Satisfiability Checking: Theory and Applications

Erika Ábrahám

RWTH Aachen University, Germany

STAF/SEFM 2016
July 06, 2016

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 quantifier-free first-order logic over different theories
using SAT modulo theories (SMT) solving, and on applications of such technologies.

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

1940

1960

1970

1980

2000

2010

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

Computer algebra systems

1960 First computer algebra systems

Gröbner bases

1970

CAD

1980

Partial CAD

Virtual substitution

2000

2010

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

1940

Computer architecture development

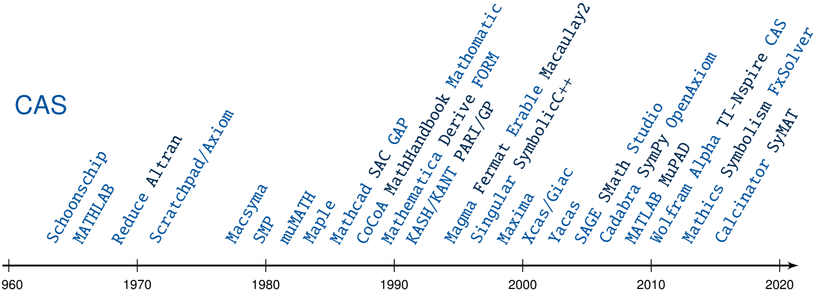
Computer algebra systems	SAT solvers (propositional logic)
	Enumeration
(1960) First computer algebra systems	DP (resolution) [Davis, Putnam'60]
	DPLL (propagation) [Davis,Putnam,Logemann,Loveland'62]
Gröbner bases	
(1970)	NP-completeness [Cook'71]
CAD	Conflict-directed backjumping
(1980)	
Partial CAD	
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

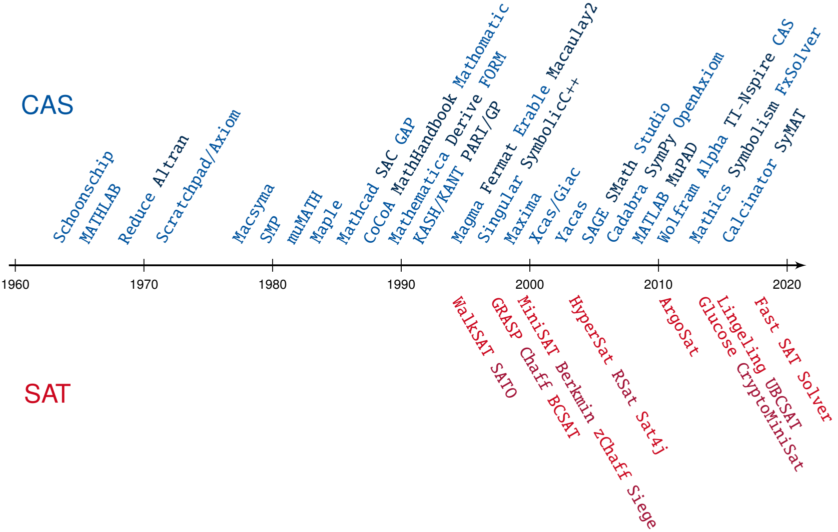
1940 Computer architecture development

Year	Computer algebra systems	SAT solvers (propositional logic)	SMT solvers (SAT modulo theories)
		Enumeration	
1960	First computer algebra systems	DP (resolution) [Davis, Putnam'60]	
	Gröbner bases	DPLL (propagation) [Davis,Putnam,Logemann,Loveland'62]	Decision procedures for combined theories [Shostak'79] [Nelson, Oppen'79]
1970		NP-completeness [Cook'71]	
	CAD	Conflict-directed backjumping	
1980			
	Partial CAD		
	Virtual substitution	CDCL [GRASP'97]	DPLL(T)
2000		Watched literals [zChaff'04]	Equalities
		Clause learning/forgetting	Uninterpreted functions
		Variable ordering heuristics	Bit-vector arithmetic
		Restarts	Array theory
2010			Arithmetic...

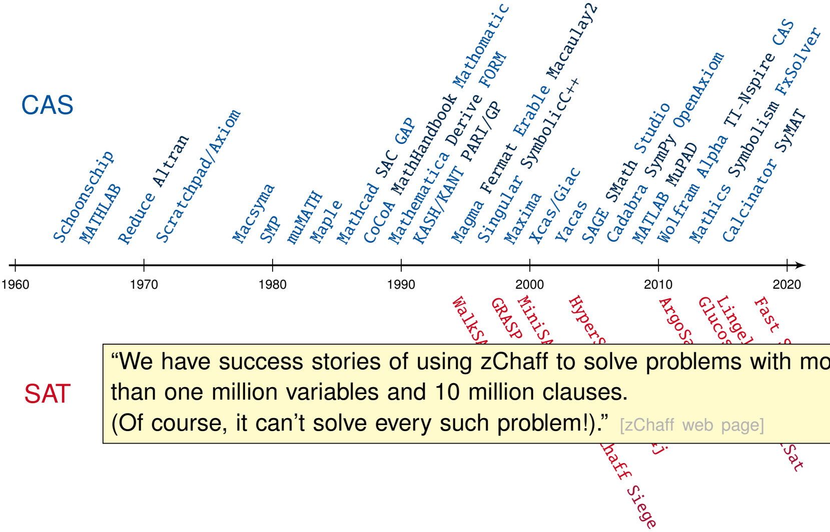
Tool development



Tool development

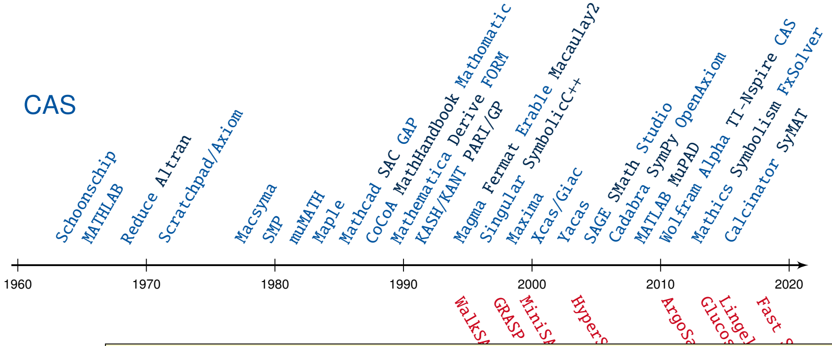


Tool development



“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]

Tool development

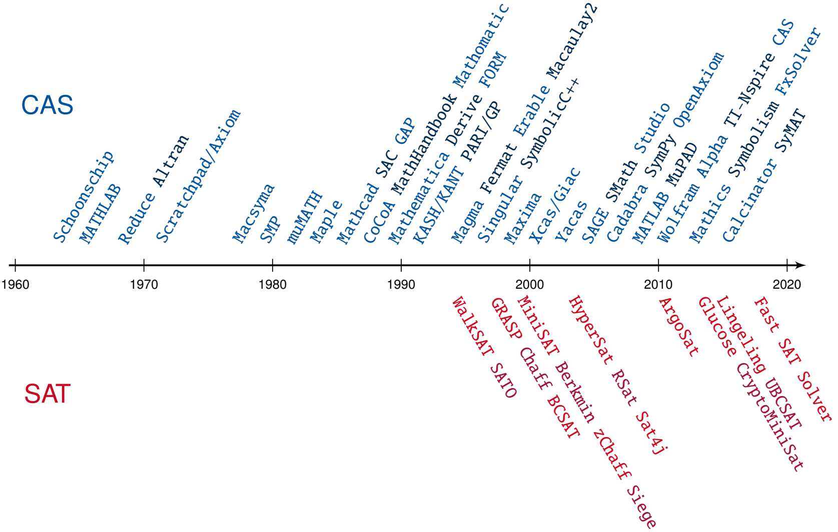


SAT

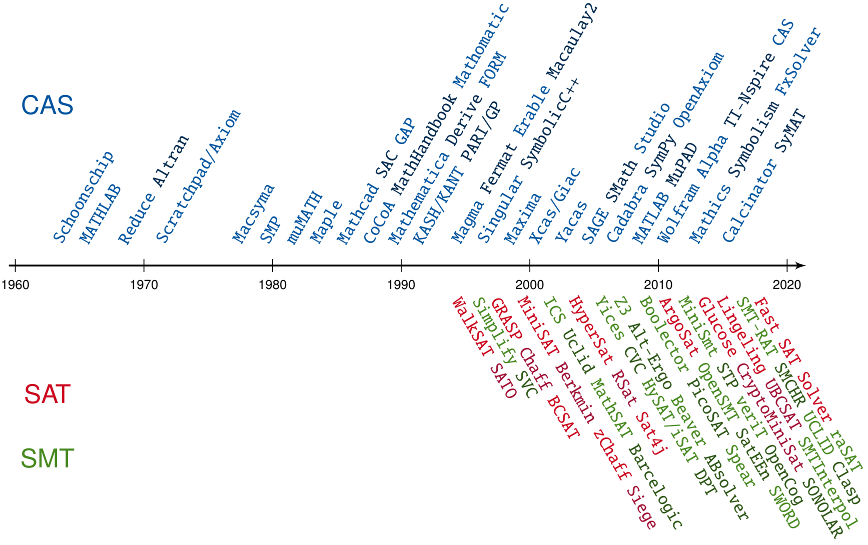
> "We have success stories of using zChaff to solve problems with more than one million variables and 10 million clauses.
> (Of course, it can't solve every such problem!)." [zChaff web page]

> " 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.

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

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

DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

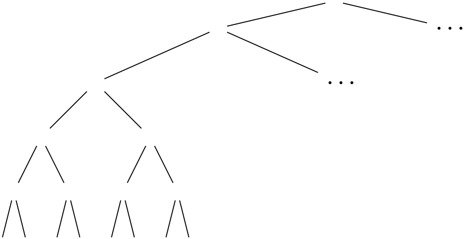
$$
\begin{array}{lcl}
c_1 : & (& \neg a \vee d \vee e \;) \\
c_2 : & (& \neg a \vee d \vee \neg e \;) \\
c_3 : & (& \neg a \vee \neg d \vee e \;) \\
c_4 : & (& \neg a \vee \neg d \vee \neg e \;) \\
c_5 : & (& a \vee b \;) \\
c_6 : & (& a \vee \neg b \;) \\
c_7 : & (& b \vee c \;) \\
c_8 : & (& \neg b \vee \neg c \;)
\end{array}
$$

DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration

$$
\begin{array}{llcccccccccc}
c_1 : & (& \neg a & \vee & & & & d & \vee & e &) \\
c_2 : & (& \neg a & \vee & & & & d & \vee & \neg e &) \\
c_3 : & (& \neg a & \vee & & & & \neg d & \vee & e &) \\
c_4 : & (& \neg a & \vee & & & & \neg d & \vee & \neg e &) \\
c_5 : & (& a & \vee & b & & & & & &) \\
c_6 : & (& a & \vee & \neg b & & & & & &) \\
c_7 : & (& & & b & \vee & c & & & &) \\
c_8 : & (& & & \neg b & \vee & \neg c & & & &)
\end{array}
$$

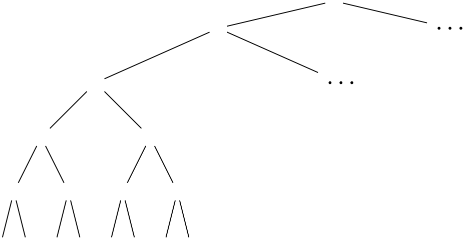


DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration

$$
\begin{array}{lllllllllll}
c_1 : & (& \neg a & \vee & & & & d & \vee & e &) \\
c_2 : & (& \neg a & \vee & & & & d & \vee & \neg e &) \\
c_3 : & (& \neg a & \vee & & & & \neg d & \vee & e &) \\
c_4 : & (& \neg a & \vee & & & & \neg d & \vee & \neg e &) \\
c_5 : & (& a & \vee & b & & & & & &) \\
c_6 : & (& a & \vee & \neg b & & & & & &) \\
c_7 : & (& & & b & \vee & c & & & &) \\
c_8 : & (& & & \neg b & \vee & \neg c & & & &)
\end{array}
$$



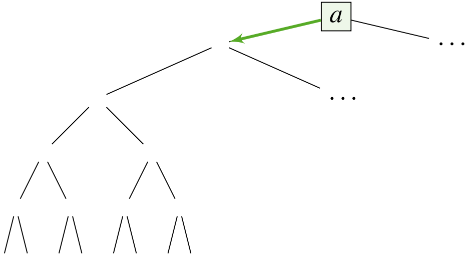
Decision

DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration

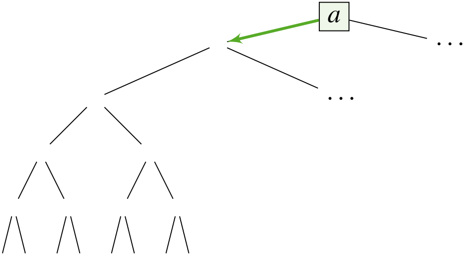
$$
\begin{array}{lllllllll}
c_1: & (& \neg a & \vee & & & d & \vee & e &) \\
c_2: & (& \neg a & \vee & & & d & \vee & \neg e &) \\
c_3: & (& \neg a & \vee & & & \neg d & \vee & e &) \\
c_4: & (& \neg a & \vee & & & \neg d & \vee & \neg e &) \\
c_5: & (& a & \vee & b & & & & &) \\
c_6: & (& a & \vee & \neg b & & & & &) \\
c_7: & (& & & b \vee c & & & & &) \\
c_8: & (& & & \neg b \vee \neg c & & & & &)
\end{array}
$$



Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration

$$
\begin{aligned}
c_1 &: (\neg a \vee d \vee e) \\
c_2 &: (\neg a \vee d \vee \neg e) \\
c_3 &: (\neg a \vee \neg d \vee e) \\
c_4 &: (\neg a \vee \neg d \vee \neg e) \\
c_5 &: (a \vee b) \\
c_6 &: (a \vee \neg b) \\
c_7 &: (b \vee c) \\
c_8 &: (\neg b \vee \neg c)
\end{aligned}
$$



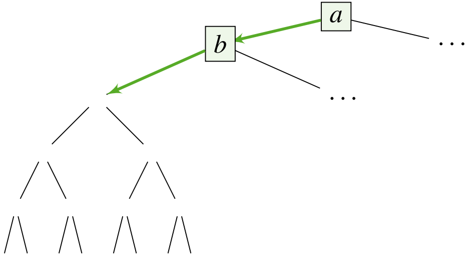
Decision

DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration

$$c_1 : (\neg a \vee d \vee e)$$
$$c_2 : (\neg a \vee d \vee \neg e)$$
$$c_3 : (\neg a \vee \neg d \vee e)$$
$$c_4 : (\neg a \vee \neg d \vee \neg e)$$
$$c_5 : (a \vee b)$$
$$c_6 : (a \vee \neg b)$$
$$c_7 : (b \vee c)$$
$$c_8 : (\neg b \vee \neg c)$$

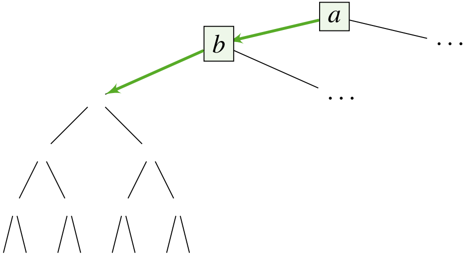


DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation

$$c_1 : (\neg a \vee d \vee e)$$
$$c_2 : (\neg a \vee d \vee \neg e)$$
$$c_3 : (\neg a \vee \neg d \vee e)$$
$$c_4 : (\neg a \vee \neg d \vee \neg e)$$
$$c_5 : (a \vee b)$$
$$c_6 : (a \vee \neg b)$$
$$c_7 : (b \vee c)$$
$$c_8 : (\neg b \vee \neg c)$$



DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation

$c_1 : (\neg a \vee d \vee e)$

$c_2 : (\neg a \vee d \vee \neg e)$

$c_3 : (\neg a \vee \neg d \vee e)$

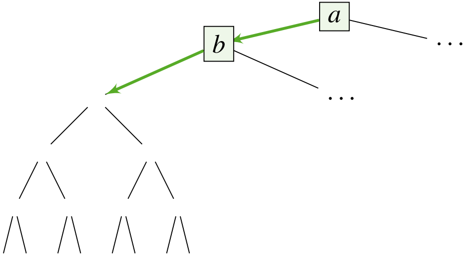
$c_4 : (\neg a \vee \neg d \vee \neg e)$

$c_5 : (a \vee b)$

$c_6 : (a \vee \neg b)$

$c_7 : (b \vee c)$

$c_8 : (\neg b \vee \neg c)$



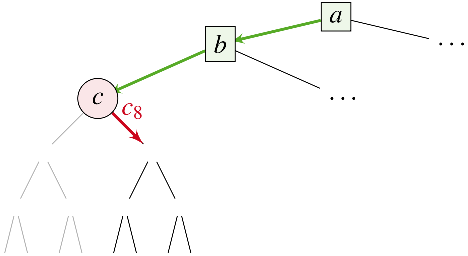
Boolean constraint propagation

DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation

$$c_1 : (\neg a \vee d \vee e)$$
$$c_2 : (\neg a \vee d \vee \neg e)$$
$$c_3 : (\neg a \vee \neg d \vee e)$$
$$c_4 : (\neg a \vee \neg d \vee \neg e)$$
$$c_5 : (a \vee b)$$
$$c_6 : (a \vee \neg b)$$
$$c_7 : (b \vee c)$$
$$c_8 : (\neg b \vee \neg c)$$

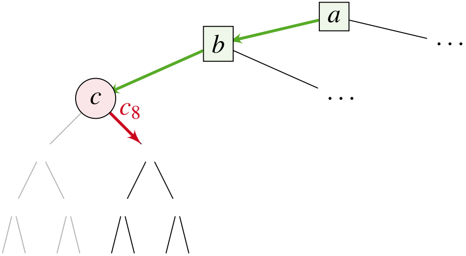


DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation

$$c_1 : (\neg a \lor d \lor e)$$
$$c_2 : (\neg a \lor d \lor \neg e)$$
$$c_3 : (\neg a \lor \neg d \lor e)$$
$$c_4 : (\neg a \lor \neg d \lor \neg e)$$
$$c_5 : (a \lor b)$$
$$c_6 : (a \lor \neg b)$$
$$c_7 : (b \lor c)$$
$$c_8 : (\neg b \lor \neg c)$$



