# SAT and SMT Solving in a Nutshell

Erika Ábrahám

RWTH Aachen University, Germany
LuFG Theory of Hybrid Systems

February 27, 2020

# What is this talk about?

## Satisfiability problem

The satisfiability problem is the problem of deciding whether a logical formula is satisfiable.

We focus on the automated solution of the satisfiability problem for first-order logic over arithmetic theories, especially using SAT and SMT solving.

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

1940 Computer architecture development

1960

1970

1980

2000

2010

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

1940 Computer architecture development

CAS

1960 Computer algebra systems

1970

CAD

1980

Partial CAD

2000 Virtual substitution

2010

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

1940 — Computer architecture development

| | CAS | SAT (propositional logic) |
|---|---|---|
| | | Enumeration |
| 1960 | Computer algebra systems | DP (resolution) [Davis, Putnam'60] |
| | | DPLL (propagation) [Davis,Putnam,Logemann,Loveland'62] |
| 1970 | | NP-completeness [Cook'71] |
| | CAD | |
| 1980 | | Conflict-directed backjumping |
| | Partial CAD | |
| | Virtual substitution | CDCL [GRASP'97] |
| 2000 | | Watched literals [zChaff'04] |
| | | Clause learning/forgetting |
| | | Variable ordering heuristics |
| 2010 | | Restarts |

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

1940 Computer architecture development

1960

1970

1980

2000

2010

## CAS

Computer algebra systems

CAD

Partial CAD

Virtual substitution

## SAT
(propositional logic)

Enumeration

DP (resolution) [Davis, Putnam'60]

DPLL (propagation) [Davis,Putnam,Logemann,Loveland'62]

NP-completeness [Cook'71]

Conflict-directed backjumping

CDCL [GRASP'97]

Watched literals [zChaff'04]

Clause learning/forgetting

Variable ordering heuristics

Restarts

## SMT
(SAT modulo theories)

Decision procedures for combined theories [Shostak'79] [Nelson, Oppen'79]

DPLL(T)

