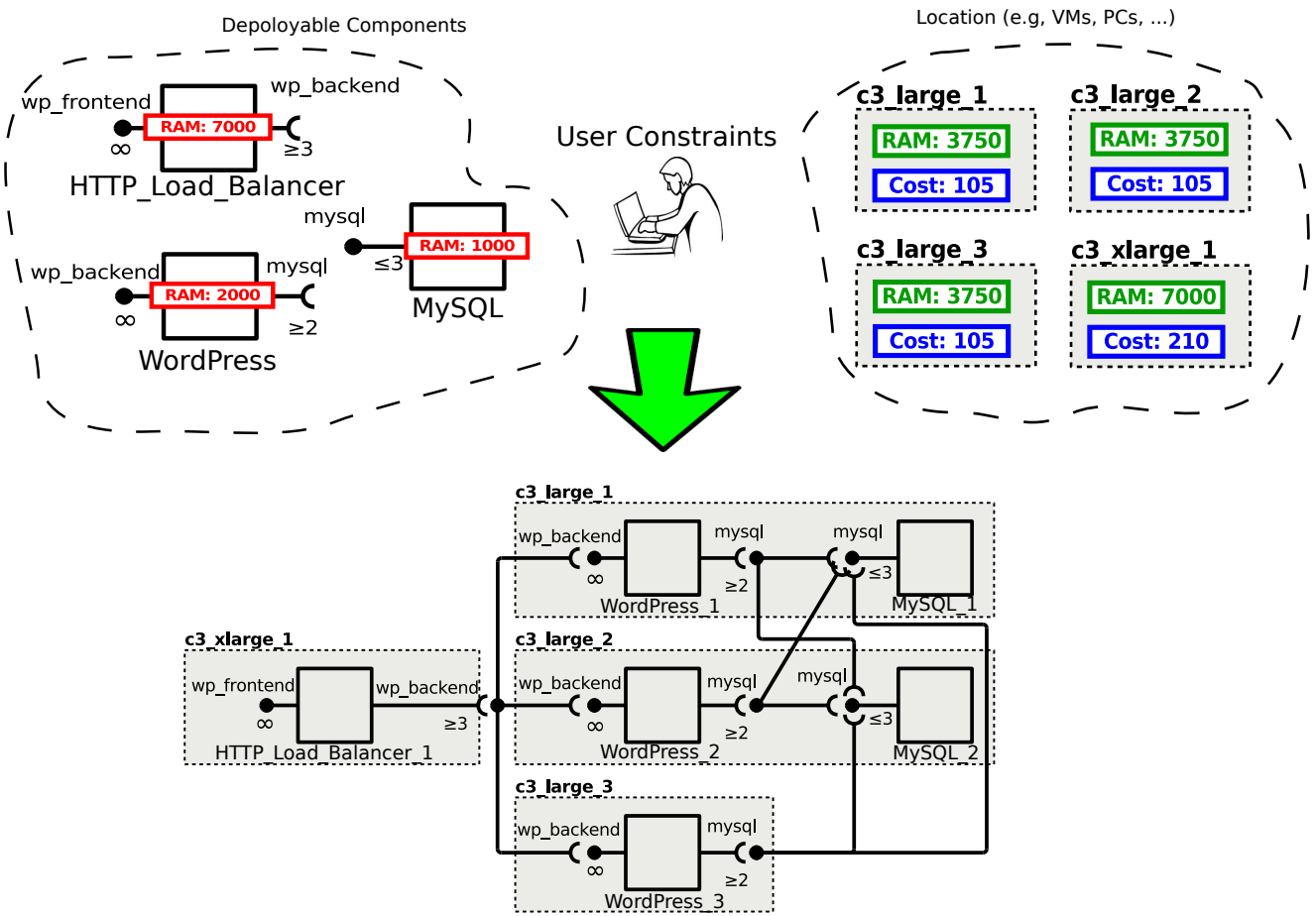
Figure 1: An example of RCPSP (Liess and Michelon 2008)

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

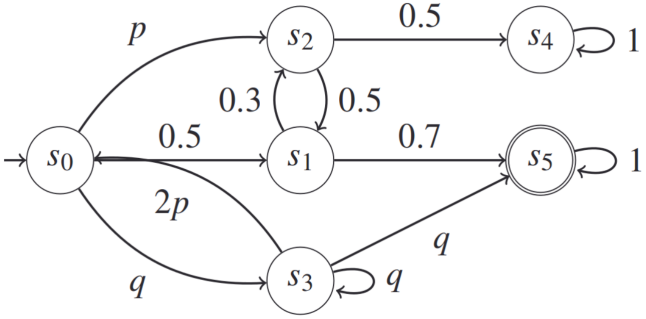


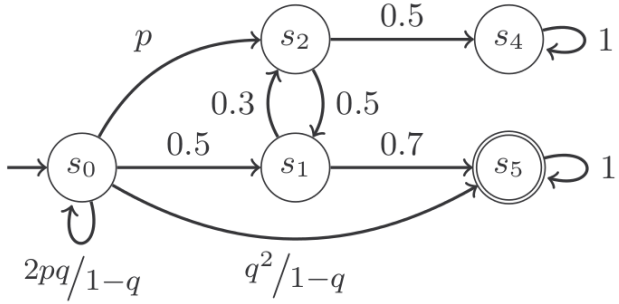
Source: E. Ábrahám, F. Corzilius, E. Broch Johnsen, G. Kremer, J. Mauro.