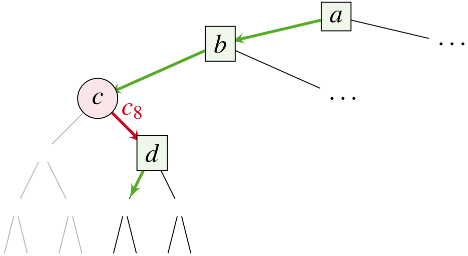
Decision

DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation

c_1	:	(	$\neg a$	$\vee$				d	$\vee$	e	)
c_2	:	(	$\neg a$	$\vee$				d	$\vee$	$\neg e$	)
c_3	:	(	$\neg a$	$\vee$				$\neg d$	$\vee$	e	)
c_4	:	(	$\neg a$	$\vee$				$\neg d$	$\vee$	$\neg e$	)
c_5	:	(	a	$\vee$	b						)
c_6	:	(	a	$\vee$	$\neg b$						)
c_7	:	(			b	$\vee$	c				)
c_8	:	(			$\neg b$	$\vee$	$\neg c$				)

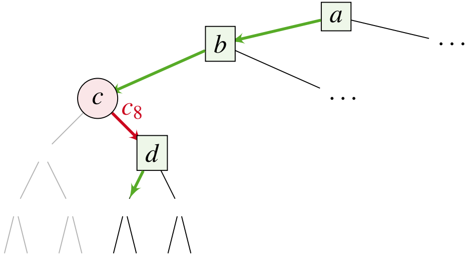


DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation

c_1 :	(	$\neg a$	$\vee$				d	$\vee$	e	)
c_2 :	(	$\neg a$	$\vee$				d	$\vee$	$\neg e$	)
c_3 :	(	$\neg a$	$\vee$				$\neg d$	$\vee$	e	)
c_4 :	(	$\neg a$	$\vee$				$\neg d$	$\vee$	$\neg e$	)
c_5 :	(	a	$\vee$	b						)
c_6 :	(	a	$\vee$	$\neg b$						)
c_7 :	(			b	$\vee$	c				)
c_8 :	(			$\neg b$	$\vee$	$\neg c$				)



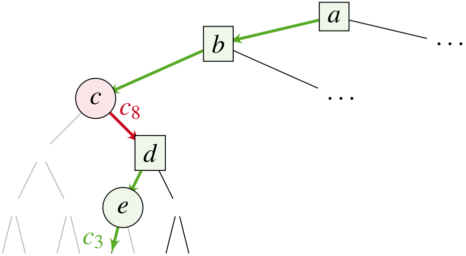
Boolean constraint propagation

DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation

c_1 :	(	$\neg a$	$\vee$				d	$\vee$	e	)
c_2 :	(	$\neg a$	$\vee$				d	$\vee$	$\neg e$	)
c_3 :	(	$\neg a$	$\vee$				$\neg d$	$\vee$	e	)
c_4 :	(	$\neg a$	$\vee$				$\neg d$	$\vee$	$\neg e$	)
c_5 :	(	a	$\vee$	b						)
c_6 :	(	a	$\vee$	$\neg b$						)
c_7 :	(			b	$\vee$	c				)
c_8 :	(			$\neg b$	$\vee$	$\neg c$				)

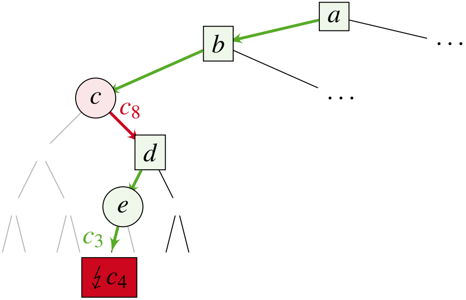


DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation

c_1 :	(	$\neg a$	$\vee$				d	$\vee$	e	)
c_2 :	(	$\neg a$	$\vee$				d	$\vee$	$\neg e$	)
c_3 :	(	$\neg a$	$\vee$				$\neg d$	$\vee$	e	)
c_4 :	(	$\neg a$	$\vee$				$\neg d$	$\vee$	$\neg e$	)
c_5 :	(	a	$\vee$	b						)
c_6 :	(	a	$\vee$	$\neg b$						)
c_7 :	(			b	$\vee$	c				)
c_8 :	(			$\neg b$	$\vee$	$\neg c$				)

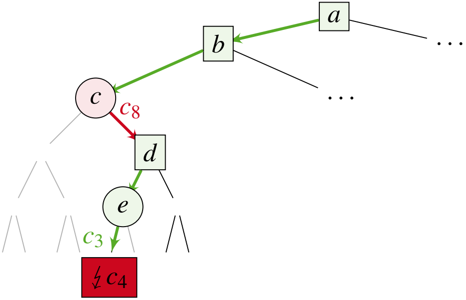


DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation

c_1 : (	$\neg a$	$\vee$			d $\vee$	e)
c_2 : (	$\neg a$	$\vee$			d $\vee$	$\neg e$)
c_3 : (	$\neg a$	$\vee$			$\neg d$ $\vee$	e)
c_4 : (	$\neg a$	$\vee$			$\neg d$ $\vee$	$\neg e$)
c_5 : (	a	$\vee$	b			)
c_6 : (	a	$\vee$	$\neg b$			)
c_7 : (			b	$\vee$ c		)
c_8 : (			$\neg b$	$\vee$ $\neg c$		)



Conflict

Resolution

Assumption: conjunctive normal form (CNF)

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:

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

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

Resolution

Assumption: conjunctive normal form (CNF)

Derivation rule form:

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

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

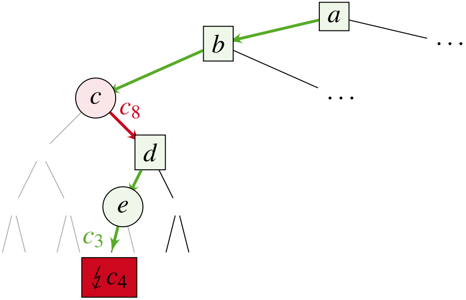
$$\exists x.\ C_x \wedge C_{\neg x} \wedge C \quad \leftrightarrow \quad \textit{Resolvents}(C_x, C_{\neg x}) \wedge C$$

DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation

c_1 :	(	$\neg a$	$\vee$				d	$\vee$	e	)
c_2 :	(	$\neg a$	$\vee$				d	$\vee$	$\neg e$	)
c_3 :	(	$\neg a$	$\vee$				$\neg d$	$\vee$	e	)
c_4 :	(	$\neg a$	$\vee$				$\neg d$	$\vee$	$\neg e$	)
c_5 :	(	a	$\vee$	b						)
c_6 :	(	a	$\vee$	$\neg b$						)
c_7 :	(			b	$\vee$	c				)
c_8 :	(			$\neg b$	$\vee$	$\neg c$				)



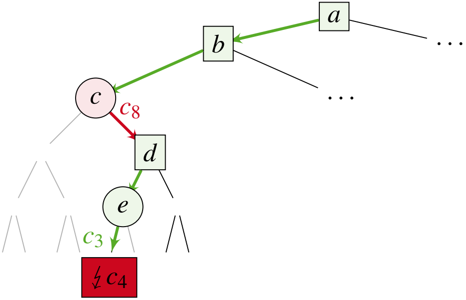
Conflict

DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation + Resolution

c_1 :	(	$\neg a$	$\vee$				d	$\vee$	e	)
c_2 :	(	$\neg a$	$\vee$				d	$\vee$	$\neg e$	)
c_3 :	(	$\neg a$	$\vee$				$\neg d$	$\vee$	e	)
c_4 :	(	$\neg a$	$\vee$				$\neg d$	$\vee$	$\neg e$	)
c_5 :	(	a	$\vee$	b						)
c_6 :	(	a	$\vee$	$\neg b$						)
c_7 :	(			b	$\vee$	c				)
c_8 :	(			$\neg b$	$\vee$	$\neg c$				)



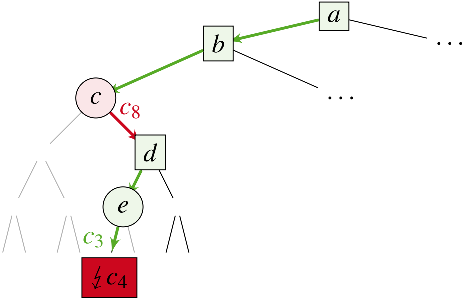
Conflict

DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation + Resolution

c_1 :	(	$\neg a$	$\vee$				d	$\vee$	e	)
c_2 :	(	$\neg a$	$\vee$				d	$\vee$	$\neg e$	)
c_3 :	(	$\neg a$	$\vee$				$\neg d$	$\vee$	e	)
c_4 :	(	$\neg a$	$\vee$				$\neg d$	$\vee$	$\neg e$	)
c_5 :	(	a	$\vee$	b						)
c_6 :	(	a	$\vee$	$\neg b$						)
c_7 :	(			b	$\vee$	c				)
c_8 :	(			$\neg b$	$\vee$	$\neg c$				)



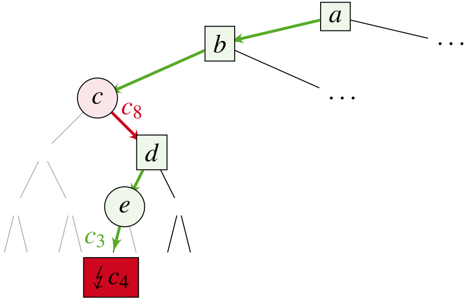
Conflict resolution and backtracking

DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation + Resolution

c_1 : (	$\neg a$	$\vee$				d	$\vee$	e	)
c_2 : (	$\neg a$	$\vee$				d	$\vee$	$\neg e$	)
c_3 : (	$\neg a$	$\vee$				$\neg d$	$\vee$	e	)
c_4 : (	$\neg a$	$\vee$				$\neg d$	$\vee$	$\neg e$	)
c_5 : (	a	$\vee$	b						)
c_6 : (	a	$\vee$	$\neg b$						)
c_7 : (			b	$\vee$	c				)
c_8 : (			$\neg b$	$\vee$	$\neg c$				)



$$\frac{c_4 : (\neg a \vee \neg d \vee \neg e) \quad c_3 : (\neg a \vee \neg d \vee e)}{c_9 : (\neg a \vee \neg d)}$$

DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation + Resolution

$c_1 : (\neg a \lor d \lor e)$

$c_2 : (\neg a \lor d \lor \neg e)$

$c_3 : (\neg a \lor \neg d \lor e)$

$c_4 : (\neg a \lor \neg d \lor \neg e)$

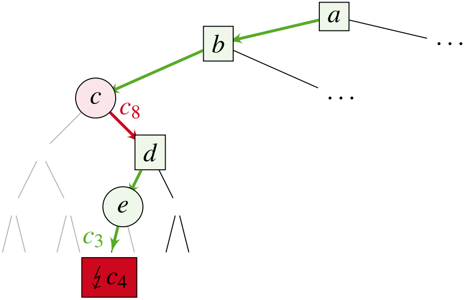
$c_5 : (a \lor b)$

$c_6 : (a \lor \neg b)$

$c_7 : (b \lor c)$

$c_8 : (\neg b \lor \neg c)$

$c_9 : (\neg a \lor \neg d)$



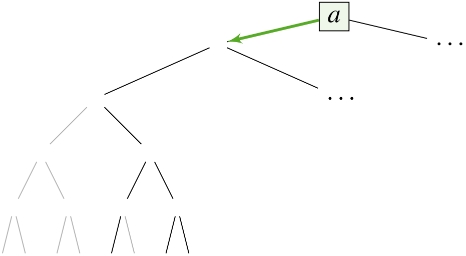
$$\frac{c_4 : (\neg a \lor \neg d \lor \neg e) \quad c_3 : (\neg a \lor \neg d \lor e)}{c_9 : (\neg a \lor \neg d)}$$

DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation + Resolution

c_1 :	(	$\neg a$	$\vee$					d	$\vee$	e	)
c_2 :	(	$\neg a$	$\vee$					d	$\vee$	$\neg e$	)
c_3 :	(	$\neg a$	$\vee$					$\neg d$	$\vee$	e	)
c_4 :	(	$\neg a$	$\vee$					$\neg d$	$\vee$	$\neg e$	)
c_5 :	(	a	$\vee$	b							)
c_6 :	(	a	$\vee$	$\neg b$							)
c_7 :	(			b	$\vee$	c					)
c_8 :	(			$\neg b$	$\vee$	$\neg c$					)
c_9 :	(	$\neg a$	$\vee$					$\neg d$			)

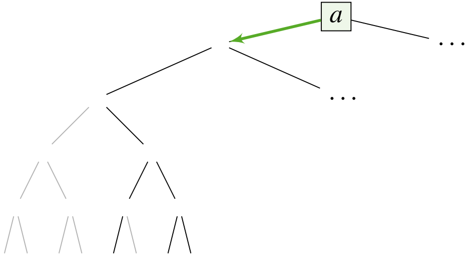


DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation + Resolution

c_1	:	(	$\neg a$	$\vee$				d	$\vee$	e	)
c_2	:	(	$\neg a$	$\vee$				d	$\vee$	$\neg e$	)
c_3	:	(	$\neg a$	$\vee$				$\neg d$	$\vee$	e	)
c_4	:	(	$\neg a$	$\vee$				$\neg d$	$\vee$	$\neg e$	)
c_5	:	(	a	$\vee$	b						)
c_6	:	(	a	$\vee$	$\neg b$						)
c_7	:	(			b	$\vee$	c				)
c_8	:	(			$\neg b$	$\vee$	$\neg c$				)
c_9	:	(	$\neg a$	$\vee$				$\neg d$			)



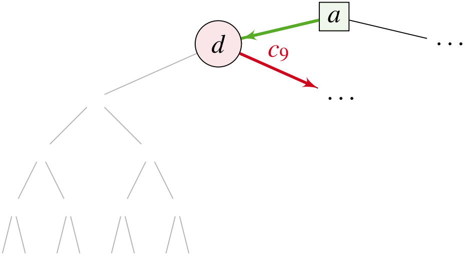
Boolean constraint propagation

DPLL SAT solving with conflict-directed clause learning

Assumption: formula in conjunctive normal form (CNF)

Ingredients: Enumeration + Boolean constraint propagation + Resolution

c_1 : (	$\neg a$	$\vee$				d	$\vee$	e	)
c_2 : (	$\neg a$	$\vee$				d	$\vee$	$\neg e$	)
c_3 : (	$\neg a$	$\vee$				$\neg d$	$\vee$	e	)
c_4 : (	$\neg a$	$\vee$				$\neg d$	$\vee$	$\neg e$	)
c_5 : (	a	$\vee$	b						)
c_6 : (	a	$\vee$	$\neg b$						)
c_7 : (			b	$\vee$	c				)
c_8 : (			$\neg b$	$\vee$	$\neg c$				)
c_9 : (	$\neg a$	$\vee$				$\neg d$			)



Satisfiability modulo theories solving

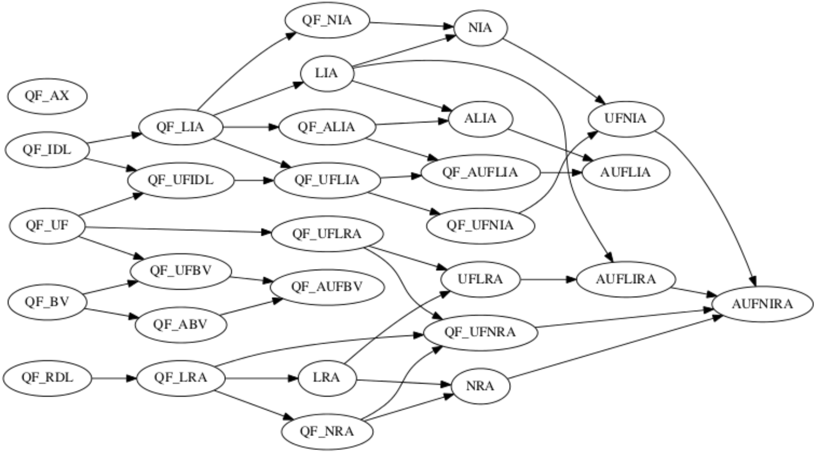
- Propositional logic is sometimes too weak for modelling.
- We need more expressive logics and decision procedures for them.
- Logics: quantifier-free (QF) fragments of first-order logic over various theories.
- Our focus: SAT-modulo-theories (SMT) solving.

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 (QF) fragments of first-order logic over various theories.
- Our focus: SAT-modulo-theories (SMT) solving.

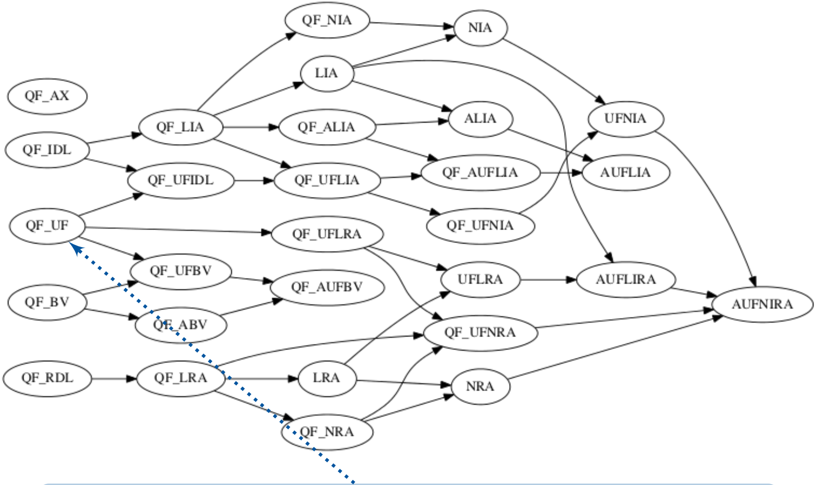
- SMT-LIB as standard input language since 2004.
- Competitions since 2005.
- SMT-COMP 2016 competition:
  - 4 tracks, 41 logical categories.
  - QF linear real arithmetic: $7 + 2$ solvers, 1626 benchmarks.
  - QF linear integer arithmetic: $6 + 2$ solvers, 5839 benchmarks.
  - QF non-linear real arithmetic: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