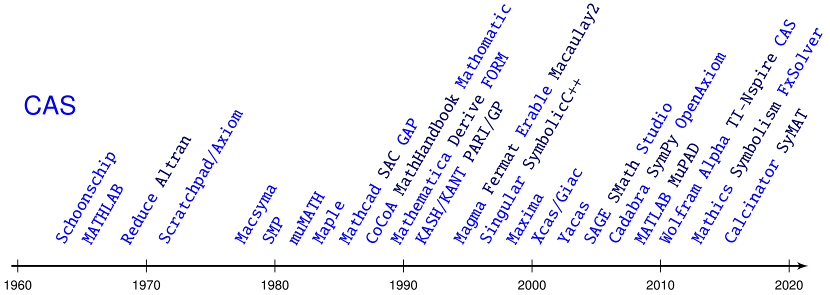
Equalities and uninterpreted functions

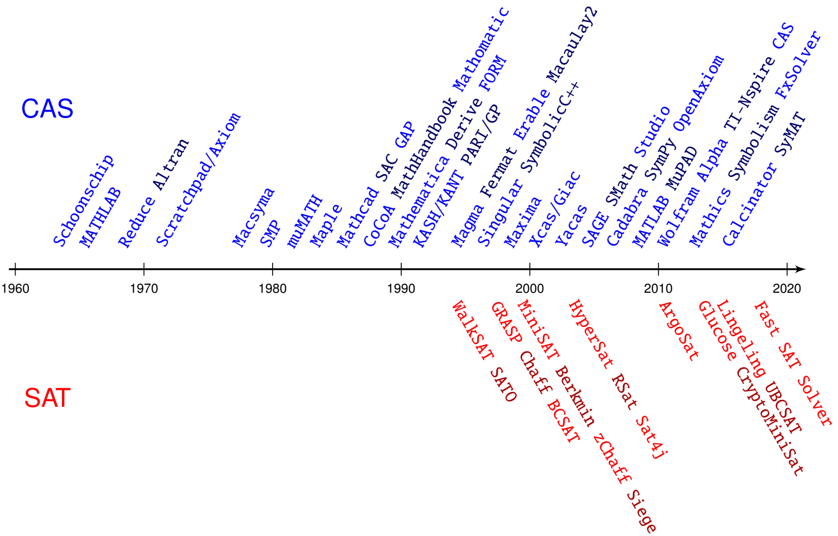
Bit-vectors

Array theory

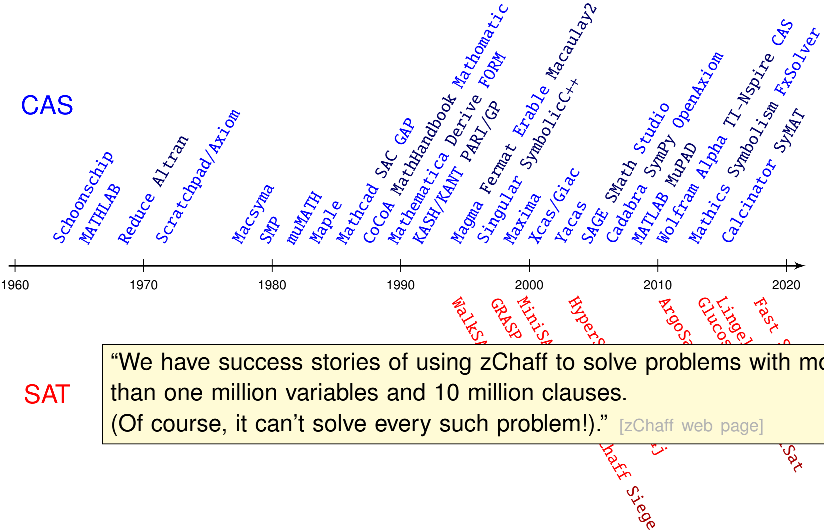
Arithmetic

# Tool development

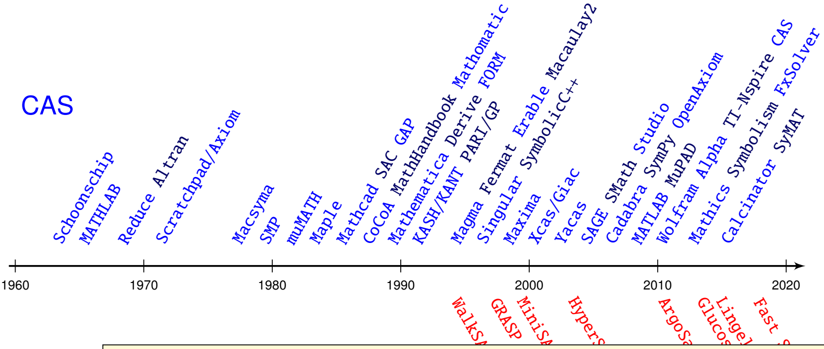



CAS
SAT
Schoonschip
MATHLAB
Reduce Altran
Scratchpad/Axiom
Macsyma
SMP
muMATH
Maple
Mathcad SAC GAP
CoCoA MathHandbook Mathomatic
Mathematica Derive FORM
KASH/KANT PARI/GP
Magma Fermat Erable Macaulay2
Singular SymbolicC++
Maxima
Xcas/Giac
Yacas
SAGE SMath Studio
Cadabra SymPy OpenAxiom
MATLAB MuPAD
Wolfram Alpha TI-Nspire CAS
Mathics Symbolism FxSolver
Calcinator SyMAT
1960
1970
1980
1990
2000
2010
2020
WalkSAT SATO
GRASP Chaff BCSAT
MiniSAT Berkmin zChaff Siege
HyperSat RSat Sat4j
ArgoSat
Glucose CryptoMiniSat
Lingeling UBCSAT
Fast SAT Solver