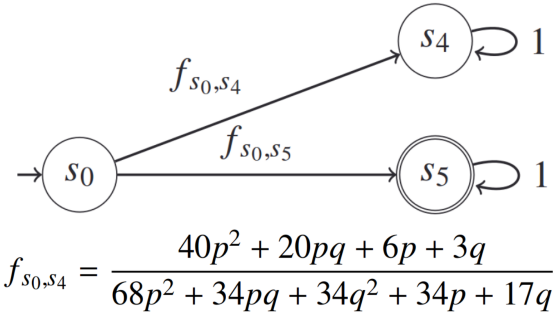
Zephyrus2: On the fly deployment optimization using SMT and CP technologies.

Submitted to SETTA'16.

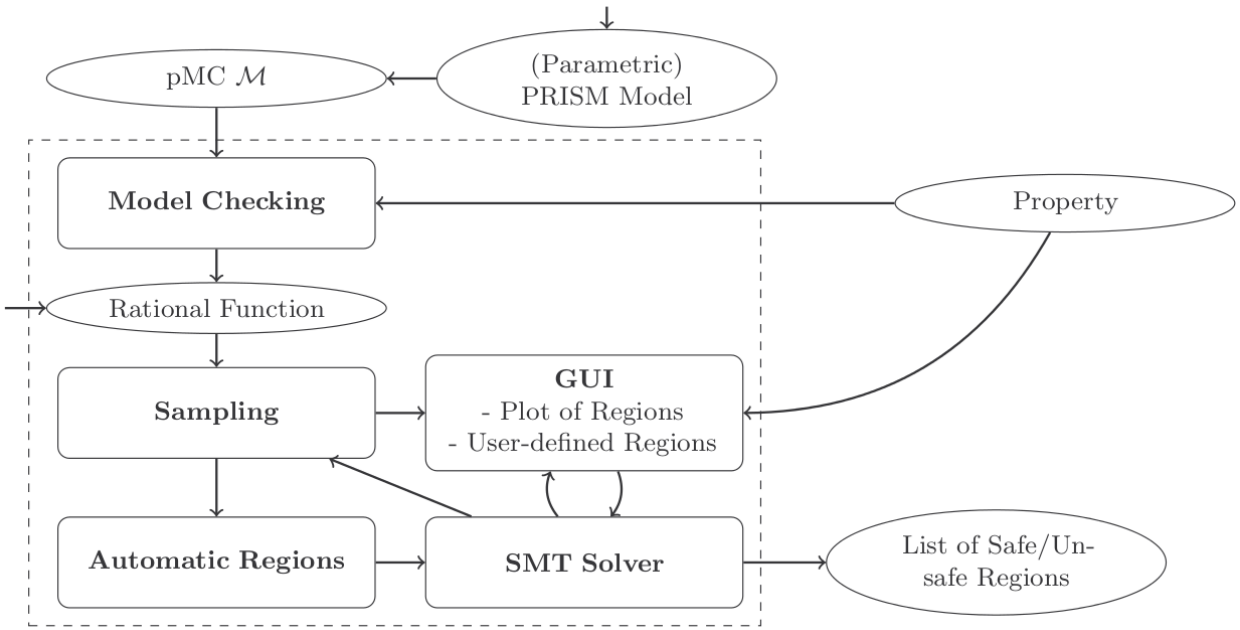
Parameter synthesis for probabilistic systems

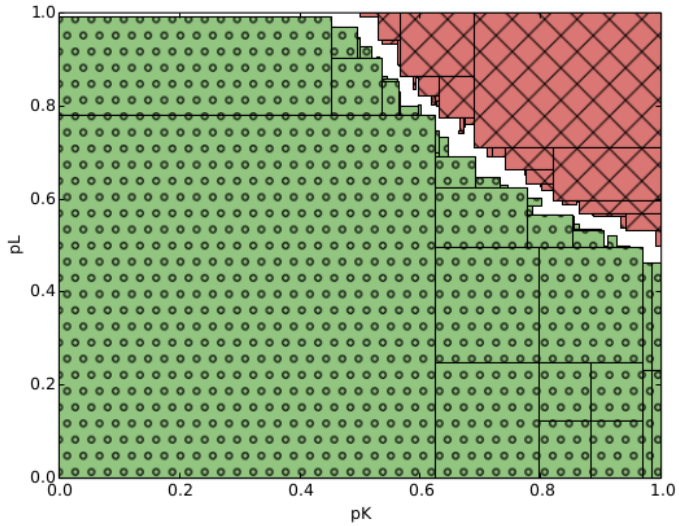






$$f_{s_0,s_4} = \frac{40p^2 + 20pq + 6p + 3q}{68p^2 + 34pq + 34q^2 + 34p + 17q}$$





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

PROPhESY: A probabilistic parameter synthesis tool.

In Proc. of CAV'15.

Concluding notes

- Satisfiability checking combines methods in innovative ways, putting big weight on (practical) efficiency.
- SAT and SMT solvers are impressively powerful general tools.
- They have a wide (and steeply increasing) range of application areas.
- They have a big impact on making our hardware and software systems more efficient and more safe.