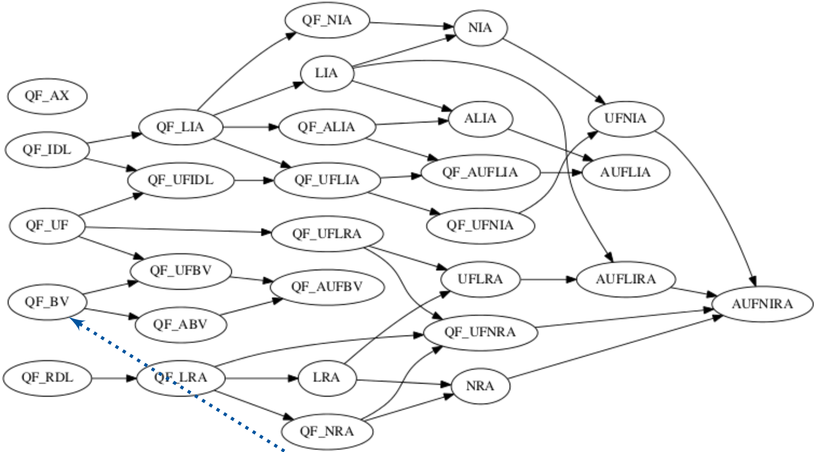


Quantifier-free equality logic with uninterpreted functions

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

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

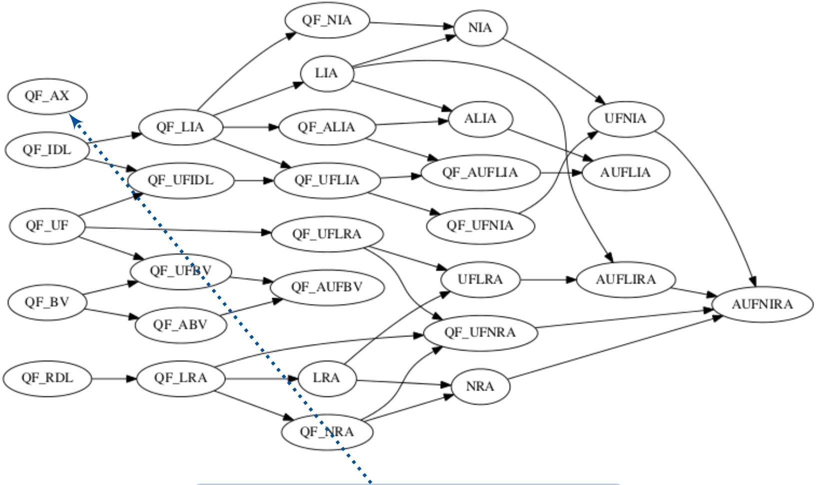
SMT-LIB theories



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

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

SMT-LIB theories

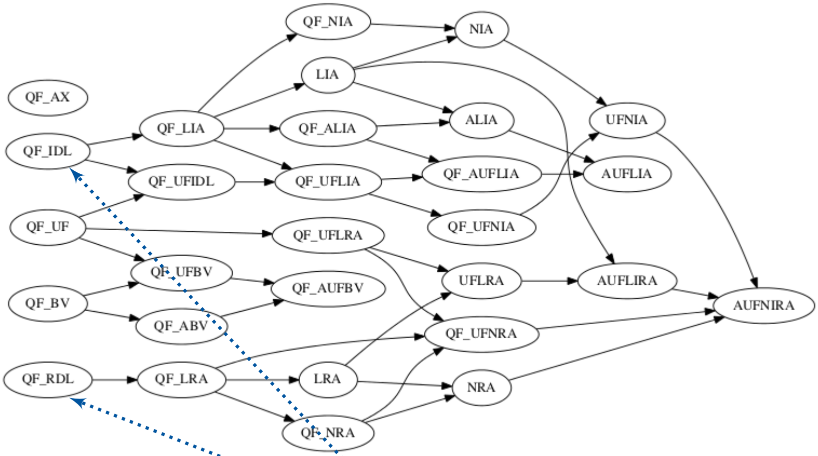


Quantifier-free array theory

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

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

SMT-LIB theories

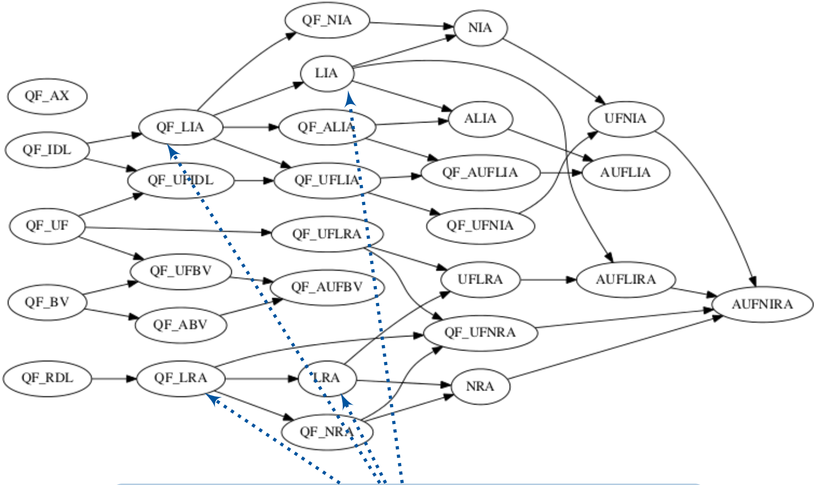


Quantifier-free integer/rational difference logic

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

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

SMT-LIB theories

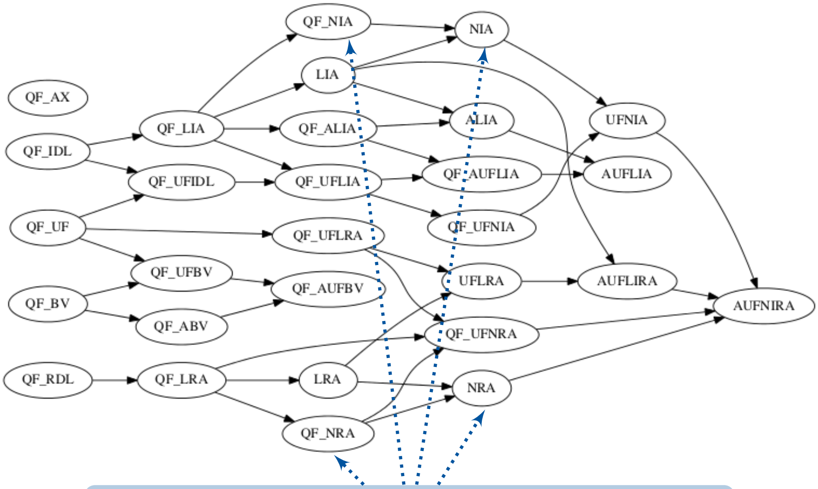


(Quantifier-free) real/integer linear arithmetic

$$3x + 7y = 8$$

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

SMT-LIB theories

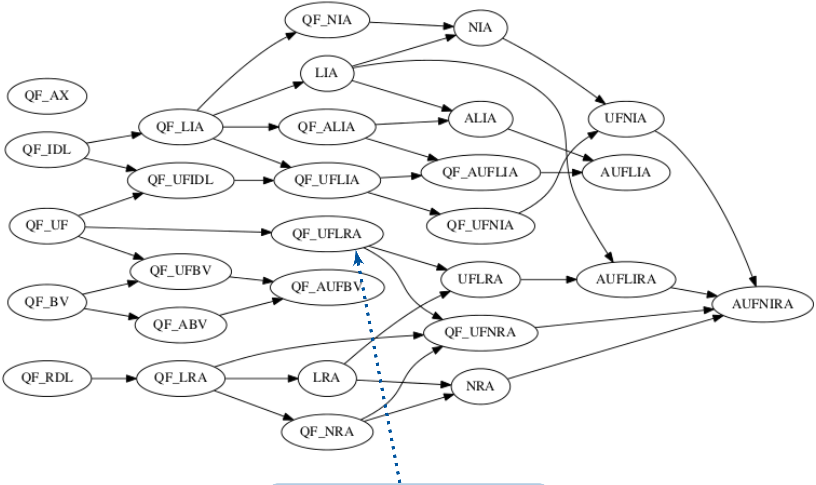


(Quantifier-free) real/integer non-linear arithmetic

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

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

SMT-LIB theories



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

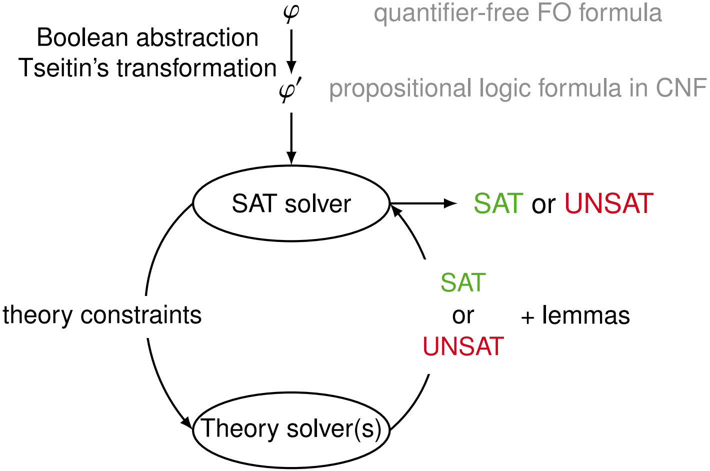
Source: `http://smtlib.cs.uiowa.edu/logics.shtml`

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



Less lazy SMT solving

Less lazy SMT solving

$$(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)$$

Less lazy SMT solving

$$(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)$$

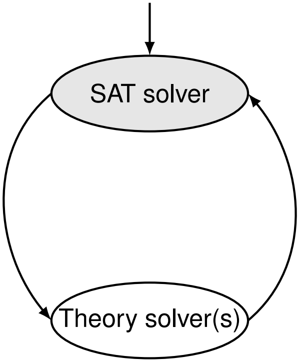
$$\downarrow$$

$$(\;a \;\lor\; b\;) \land (\;c \;\lor\; d\;)$$

Less lazy SMT solving

$$(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)$$

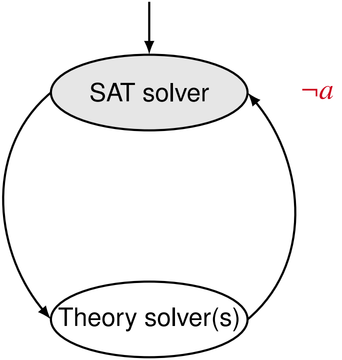
$$(\ a \ \lor \ b\ ) \land (\ c \ \lor \ d\ )$$



Less lazy SMT solving

$$(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)$$

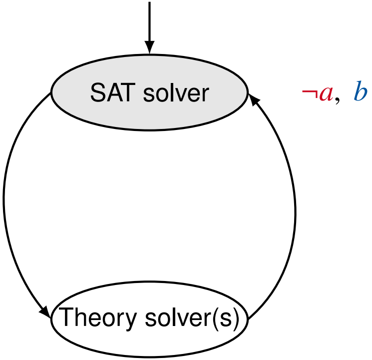
$$(\ a \ \lor \ b\ ) \land (\ c \ \lor \ d\ )$$



Less lazy SMT solving

$$(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)$$

$$(\ a \ \lor \ b \ ) \land (\ c \ \lor \ d \ )$$

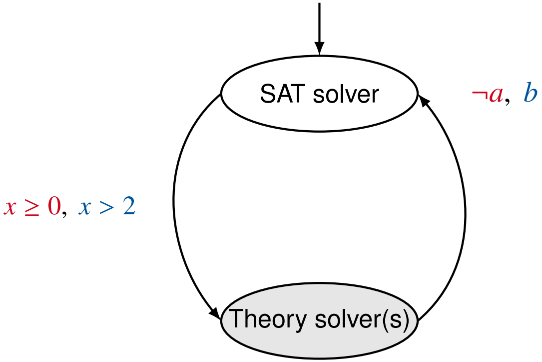


Less lazy SMT solving

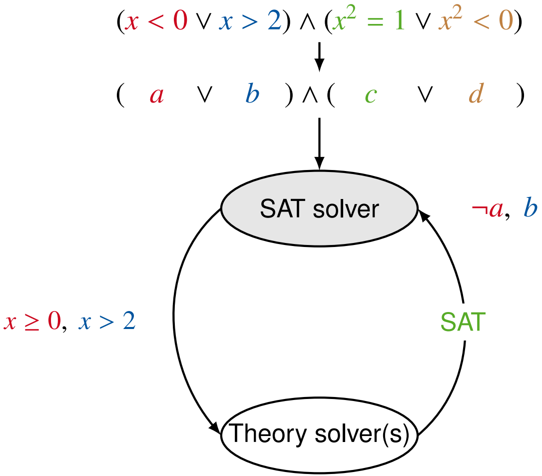
$$(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)$$

$$\downarrow$$

$$(a \lor b) \land (c \lor d)$$



Less lazy SMT solving

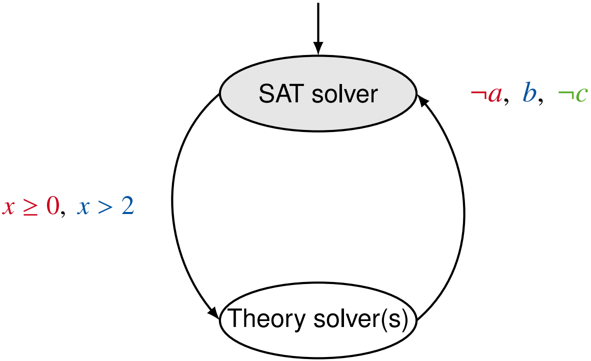


Less lazy SMT solving

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

$$\downarrow$$

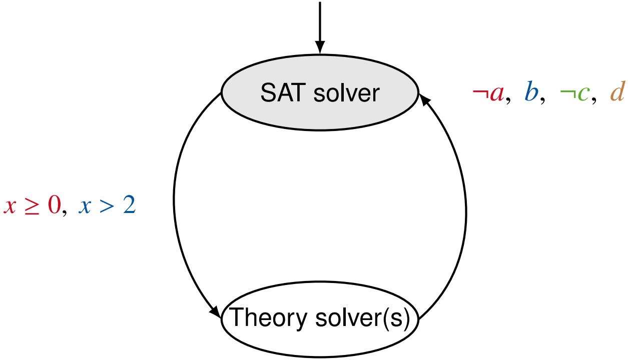
$$(\ a \ \vee \ b\ ) \wedge (\ c \ \vee \ d\ )$$



Less lazy SMT solving

$$(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)$$

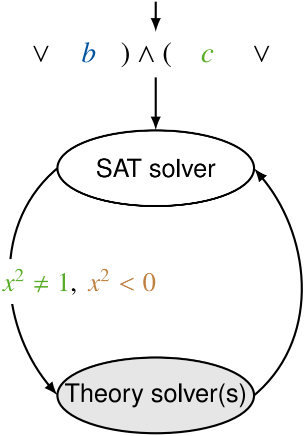
$$(a \lor b) \land (c \lor d)$$



Less lazy SMT solving

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

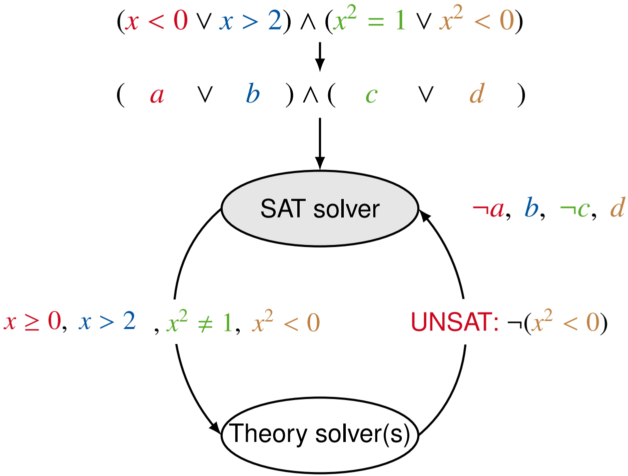
$(a \vee b) \wedge (c \vee d)$



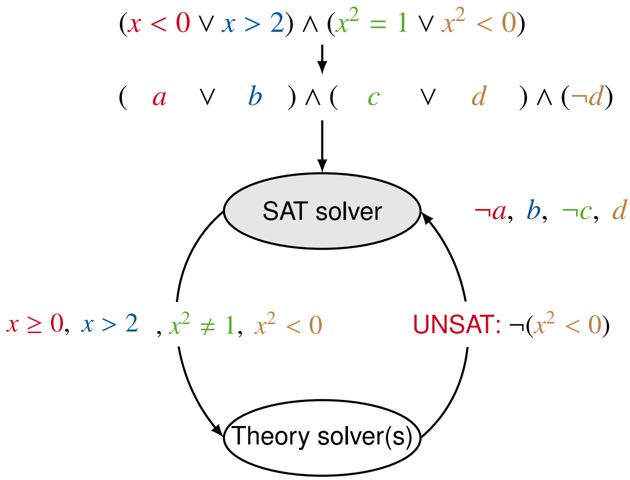
$\neg a,\ b,\ \neg c,\ d$

$x \geq 0,\ x > 2,\ x^2 \neq 1,\ x^2 < 0$

Less lazy SMT solving



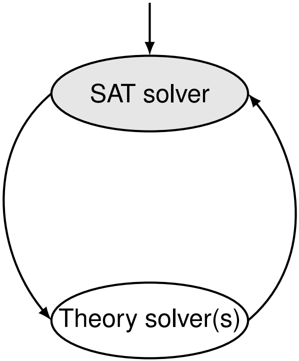
Less lazy SMT solving



Less lazy SMT solving

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

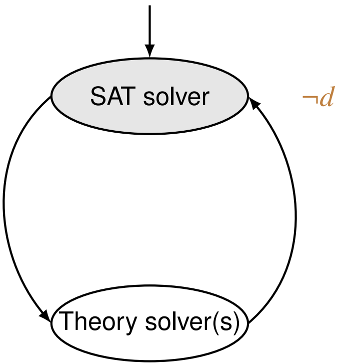
$$(\ a \ \vee \ b\ ) \wedge (\ c \ \vee \ d\ ) \wedge (\neg d)$$



Less lazy SMT solving

$$(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)$$

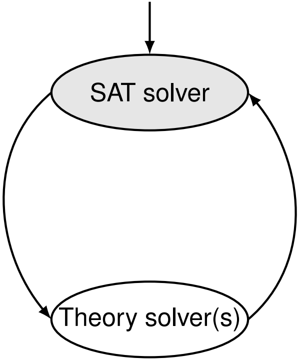
$$(a \lor b) \land (c \lor d) \land (\neg d)$$