# Tool development




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

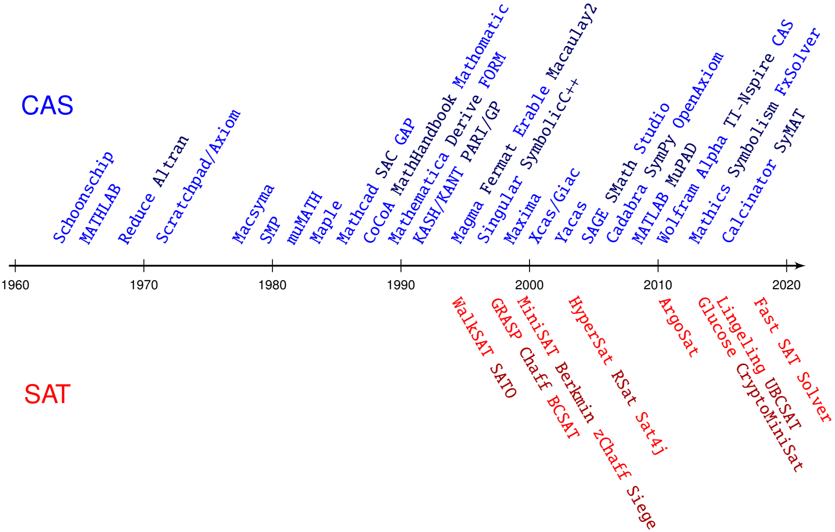



SAT

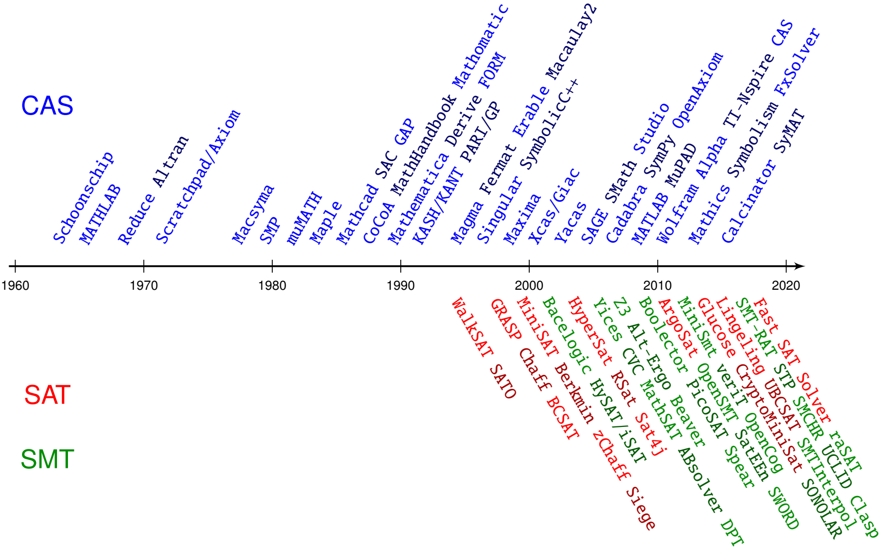
"We have success stories of using zChaff to solve problems with more than one million variables and 10 million clauses.
(Of course, it can't solve every such problem!)." [zChaff web page]

" The efficiency of our programs allowed us to solve over one hundred open quasigroup problems in design theory." [SATO web page]

# Tool development



# Tool development



# 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.

  2014 SAT Competition: $3$ categories, $79$ participants with $137$ solvers.

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

# 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

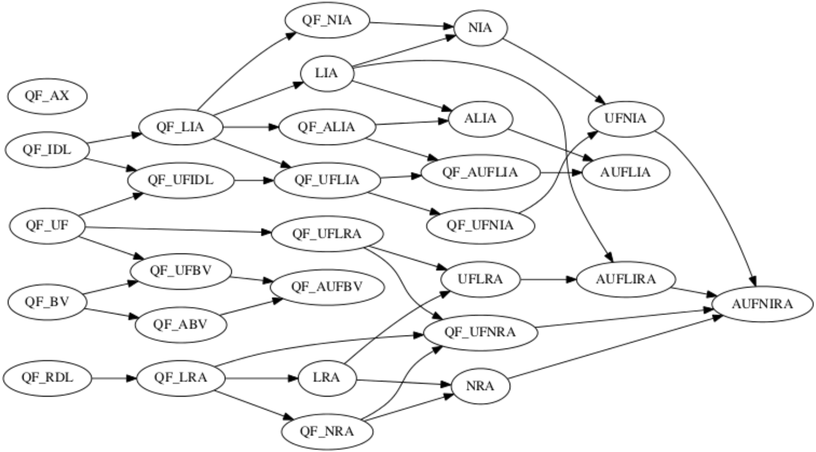
- Propositional logic is sometimes too weak for modelling.
- We need more expressive logics and decision procedures for them.
- Logics:
  quantifier-free fragments of first-order logic over various theories.
- Our focus: SAT-modulo-theories (SMT) solving.
- SMT-LIB as standard input language since 2004.
- Competitions since 2005.
- SMT-COMP 2014 competition:
  - $32$ logical categories, $20$ solvers.
  - Linear real arithmetic (since 2005): $6$ solvers.
  - Non-linear real arithmetic (since 2010): $4$ solvers.
  - $67426$ benchmark instances.

# 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 2014 competition:
  - $32$ logical categories, $20$ solvers.
  - Linear real arithmetic (since 2005): $6$ solvers.
  - Non-linear real arithmetic (since 2010): $4$ solvers.
  - $67426$ benchmark instances.

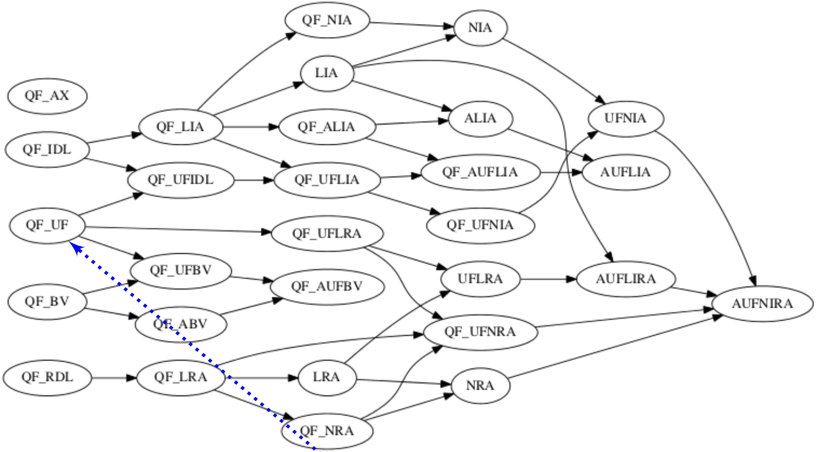
SMT applications: verification (model checking, static analysis, termination analysis); test case generation; controller synthesis; predicate abstraction; equivalence checking; scheduling; planning; product design automation and optimisation, . . .