Less lazy SMT solving

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

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

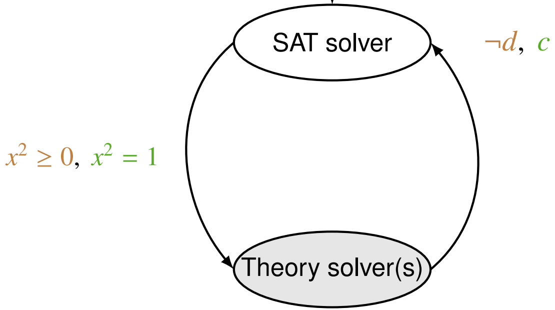


$\neg d,\ c$

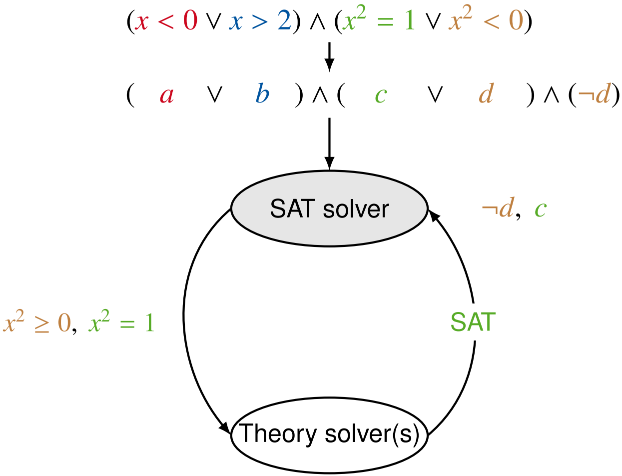
Less lazy SMT solving

$(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)$

$(\; a \;\lor\; b \;) \land (\; c \;\lor\; d \;) \land (\neg d)$



Less lazy SMT solving

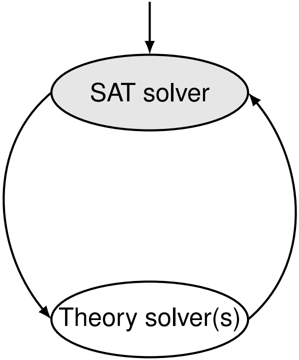


Less lazy SMT solving

$$(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)$$

$$\downarrow$$

$$(a \lor b) \land (c \lor d) \land (\neg d)$$



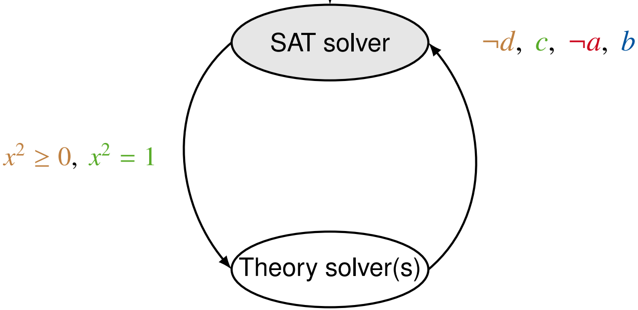
$\neg d,\ c,\ \neg a$

$x^2 \geq 0,\ x^2 = 1$

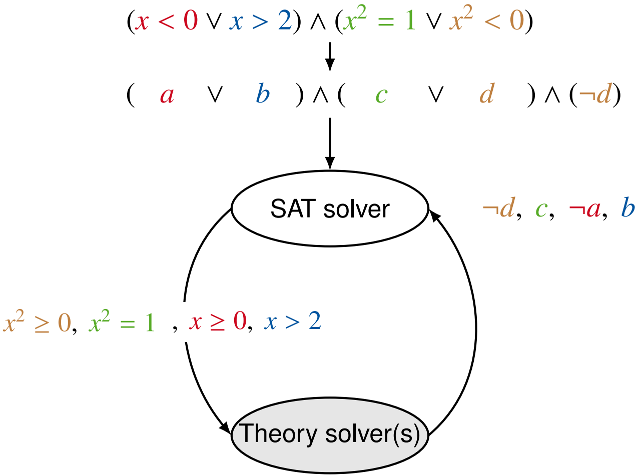
Less lazy SMT solving

$$(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)$$

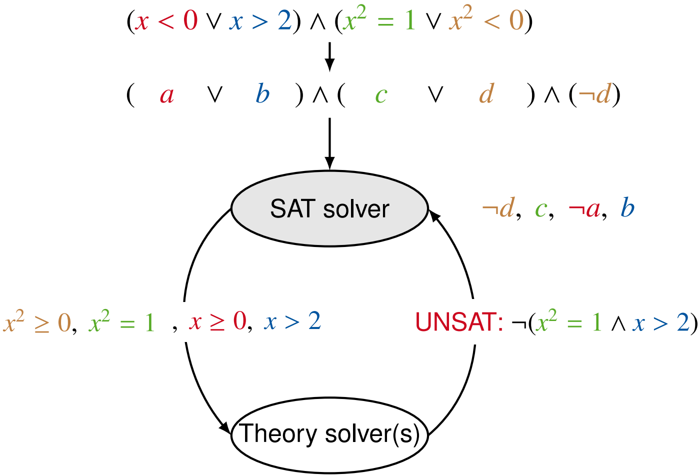
$$(a \lor b) \land (c \lor d) \land (\neg d)$$



Less lazy SMT solving

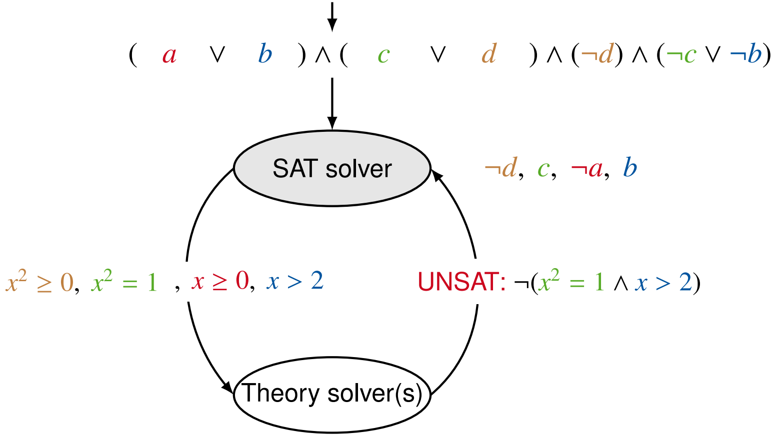


Less lazy SMT solving



Less lazy SMT solving

$(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)$



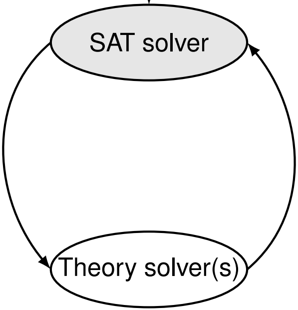
Less lazy SMT solving

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

$\downarrow$

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

$\downarrow$



$\neg d, \; c$

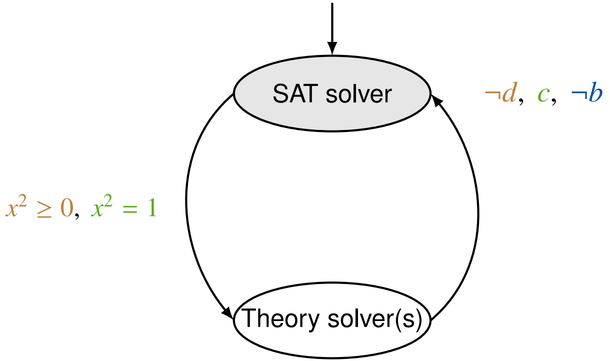
$x^2 \geq 0, \; x^2 = 1$

Less lazy SMT solving

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

$\downarrow$

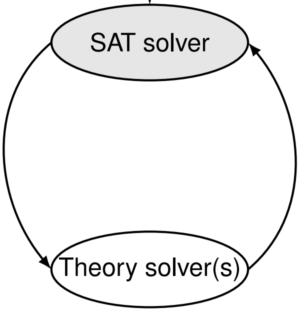
$(a \vee b) \wedge (c \vee d) \wedge (\neg d) \wedge (\neg c \vee \neg b)$



Less lazy SMT solving

$$(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)$$

$$(a \lor b) \land (c \lor d) \land (\neg d) \land (\neg c \lor \neg b)$$



$$\neg d,\ c,\ \neg b,\ a$$

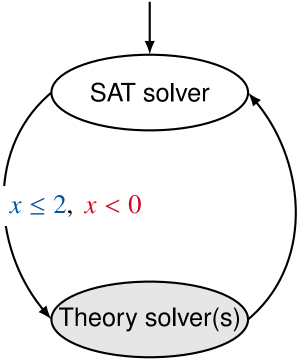
$$x^2 \geq 0,\ x^2 = 1$$

Less lazy SMT solving

$(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)$

$\downarrow$

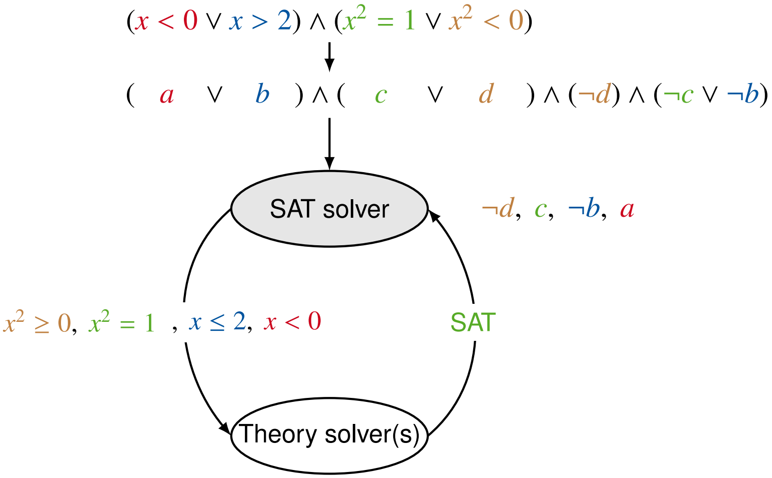
$(a \lor b) \land (c \lor d) \land (\neg d) \land (\neg c \lor \neg b)$



$\neg d,\ c,\ \neg b,\ a$

$x^2 \geq 0,\ x^2 = 1\ ,\ x \leq 2,\ x < 0$

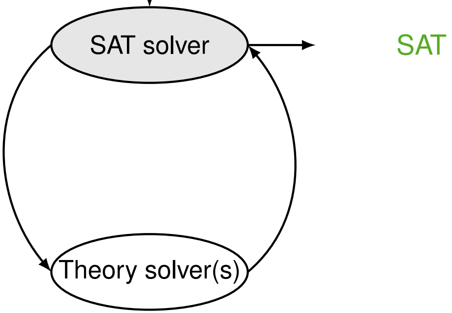
Less lazy SMT solving



Less lazy SMT solving

$$(x < 0 \lor x > 2) \land (x^2 = 1 \lor x^2 < 0)$$

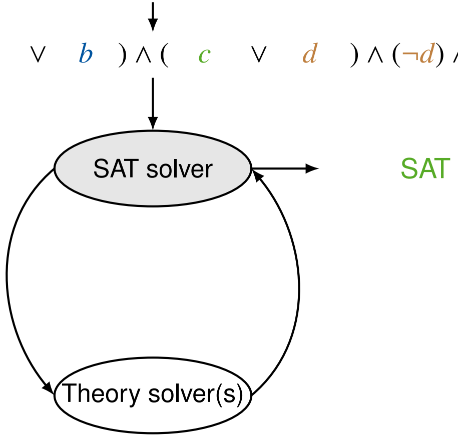
$$(a \lor b) \land (c \lor d) \land (\neg d) \land (\neg c \lor \neg b)$$



Less lazy SMT solving

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

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



N.B. There are also other SMT solving techniques, which more closely integrate some theory-solving parts into the SAT-solving mechanism.

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)

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.

Satisfiablility checking and symbolic computation

Bridging two communities to solve real problems

http://www.sc-square.org/CSA/welcome.html

SC^2

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 **SC^2 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

Satisfiablility checking and symbolic computation

Bridging two communities to solve real problems

http://www.sc-square.org/CSA/welcome.html

SC^2

Satisfiability Checking and Symbolic Computation

Bridging Two Communities to Solve Real Problems
Coordination and Support Activity

SUMMARY

This project is funded (subject to co
general push to create a real **SC^2 com**

Background

The use of advanced methods to sol
Computation is concerned with the
developments in the area of Satisfiabi
both communities have made remark
rapidly increasing size and complexity
prevailing problems with a direct effe
and the safety of computer systems.
out of reach of pencil and paper devel
other, despite strong reasons for them
communities rarely interact, and also their tools lack common, mutual interfaces for unifying their strengths. Bridges between the communities in the form of common platforms and roadmaps are necessary to initiate an exchange, and to support and to direct their interaction. These are the main objectives of this CSA. We will initiate a wide range of activities to bring the two communities together, identify common challenges, offer global events and bilateral visits, propose standards, and so on. We believe that these activities will

Consortium

University of Bath	**James Davenport**; **Russell Bradford**
RWTH Aachen	Erika Ábrahám
Fondazione Bruno Kessler	Alberto Griggio; Alessandro Cimatti
Università degli Studi di Genova	**Anna Bigatti**
Maplesoft Europe Ltd	Jürgen Gerhard; Stephen Forrest
Université de Lorraine (LORIA)	Pascal Fontaine
Coventry University	**Matthew England**
University of Oxford	Martin Brain
Universität Kassel	Werner Seiler; **John Abbott**
Max Planck Institut für Informatik	Thomas Sturm
Universität Linz	**Bruno Buchberger**; **Wolfgang Windsteiger**; Roxana-Maria Holom

Our SMT-RAT library

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

[SAT'12, SAT'15]

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

SMT RAT

Florian Corzilius

Gereon Kremer

Ulrich Loup

Our SMT-RAT library

SMT Solver

Strategic composition of SMT-RAT modules

SMT-RAT

(SMT real-algebraic toolbox)

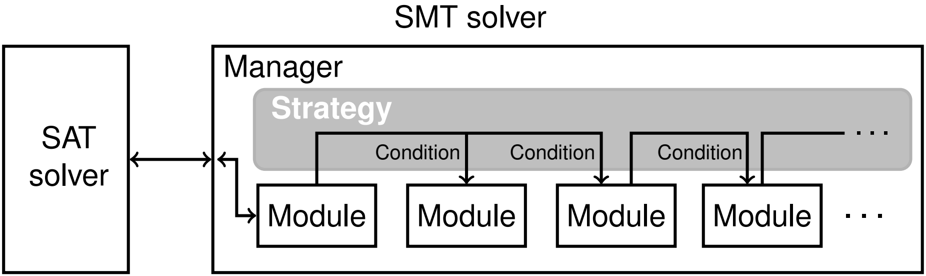
preprocessing, SAT and theory solver modules

CArL

real-arithmetic computations

gmp, Eigen3, boost

Strategic composition of solver modules in SMT-RAT



Solver modules in SMT-RAT

- Libraries for basic arithmetic computations [NFM'11, CAI'11]
- SAT solver
- CNF converter
- Preprocessing/simplifying modules
- Interval constraint propagation
- Simplex
- Virtual substitution [FCT'11, PhD Corzilius]
- Cylindrical algebraic decomposition
  [CADE-24, PhD Loup, PhD Kremer]
- Gröbner bases [CAI'13]
- Generalised branch-and-bound [CASC'16]

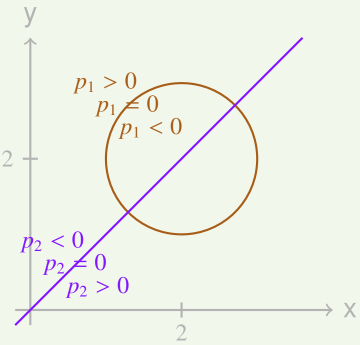
Solution sets and P-sign-invariant regions

$\mathbb{Z}[x_1, \ldots, x_n]$ is the set of all polynomials over variables $x_1, \ldots, x_n$.

Example

$$\left.\begin{array}{l} p_1 = (x-2)^2 + (y-2)^2 - 1 \\ p_2 = x - y \end{array}\right\} \in \mathbb{Z}[x,y]$$

$$C = \{\, p_1 < 0,\ p_2 = 0 \,\}$$
$$P = \{\, p_1,\ p_2 \,\}$$



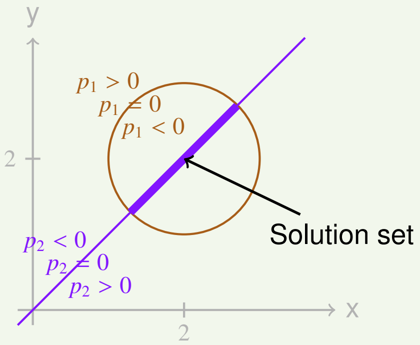
Solution sets and P-sign-invariant regions

$\mathbb{Z}[x_1, \ldots, x_n]$ is the set of all polynomials over variables $x_1, \ldots, x_n$.

Example

$$\left.\begin{array}{l} p_1 = (x-2)^2 + (y-2)^2 - 1 \\ p_2 = x - y \end{array}\right\} \in \mathbb{Z}[x, y]$$

$$C = \{\, p_1 < 0,\ p_2 = 0 \,\}$$
$$P = \{\, p_1,\ p_2 \,\}$$



Solution sets and P-sign-invariant regions

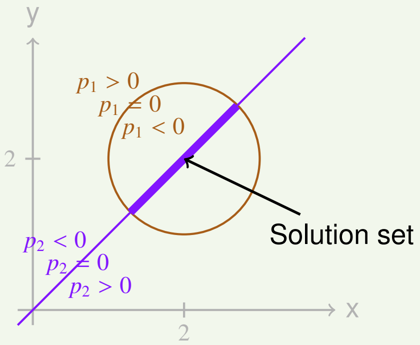
$\mathbb{Z}[x_1, \ldots, x_n]$ is the set of all polynomials over variables $x_1, \ldots, x_n$.