# SMT-LIB theories




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

# SMT-LIB theories

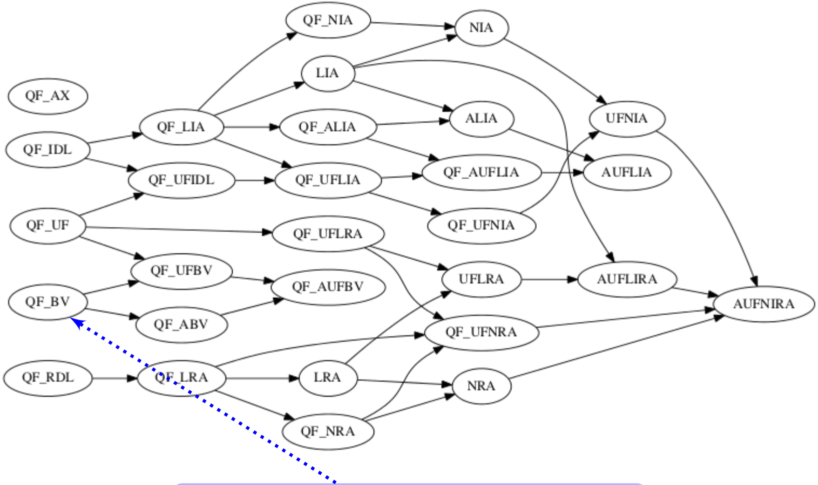



Quantifier-free equality logic with uninterpreted functions

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

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

# SMT-LIB theories

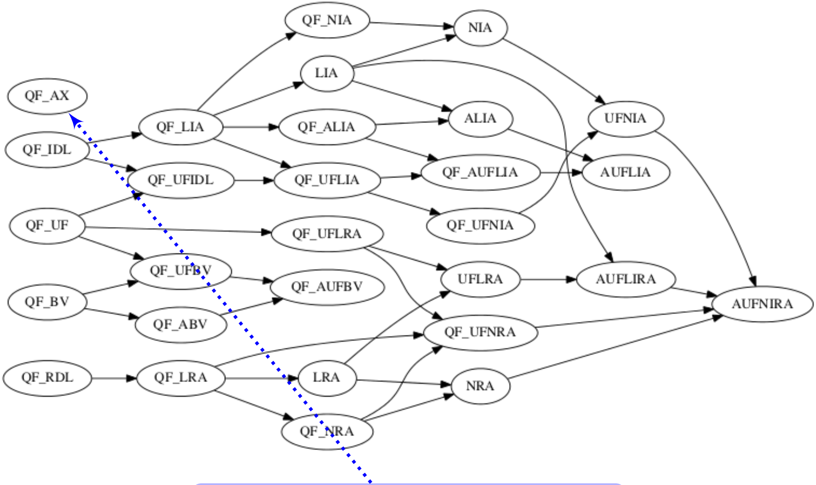



Quantifier-free bit-vector arithmetic

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

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

# SMT-LIB theories

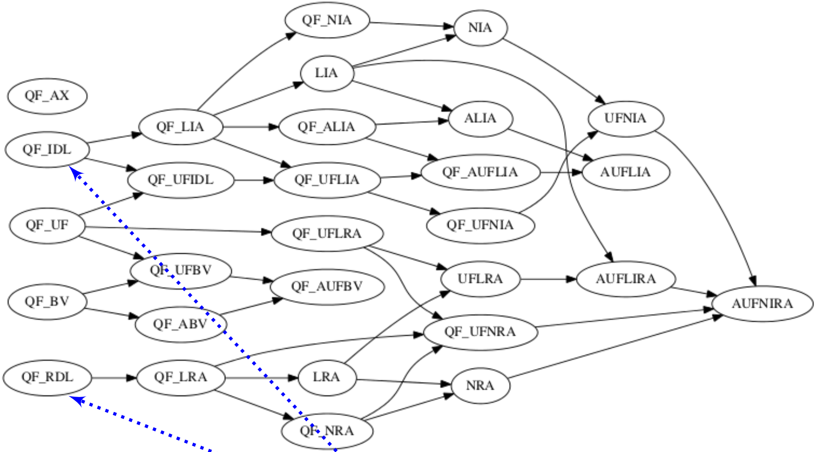



Quantifier-free array theory

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

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

# SMT-LIB theories

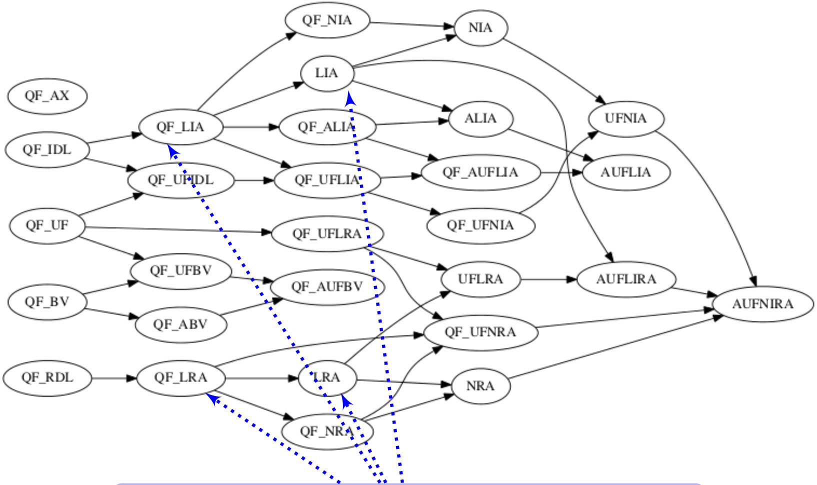



Quantifier-free integer/rational difference logic

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

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

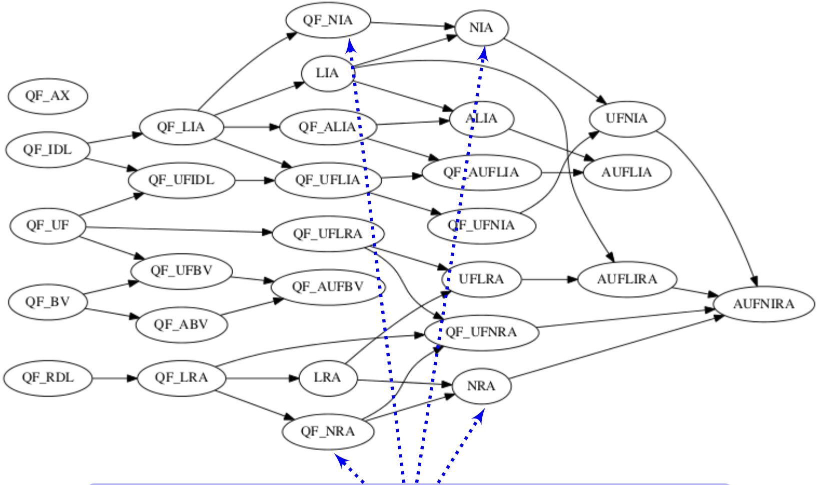
# SMT-LIB theories




(Quantifier-free) real/integer linear arithmetic

$$3x + 7y = 8$$

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

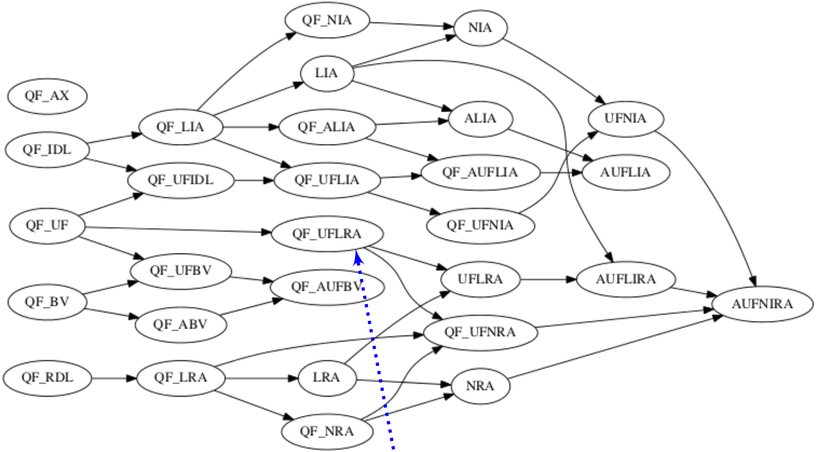



(Quantifier-free) real/integer non-linear arithmetic

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

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`

# Eager vs. lazy SMT solving

- We focus on lazy SMT solving.
- Alternative eager approach: transform problems into propositional logic and use SAT solving for satisfiability checking.
- Condition: Logic is not more expressive than propositional logic.