Example

$$\left.\begin{array}{l} p_1 = (x-2)^2 + (y-2)^2 - 1 \\ p_2 = x - y \end{array}\right\} \in \mathbb{Z}[x, y]$$

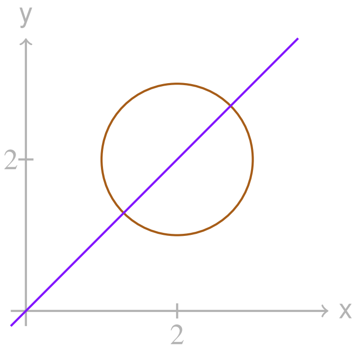
$C = \{\, p_1 < 0,\ p_2 = 0 \,\}$

$P = \{\, p_1,\ p_2 \,\}$

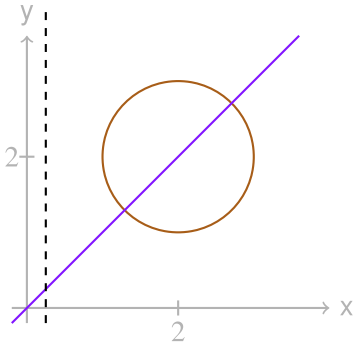


Solution set ≡ finite union of P-sign-invariant regions

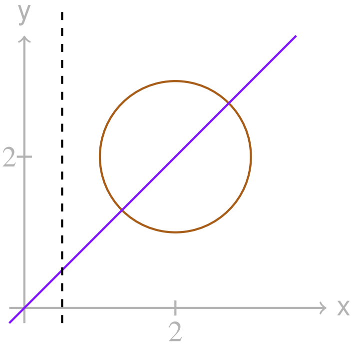
Delineability on an example



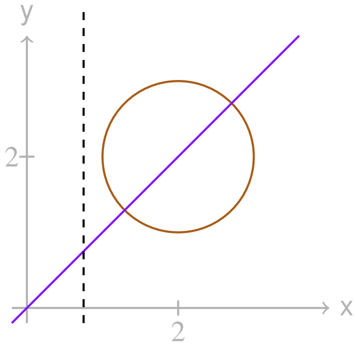
Delineability on an example



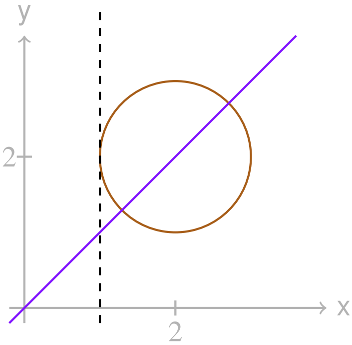
Delineability on an example



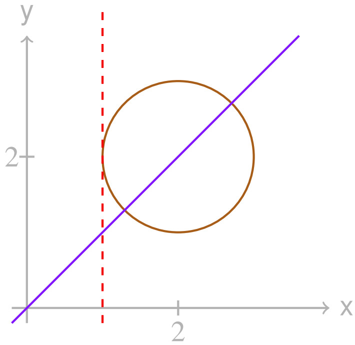
Delineability on an example



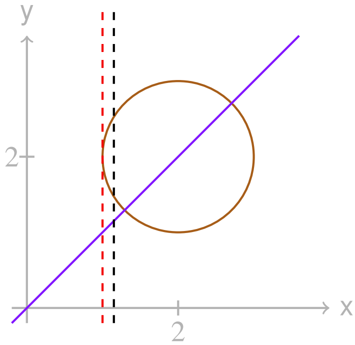
Delineability on an example



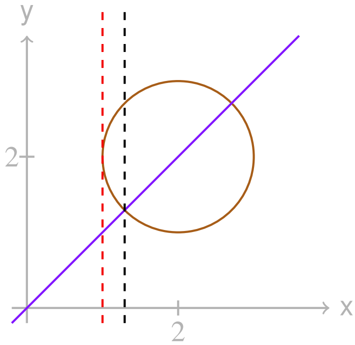
Delineability on an example



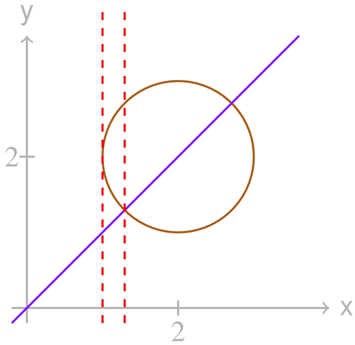
Delineability on an example



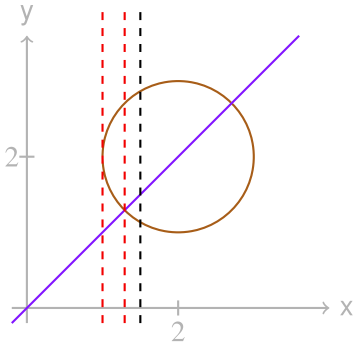
Delineability on an example



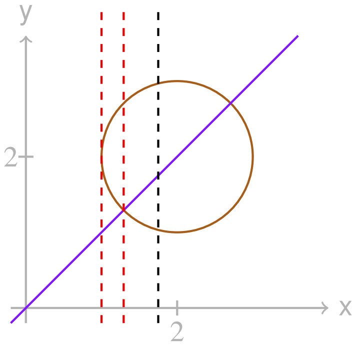
Delineability on an example



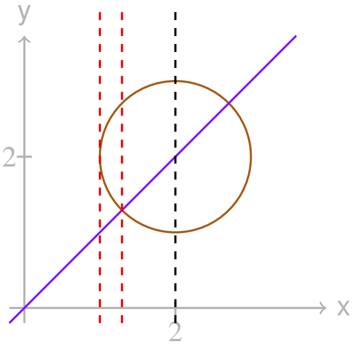
Delineability on an example



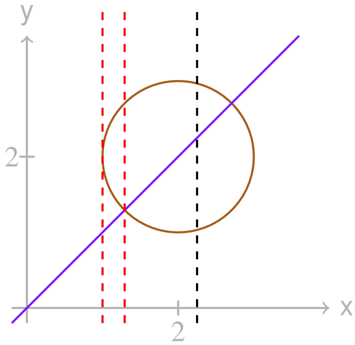
Delineability on an example



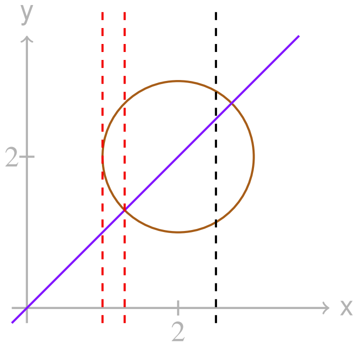
Delineability on an example



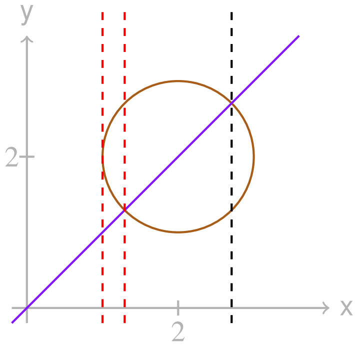
Delineability on an example



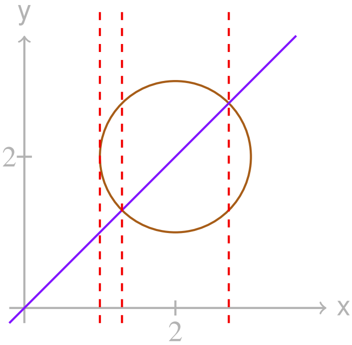
Delineability on an example



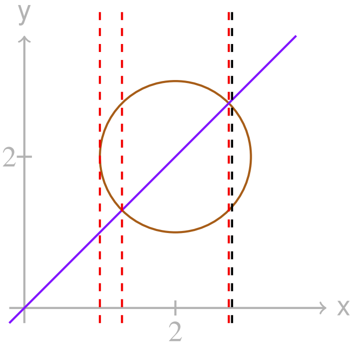
Delineability on an example



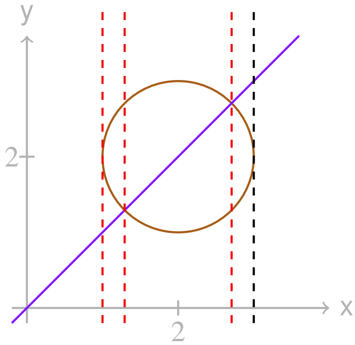
Delineability on an example



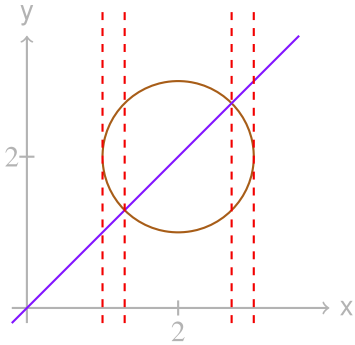
Delineability on an example



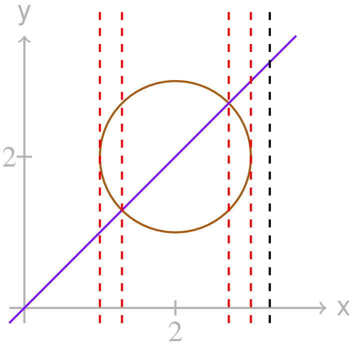
Delineability on an example



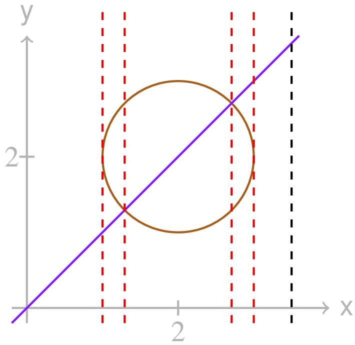
Delineability on an example



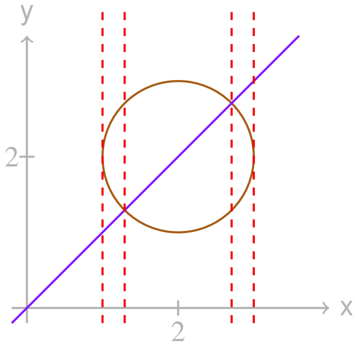
Delineability on an example



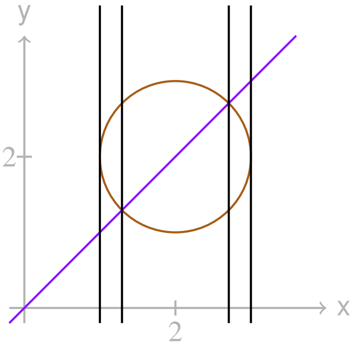
Delineability on an example



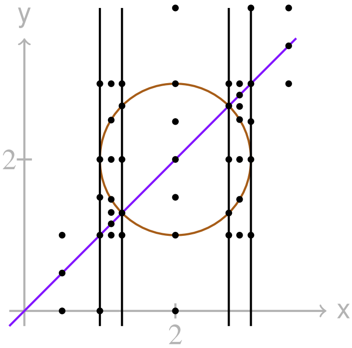
Delineability on an example



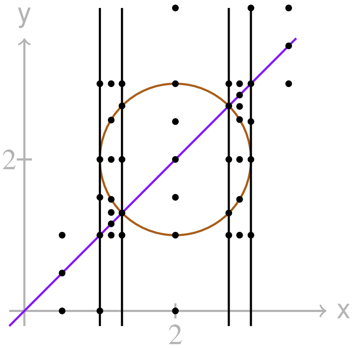
Delineability on an example



Delineability on an example

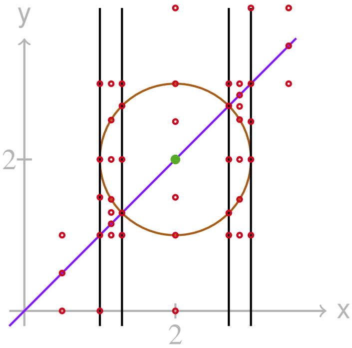


Delineability on an example



$$C = \{\ p_1 < 0,\ p_2 = 0\ \}$$

Delineability on an example

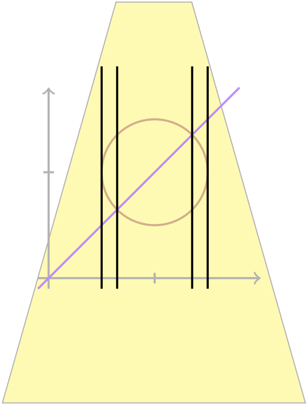


$$C = \{\, p_1 < 0,\ p_2 = 0 \,\}$$

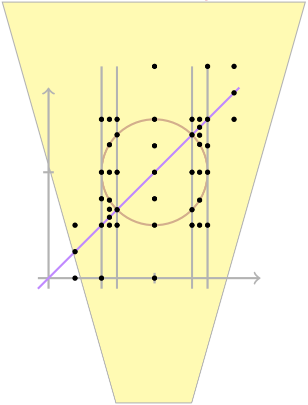
Cylindrical algebraic decomposition (CAD)

A CAD for a set P of polynomials from $\mathbb{Z}[x_1, \ldots, x_n]$ splits $\mathbb{R}^n$ into a finite number of P-sign-invariant regions.

Projection phase



Construction phase



Cylindrical algebraic decomposition (CAD)

A CAD for a set P of polynomials from $\mathbb{Z}[x_1, \ldots, x_n]$ splits $\mathbb{R}^n$ into a finite number of P-sign-invariant regions.

Projection phase

Polynomials $P = P_n \subseteq \mathbb{Z}[x_1, \ldots, x_n]$

↓

Polynomials $P_{n-1} \subseteq \mathbb{Z}[x_1, \ldots, x_{n-1}]$

↓

$\ldots$

Polynomials $P_2 \subseteq \mathbb{Z}[x_1, x_2]$

↓

Polynomials $P_1 \subseteq \mathbb{Z}[x_1]$ ⟶

Construction phase

Samples in $\mathbb{R}^n$

↑

Samples in $\mathbb{R}^{n-1}$

↑

$\ldots$

Samples in $\mathbb{R}^2$

↑

Samples in $\mathbb{R}^1$

Cylindrical algebraic decomposition example

$P_2 = \{(x-2)^2 + (y-2)^2 - 1, x - y\}$

Cylindrical algebraic decomposition example

$$P_2 = \{(x-2)^2 + (y-2)^2 - 1, x - y\}$$

projection

$$P_1 = \{2x^2 - 8x + 7, x^2 - 4x + 3, \dots\}$$

Cylindrical algebraic decomposition example

$$P_2 = \{(x-2)^2 + (y-2)^2 - 1,\ x - y\}$$

projection

$$P_1 = \{2x^2 - 8x + 7,\ x^2 - 4x + 3, \ldots\}$$

zeros of P_1

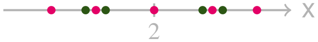
Cylindrical algebraic decomposition example

$P_2 = \{(x-2)^2 + (y-2)^2 - 1, x - y\}$

projection

$P_1 = \{2x^2 - 8x + 7, x^2 - 4x + 3, \ldots\}$

zeros of P_1



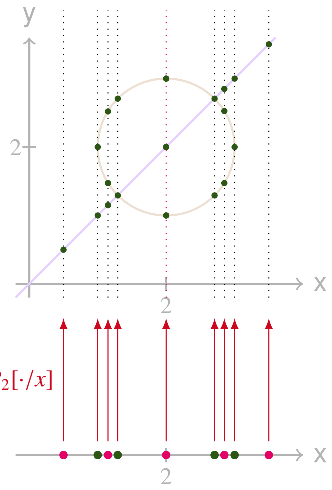
Cylindrical algebraic decomposition example

$$P_2 = \{(x-2)^2 + (y-2)^2 - 1, x - y\}$$

projection

$$P_1 = \{2x^2 - 8x + 7, x^2 - 4x + 3, \dots\}$$

zeros of P_1



Cylindrical algebraic decomposition example

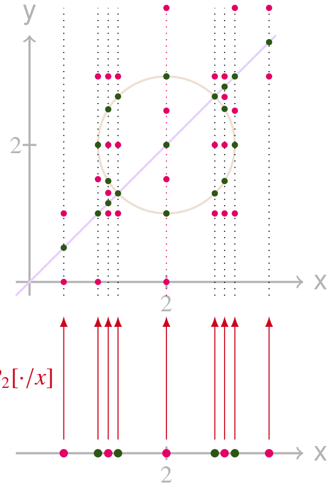
$$P_2 = \{(x-2)^2 + (y-2)^2 - 1, x - y\}$$

projection

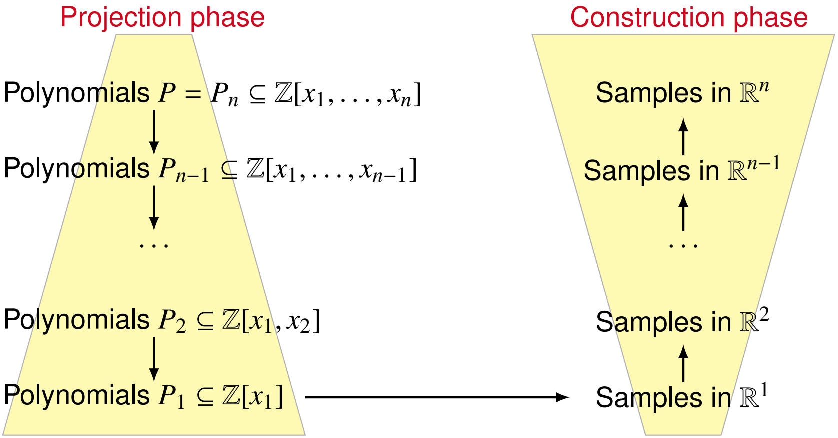
$$P_1 = \{2x^2 - 8x + 7, x^2 - 4x + 3, \dots\}$$