# (Full/less) lazy SMT solving

# (Full/less) lazy SMT solving

$\varphi$ quantifier-free FO formula

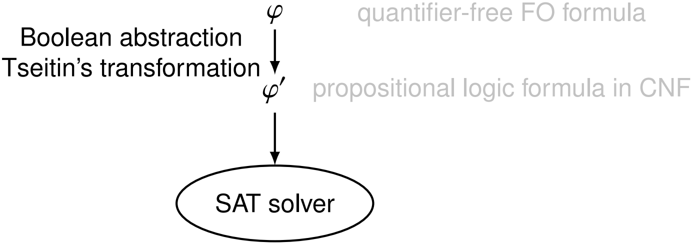
# (Full/less) lazy SMT solving

$\varphi$ quantifier-free FO formula

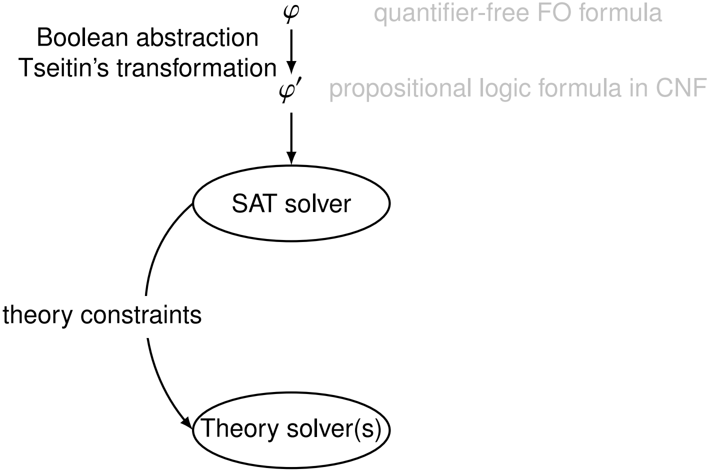
Boolean abstraction
Tseitin's transformation
$\downarrow$

$\varphi'$ propositional logic formula in CNF

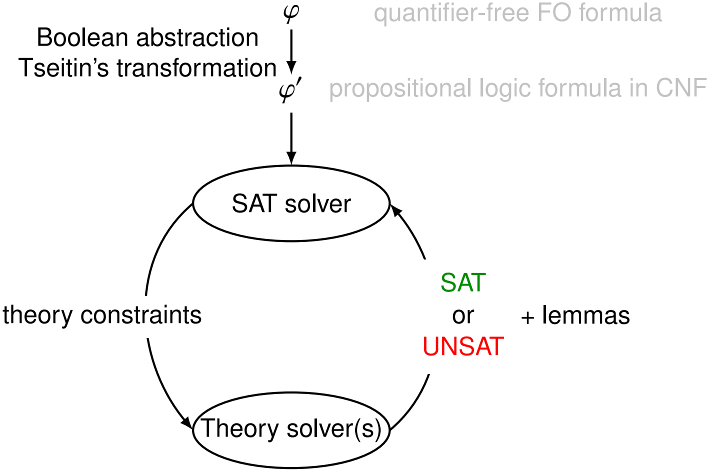
# (Full/less) lazy SMT solving



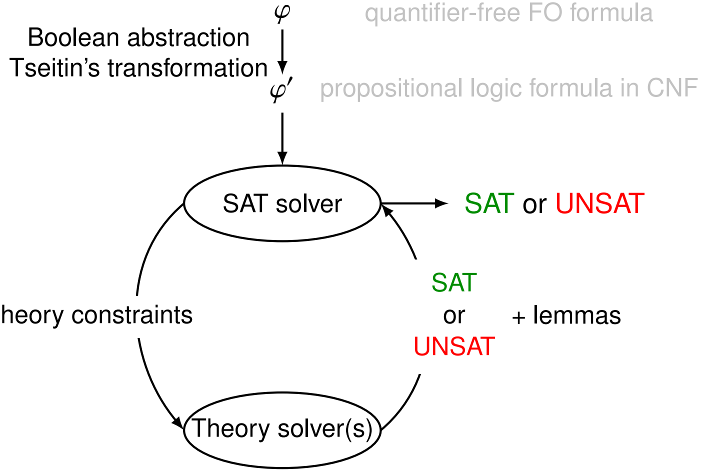
# (Full/less) lazy SMT solving



# (Full/less) lazy SMT solving



# (Full/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)

SMT solvers: `Alt-Ergo`, `CVC4`, `iSAT3`, `MathSAT5`, `OpenSMT2`, `SMT-RAT`, `veriT`, `Yices2`, `Z3`

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)

SMT solvers: `Alt-Ergo`, `AProVE`, `iSAT3`, `MiniSmt`, `SMT-RAT`, `Z3`

# Some corresponding implementations in CAS

## Gröbner bases

- `CoCoA`, `F4`, `Maple`, `Mathematica`, `Maxima`, `Singular`, `Reduce`, ...

## Cylindrical algebraic decomposition (CAD)

- `Mathematica`, `QEPCAD`, `Reduce`, ...

## Virtual substitution (VS)

- `Reduce`, ...