zeros of P_1

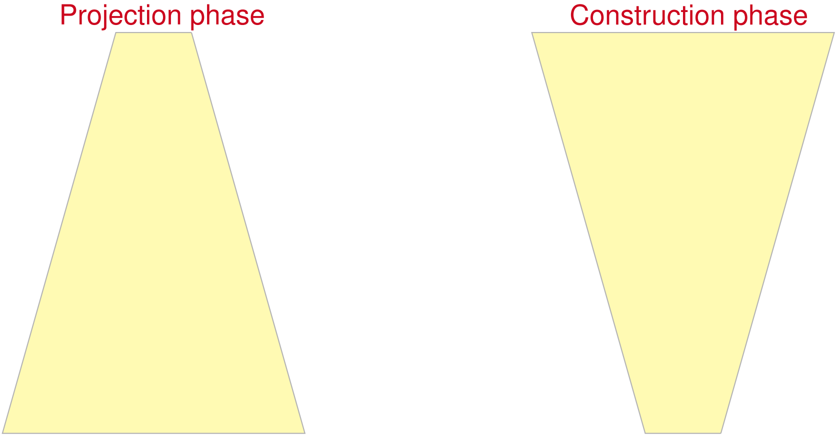
zeros of $P_2[\cdot/x]$



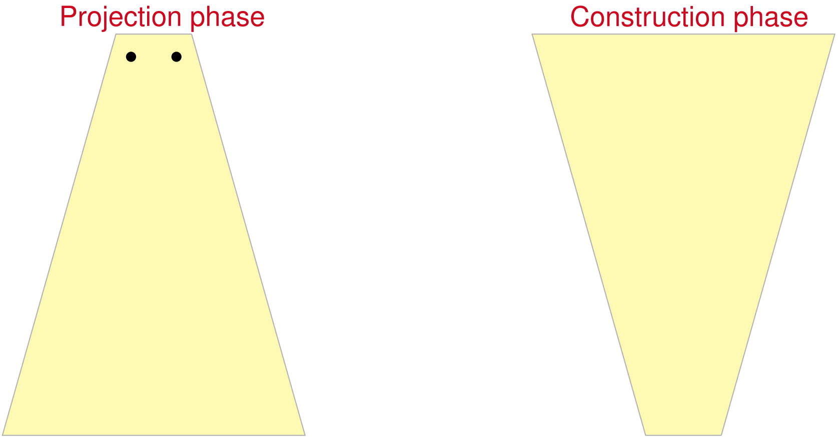
Incrementality and backtracking



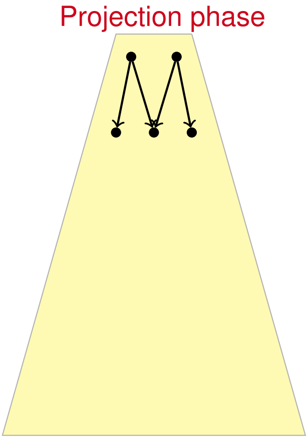
Incrementality and backtracking

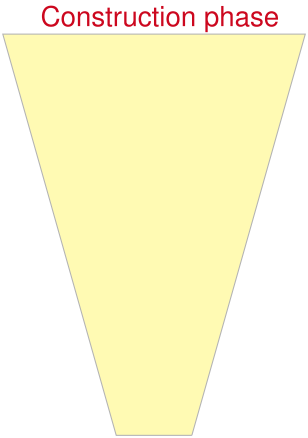


Incrementality and backtracking



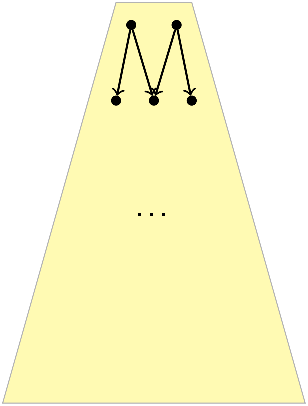
Incrementality and backtracking



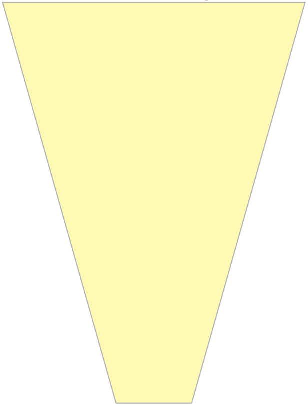


Incrementality and backtracking

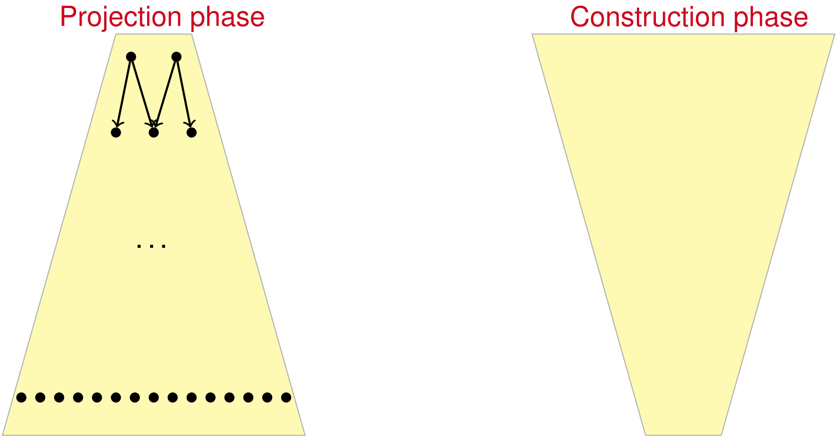
Projection phase



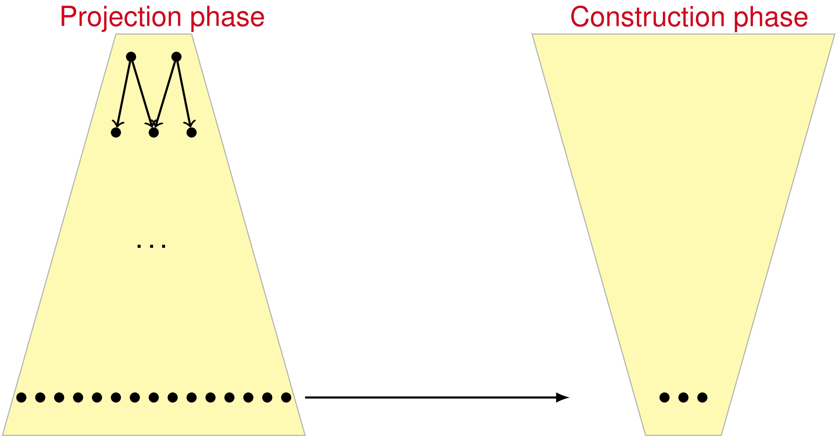
Construction phase



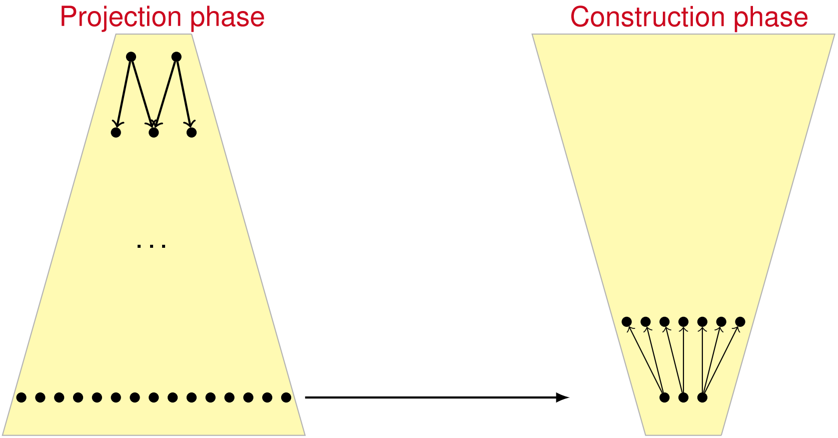
Incrementality and backtracking



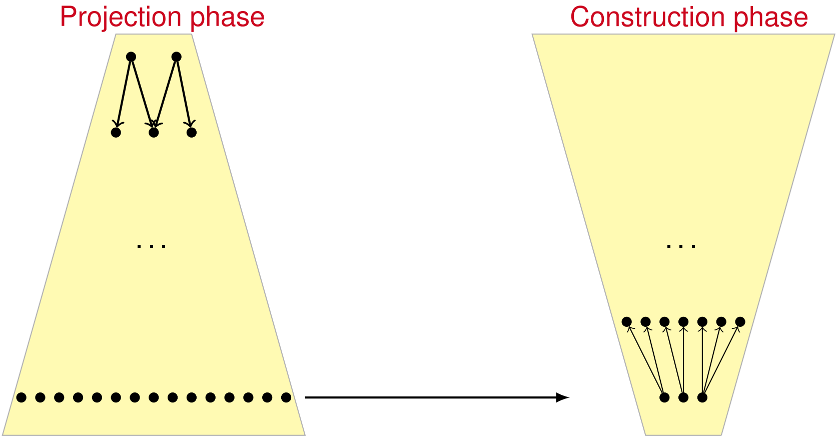
Incrementality and backtracking



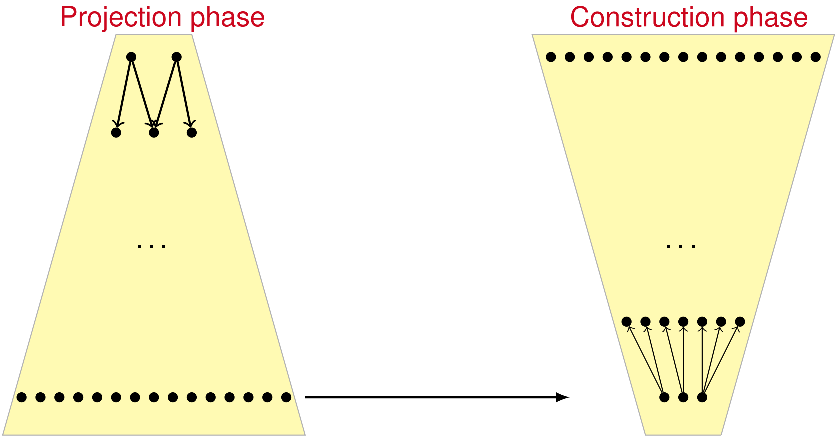
Incrementality and backtracking



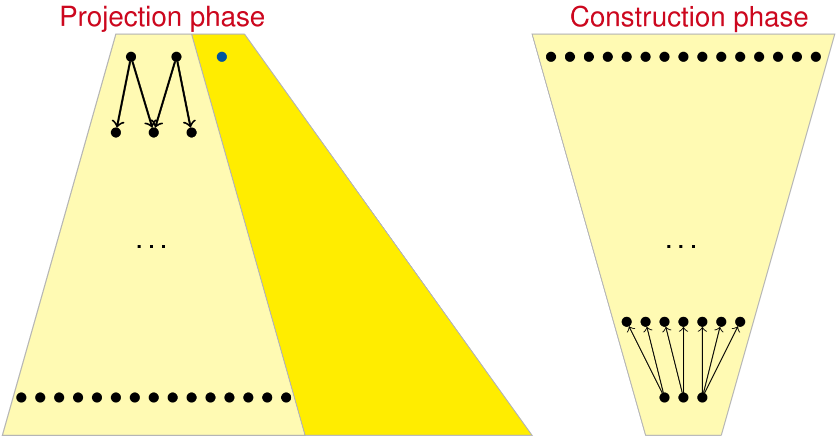
Incrementality and backtracking



Incrementality and backtracking



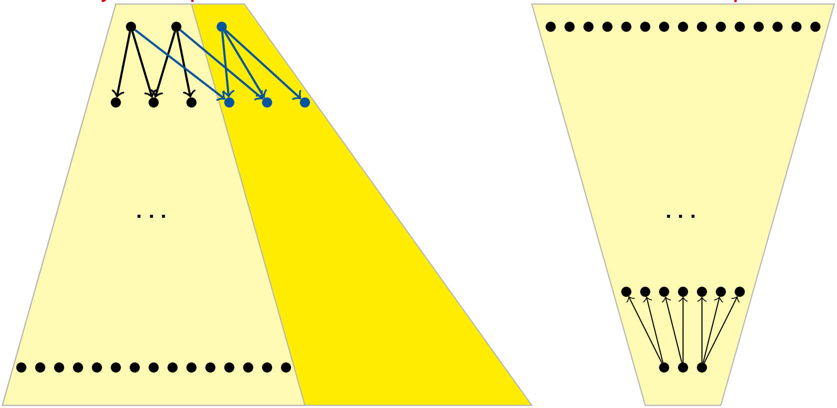
Incrementality and backtracking



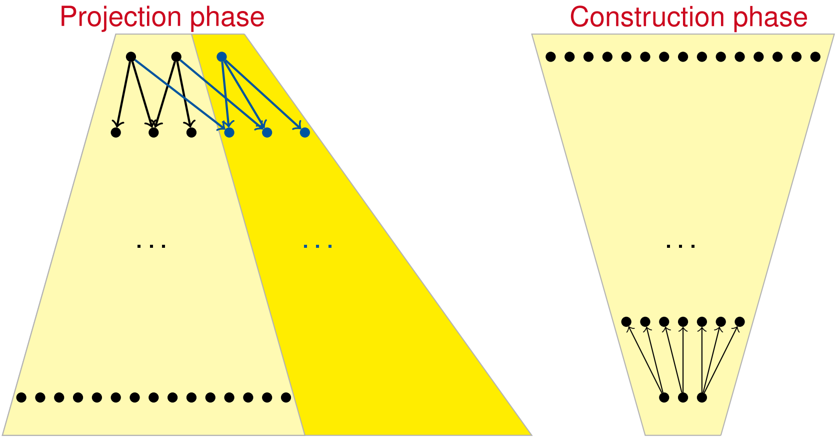
Incrementality and backtracking

Projection phase

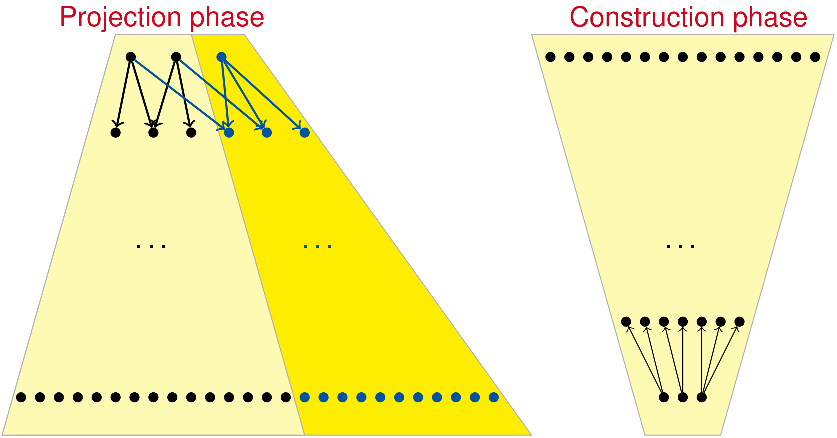
Construction phase



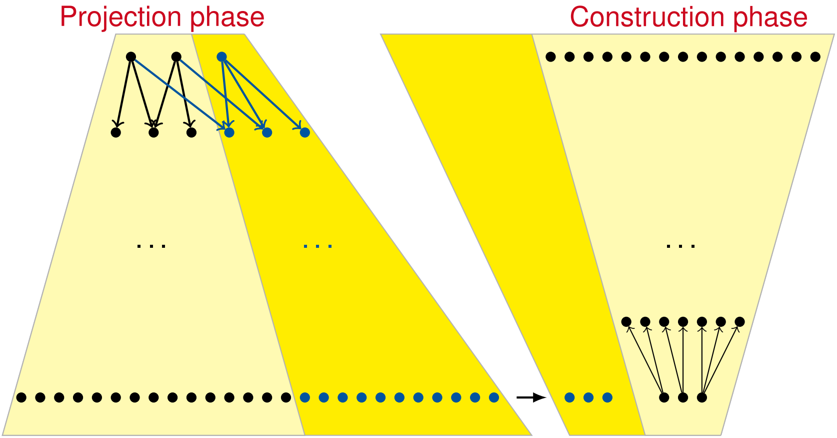
Incrementality and backtracking



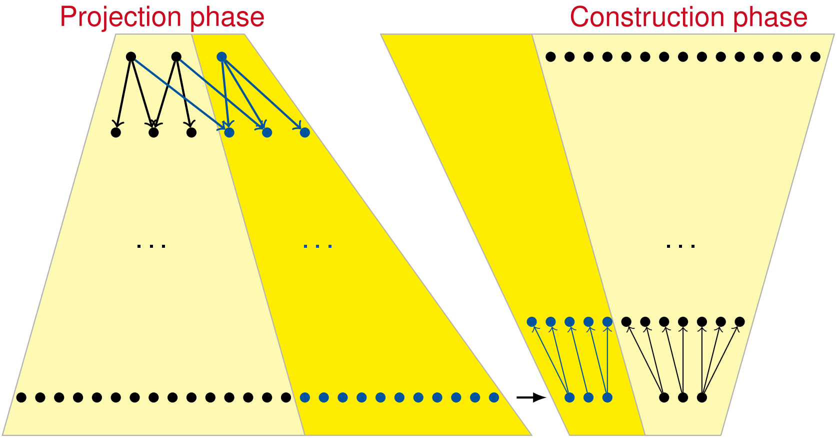
Incrementality and backtracking



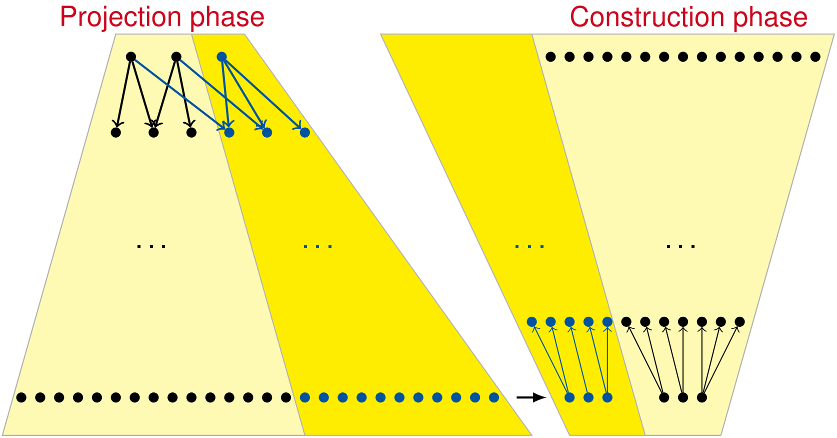
Incrementality and backtracking



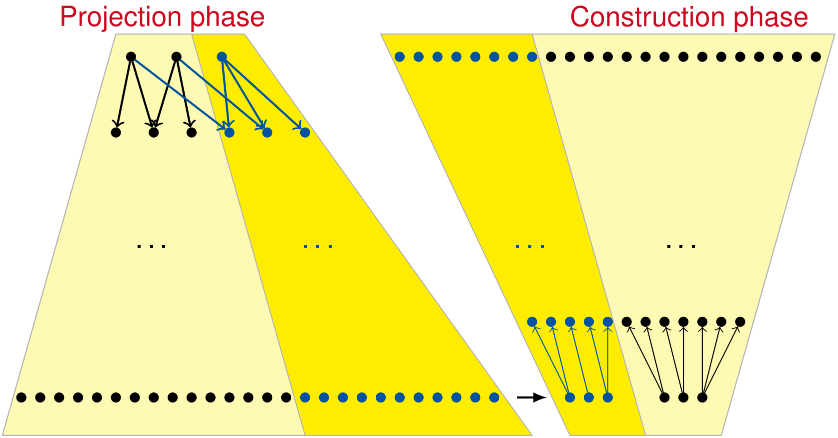
Incrementality and backtracking



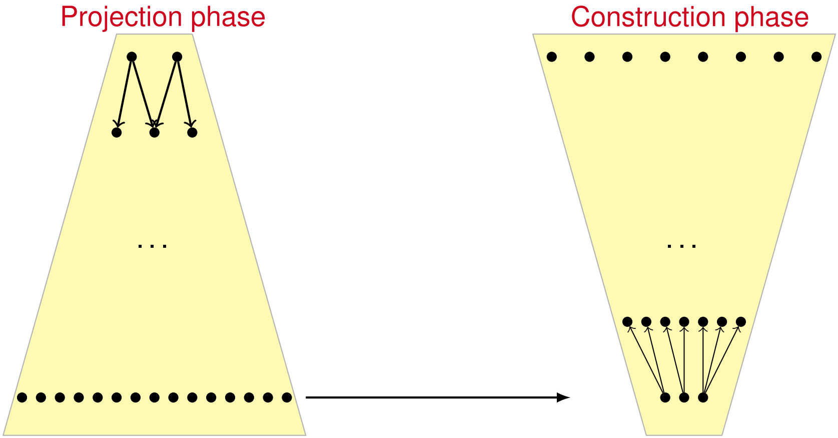
Incrementality and backtracking



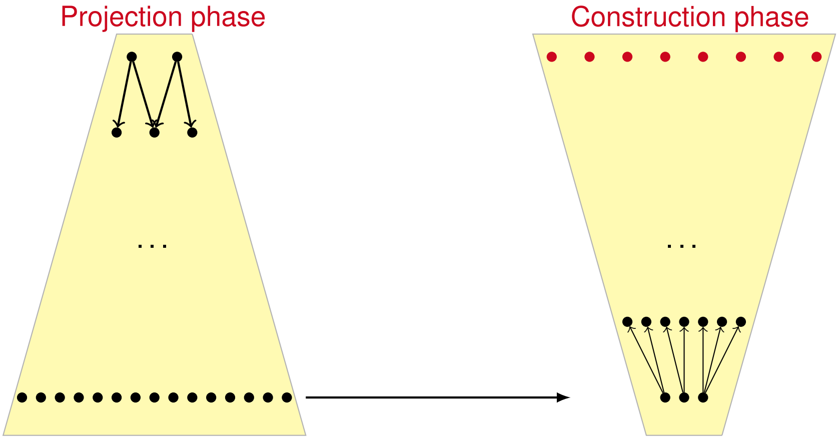
Incrementality and backtracking



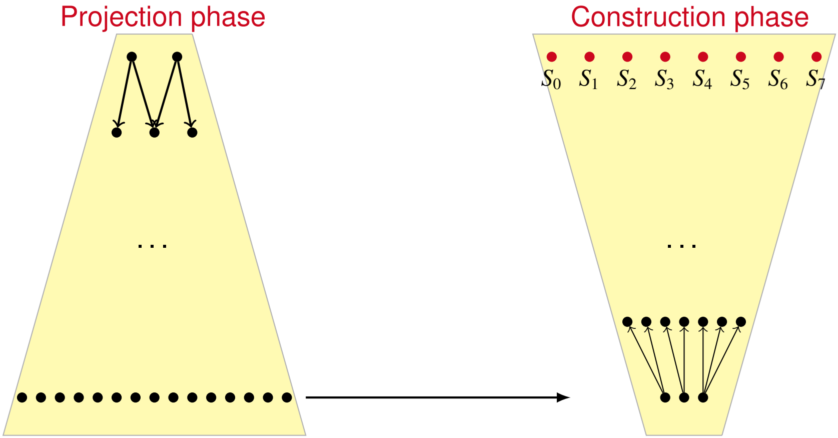
Explanations



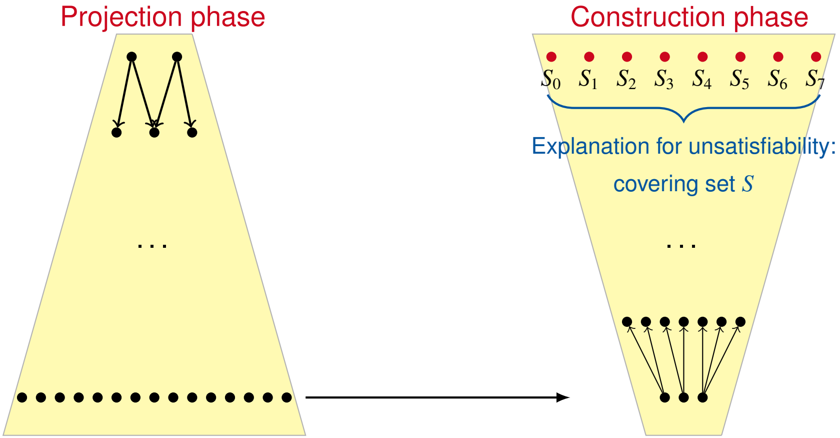
Explanations



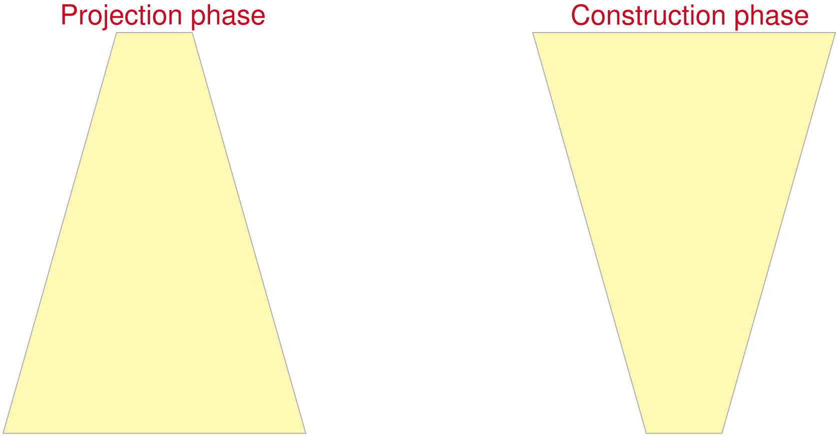
Explanations



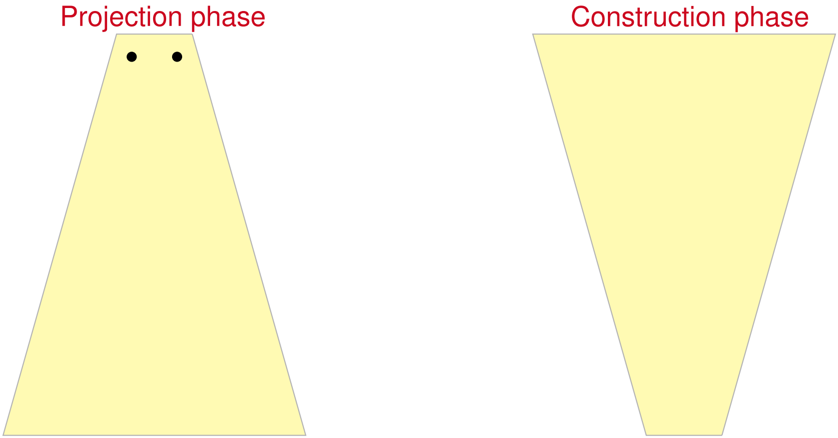
Explanations



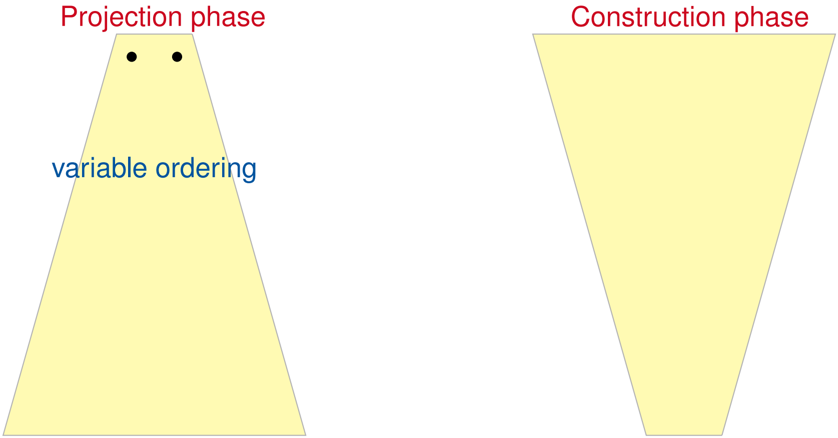
Heuristics



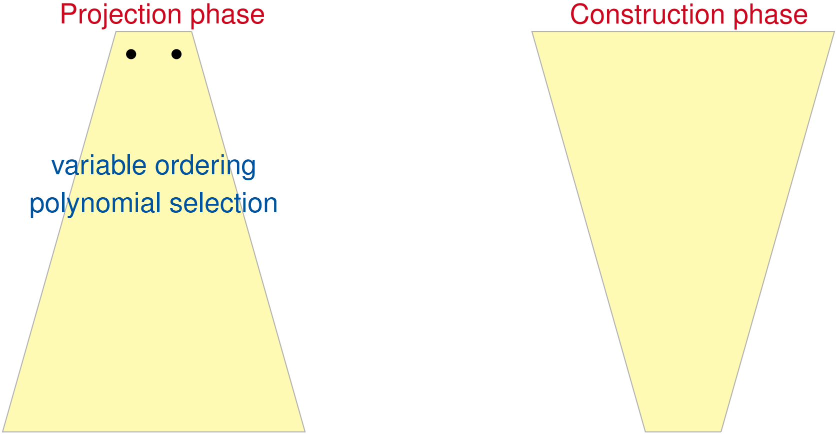
Heuristics



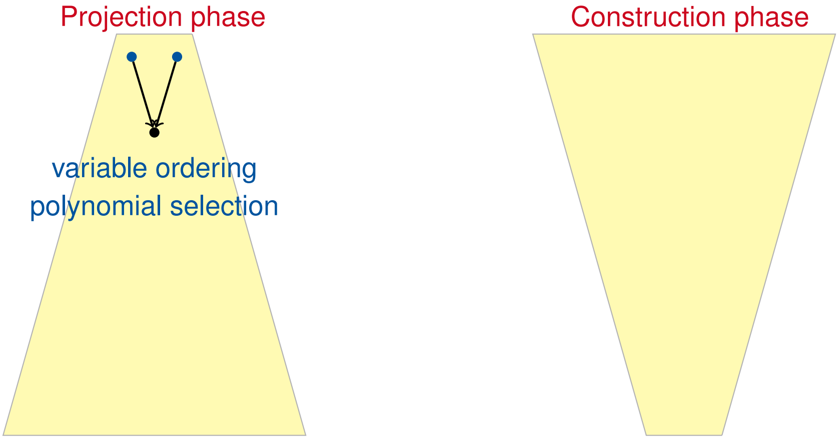
Heuristics



Heuristics

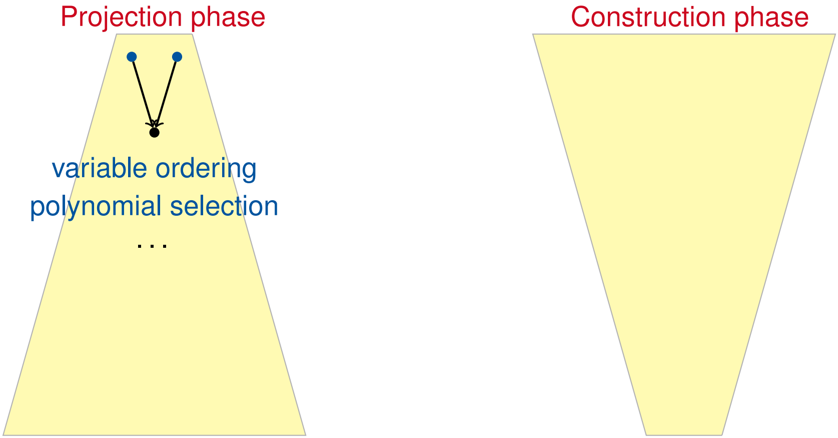

Projection phase
variable ordering
polynomial selection
Construction phase

Heuristics

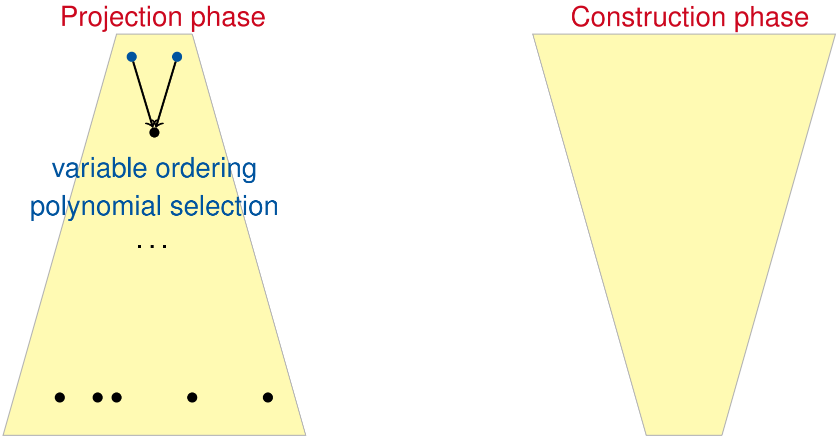


Projection phase
variable ordering
polynomial selection
Construction phase

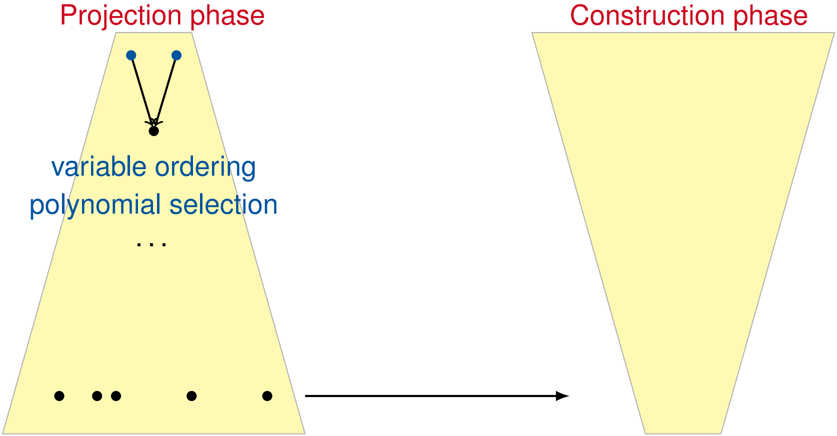
Heuristics


Projection phase
variable ordering
polynomial selection
...
Construction phase

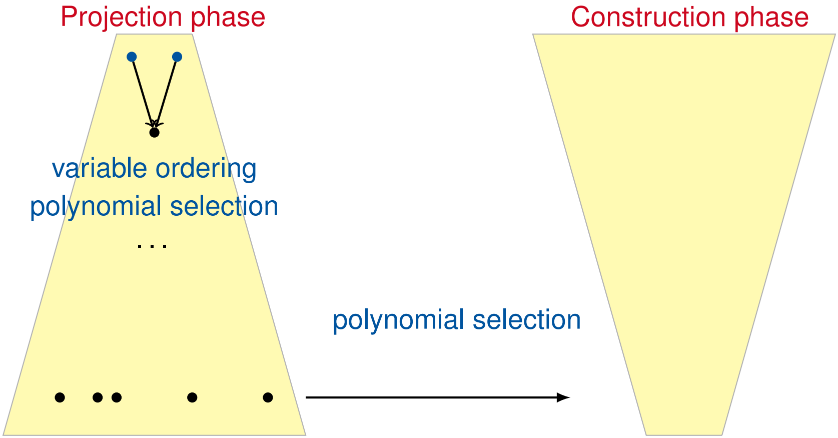
Heuristics


Projection phase
variable ordering
polynomial selection
...
Construction phase

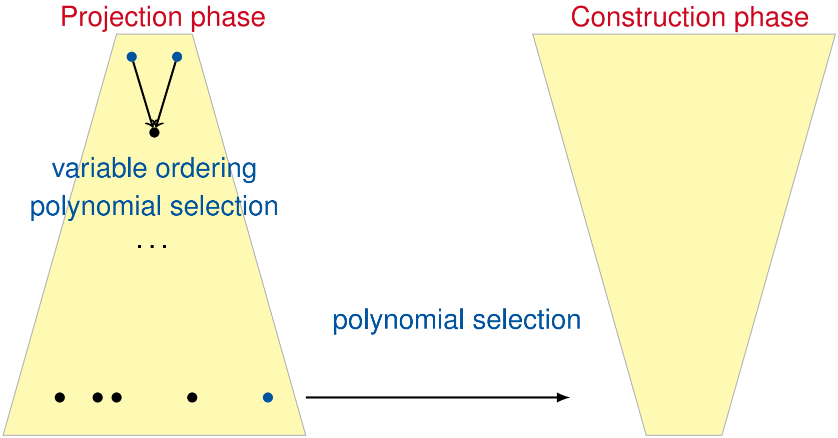
Heuristics


Projection phase
Construction phase
variable ordering
polynomial selection
...

Heuristics

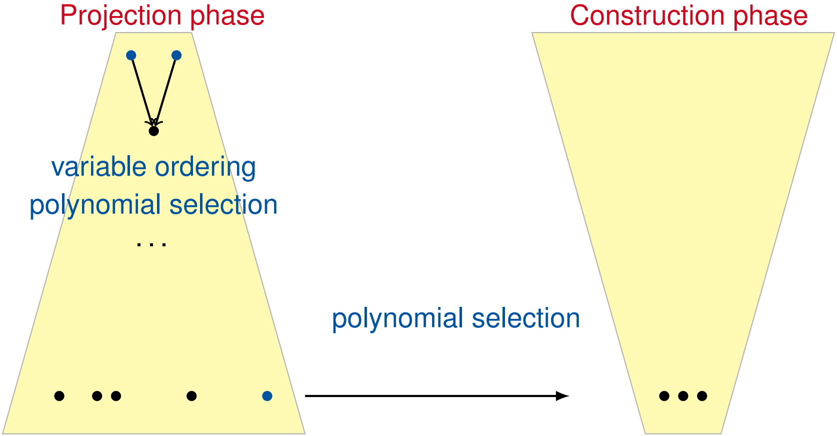

Projection phase
Construction phase
variable ordering
polynomial selection
...
polynomial selection

Heuristics

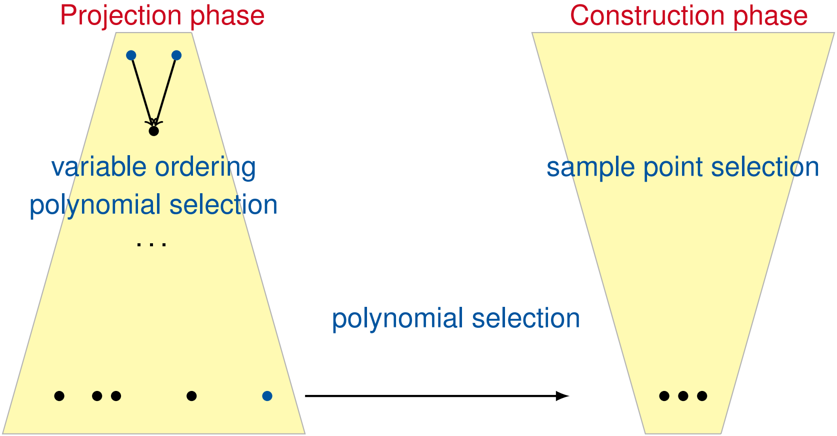


Projection phase
Construction phase
variable ordering
polynomial selection
. . .
polynomial selection

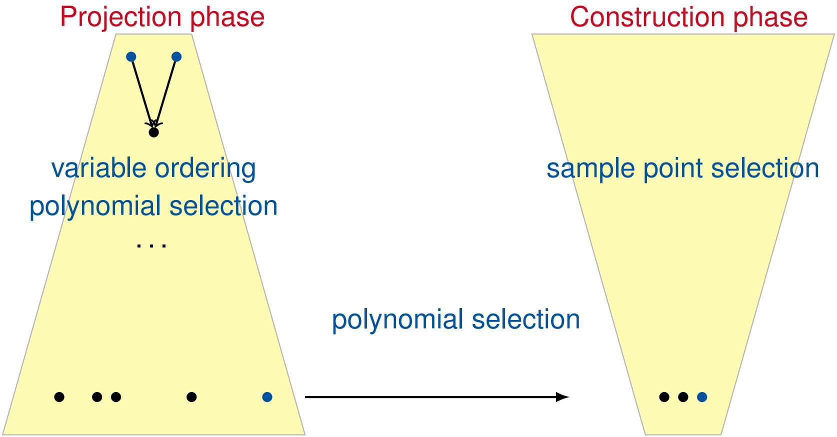
Heuristics


Projection phase
Construction phase
variable ordering
polynomial selection
...
polynomial selection

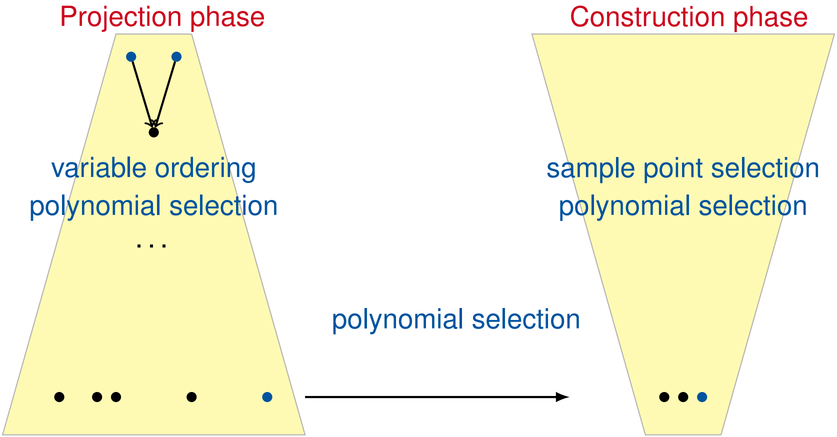
Heuristics


Projection phase
variable ordering
polynomial selection
...
Construction phase
sample point selection
polynomial selection