Strength: Efficient for conjunctions of real constraints.

# Some corresponding implementations in CAS

## Gröbner bases

- CoCoA, F4, Maple, Mathematica, Maxima, Singular, Reduce, ...

## Cylindrical algebraic decomposition (CAD)

- Mathematica, QEPCAD, Reduce, ...

## Virtual substitution (VS)

- Reduce, ...

Strength: Efficient for conjunctions of real constraints.

So why don't we just plug in an algebraic decision procedure as theory solver into an SMT solver?

# Why not use CAS 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.

# Why not use CAS 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,
  - they are seldomly available as libraries, and
  - are usually not thread-safe.

# Why not use CAS 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,
  - they are seldomly available as libraries, and
  - are usually not thread-safe.
- Usually, SMT-adaptations are tricky.

# Our SMT-RAT library

We have developed the SMT-RAT library of theory modules.

[SAT'12, SAT'15]

`https://github.com/smtrat/smtrat/wiki`

# Some experimental results

We compare:

- `Z3` (SMT solver, Microsoft)
- `redlog` (reference implementation of virtual substitution in `Reduce`)
- SMT-RAT with two strategies.

$\texttt{rat}_1$: CNF → Preproc → SAT ⟶ ICP ⟶ VirtualSub ⟶ CAD

$\texttt{rat}_2$: CNF → Preproc
- → SAT ⟶ ICP ⟶ VirtualSub ⟶ CAD
- → SAT → Simplex → VirtualSub ⟶ CAD

# Some experimental results

| Benchmark (#examples) | z3 # | z3 time | redlog # | redlog time | $rat_1$ # | $rat_1$ time | $rat_2$ # | $rat_2$ time |
|---|---|---|---|---|---|---|---|---|
| HONG (20) | 40.0% | 5.6 | 30.0% | 3.7 | 100.0% | < 1 | 100.0% | < 1 |
| - sat | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| - unsat | 8 | 3.7 | 6 | 5.6 | 20 | < 1 | 20 | < 1 |
| KISSING (45) | 68.9% | 1248.7 | 13.3% | 3.3 | 35.6% | 375.9 | 28.9% | 54.4 |
| - sat | 31 | 1248.7 | 6 | 3.3 | 16 | 375.9 | 13 | 54.4 |
| - unsat | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| METI-TARSKI (7713) | 99.9% | 405.6 | 96.6% | 11617.9 | 92.8% | 4658.3 | 95.6% | 3109.4 |
| - sat | 5025 | 140.8 | 4859 | 7128.7 | 4740 | 2952.1 | 4815 | 2290.4 |
| - unsat | 2681 | 264.8 | 2590 | 4489.2 | 2418 | 1706.2 | 2560 | 819 |
| ZANKL (166) | 53.0% | 267.6 | 22.3% | 178.0 | 25.9% | 217.4 | 25.9% | 101.3 |
| - sat | 61 | 266.3 | 27 | 156.0 | 27 | 216.8 | 26 | 80.4 |
| - unsat | 27 | 1.3 | 10 | 22.0 | 16 | < 1 | 17 | 20.9 |
| KEYMAERA (421) | 99.8% | 11.8 | 99.5% | 209.3 | 96.9% | 17 | 98.1% | 25.3 |
| - sat | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| - unsat | 420 | 11.8 | 419 | 209.3 | 408 | 17 | 413 | 25.3 |
| WITNESS (99) | 21.2% | 153.5 | 5.1% | 62.1 | 64.6% | 332.2 | 75.8% | 937.9 |
| - sat | 4 | 106 | 5 | 62.1 | 47 | 331.9 | 58 | 937.6 |
| - unsat | 17 | 47.5 | 0 | 0 | 17 | < 1 | 17 | < 1 |

# Upcoming research directions in SMT solving

Improve usability:

- User-friendly models
- Dedicated SMT solvers

Increase scalability:

- Performance optimisation (better lemmas, heuristics, cache behaviour, . . . )
- Novel combination of decision procedures
- Parallelisation

Extend functionality:

- Unsatisfiable cores, proofs, interpolants
- Quantified arithmetic formulas
- Linear and non-linear (global) optimisation