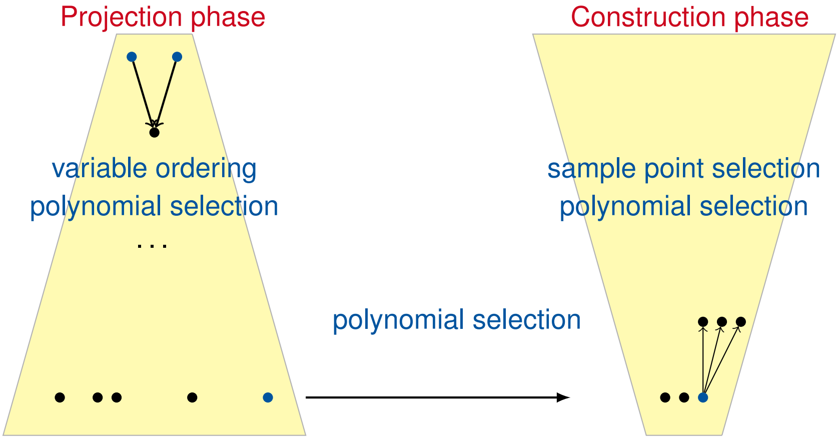
Heuristics


Projection phase
Construction phase
variable ordering
polynomial selection
...
sample point selection
polynomial selection

Heuristics

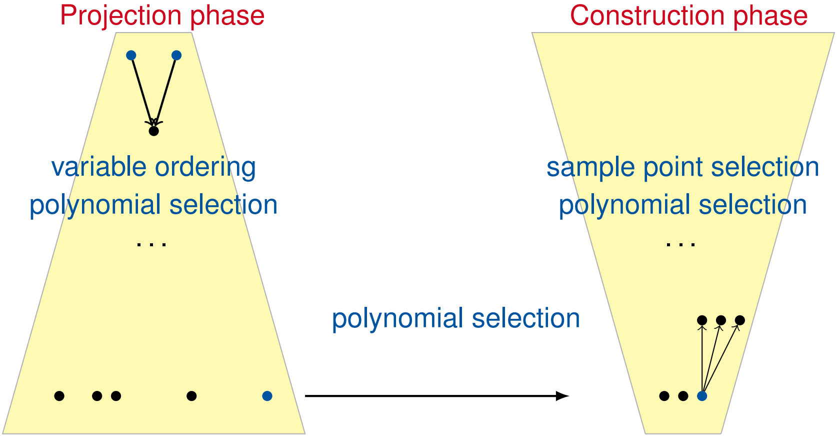

Projection phase
Construction phase
variable ordering
polynomial selection
...
sample point selection
polynomial selection
polynomial selection

Heuristics



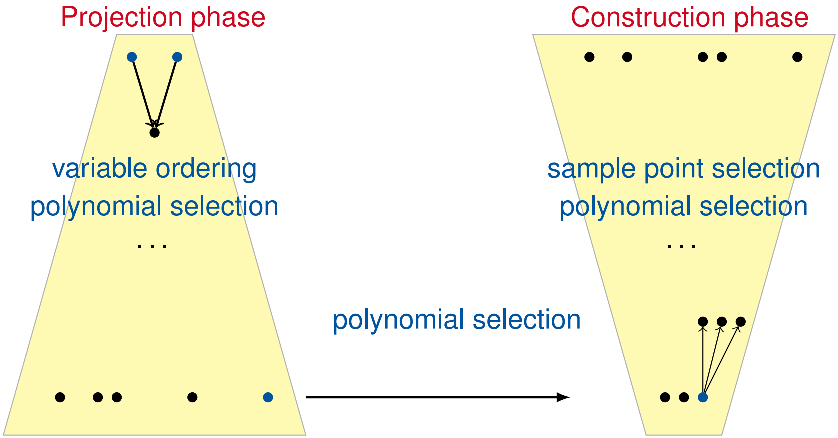
Projection phase
Construction phase
variable ordering
polynomial selection
...
sample point selection
polynomial selection
polynomial selection

Heuristics



Projection phase
variable ordering
polynomial selection
...
polynomial selection
Construction phase
sample point selection
polynomial selection
...

Heuristics


Projection phase
variable ordering
polynomial selection
...
polynomial selection
Construction phase
sample point selection
polynomial selection
...

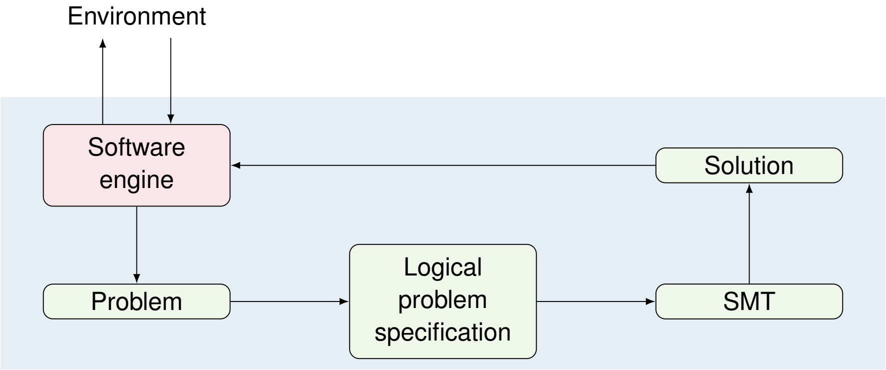
Some SMT-COMP 2016 results

Solver	QF_NRA sequential (10245)			QF_NIA sequential (8593)		
	Correctly solved	Total time	Time per instance	Correctly solved	Total time	Time per instance
AProVE	-	-	-	8273	8527.66	1.03
CVC4	2694	150.24	0.05	8231	161418.04	19.61
ProB	-	-	-	7557	13586.05	1.79
raSAT 0.3	8431	13576.52	1.61	7544	70228.9	9.31
raSAT 0.4	9024	11176.39	1.23	8017	159247.55	19.86
SMT-RAT	9026	51053.15	5.65	8443	6234.5	0.73
Yices	10019	61989.88	6.18	8451	8523.4	1.00
[Z3]	10056	24785.38	2.46	8566	27718.2	3.23

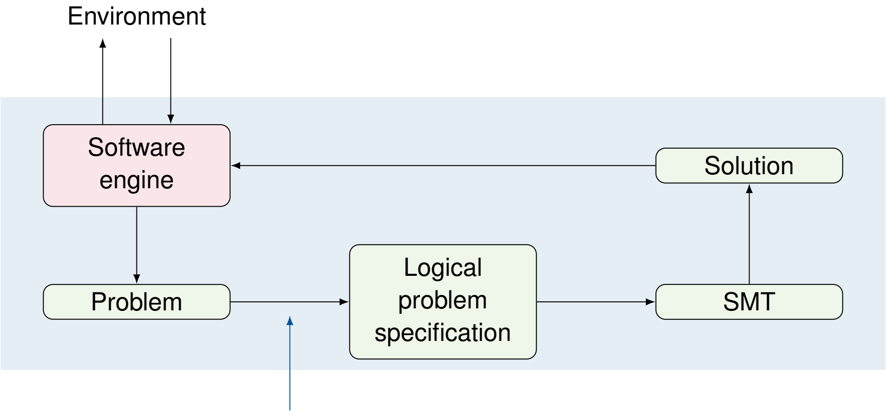
SMT applications

- model checking
- termination analysis
- runtime verification
- test case generation
- controller synthesis
- predicate abstraction
- equivalence checking
- scheduling
- planning
- deployment optimisation on the cloud
- product design automation
- ...

SMT embedding structure



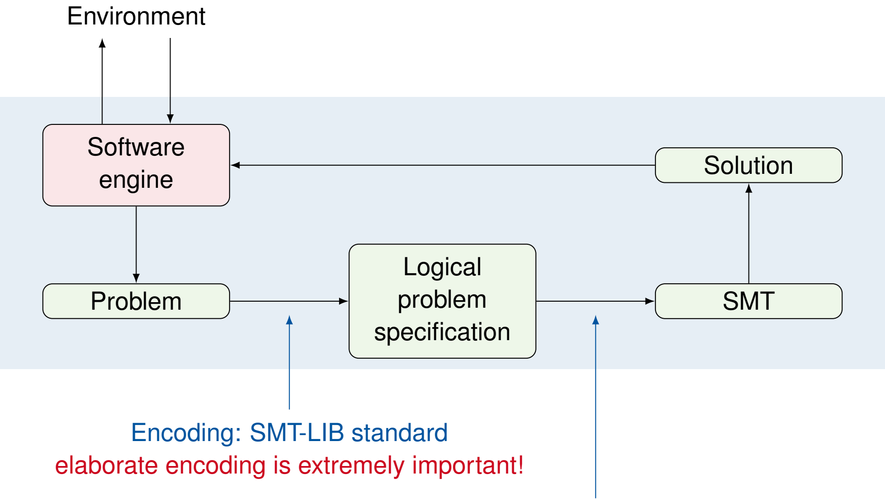
SMT embedding structure



Encoding: SMT-LIB standard

elaborate encoding is extremely important!

SMT embedding structure



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

SMT-LIB syntax → free solver choice

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++

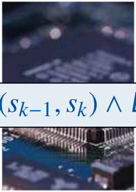
Carnegie Mellon

CBMC Homepage

Bounded Model Checking for Software

Logical encoding of finite unsafe paths

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

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.

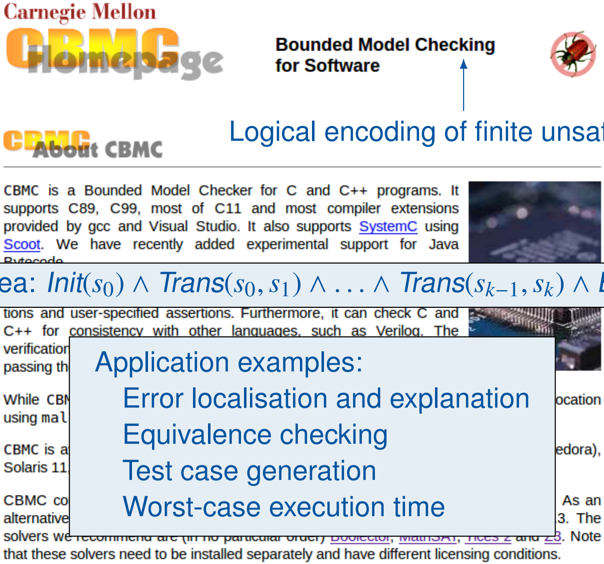
While CBMC is aimed for embedded software, it also supports dynamic memory allocation using `malloc` and `new`. For questions about CBMC, contact Daniel Kroening.

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++



Logical encoding of finite unsafe paths

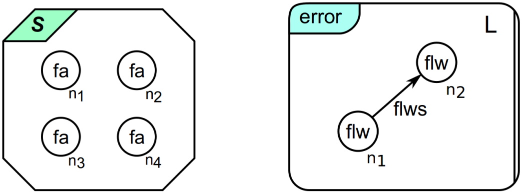
Encoding idea: $Init(s_0) \wedge Trans(s_0, s_1) \wedge \ldots \wedge Trans(s_{k-1}, s_k) \wedge Bad(s_0, \ldots, s_k)$

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/`

BMC for graph transformation systems



(a) Initial graph S (b) Forbidden pattern

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

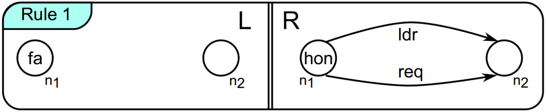


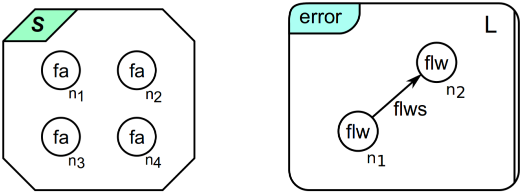
Fig. 2. Rule 1 of the car platooning GTS [1]

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]

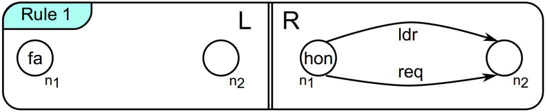


Fig. 2. Rule 1 of the car platooning GTS [1]

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

↓

Apply bounded model checking

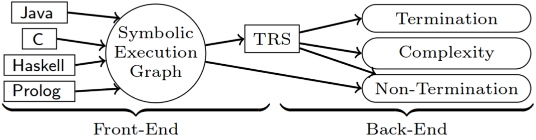
Source: T. Isenberg, D. Steenken, and H. Wehrheim.

Bounded Model Checking of Graph Transformation Systems via SMT Solving.

In Proc. FMOODS/FORTE'13.

Termination analysis for programs

Automated Program Verification Environment



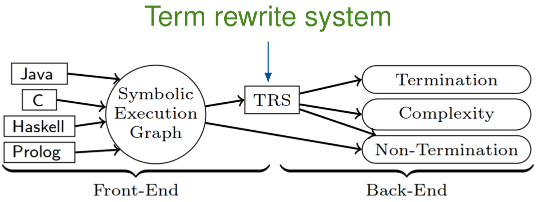
Source: T. Ströder, C. Aschermann, F. Frohn, J. Hensel, J. Giesl.

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

In Proc. TACAS'15.

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).

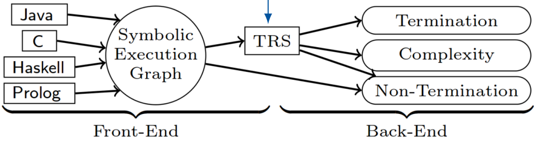
In Proc. TACAS'15.

Termination analysis for programs

AProVE

Automated Program Verification Environment

Term rewrite system



Term rewrite system

↓

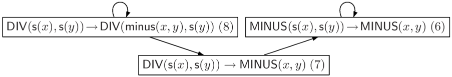
Dependency pairs

↓

Chains

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

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



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

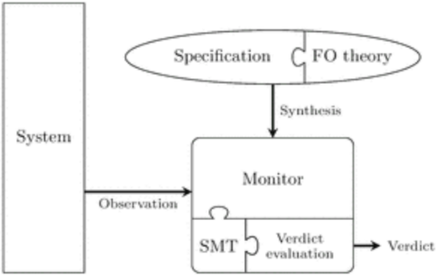


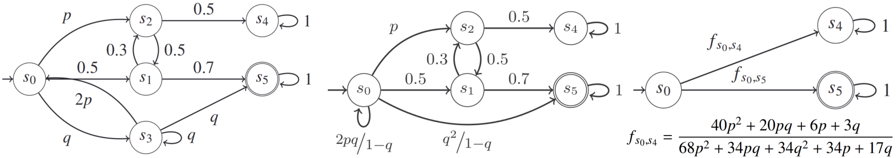
Fig. 1 Schematic overview of the monitoring approach

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

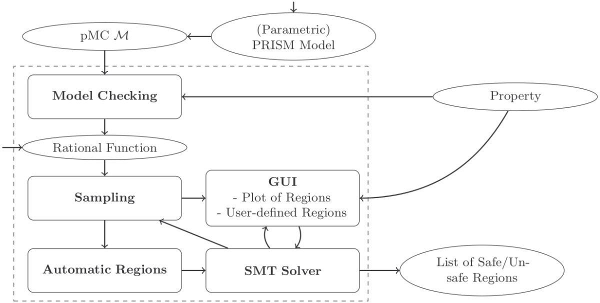
Monitoring modulo theories.

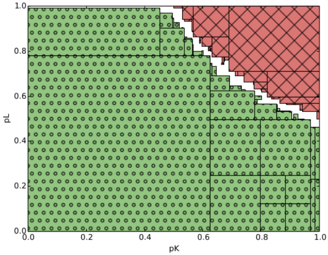
International Journal on Software Tools for Technology Transfer, 18(2):205-225, April 2016.

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.

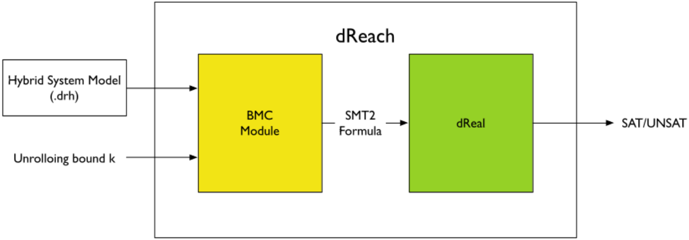
In Proc. of CAV'15.

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/`

Planning

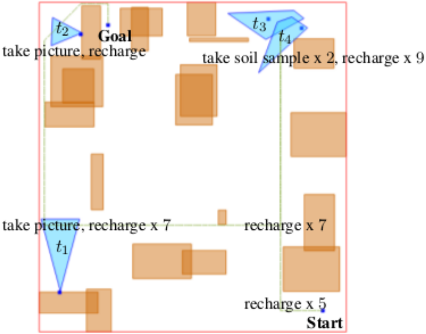


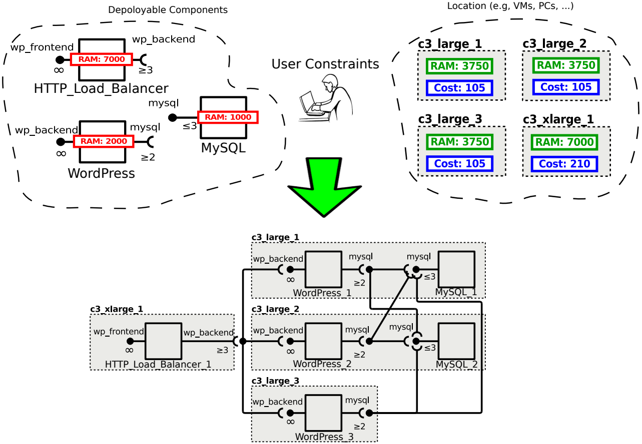
Figure 1: A GEOMETRIC ROVERS example instance, showing the starting and goal locations of the rover, areas where tasks can be performed (blue) and obstacles (orange) and a plan solving the task (green). The red box indicates the bounds of the environment.

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

Numeric planning with disjunctive global constraints via SMT.

In Proc. of ICASP'16.

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.

Scheduling

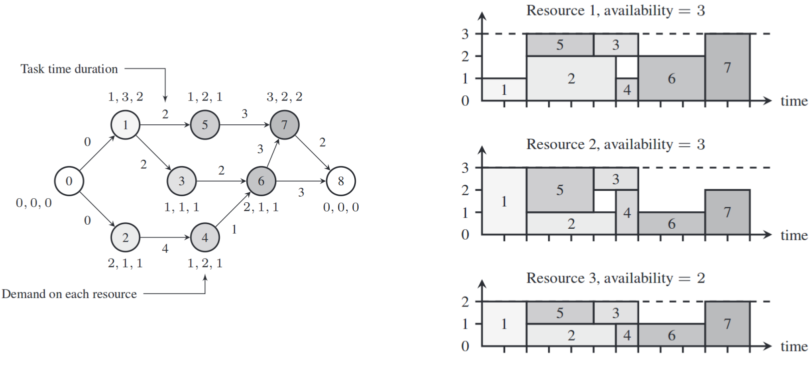


Figure 1: An example of RCPSP (Liess and Michelon 2008)

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.

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