Practical Course: SMT Solving

SMT Solving for Equality Logic and Uninterpreted Functions

Erika Ábrahám, Rebecca Haehn, Gereon Kremer, Stefan Schupp

Winter term 2018/19

Satisfiability Checking

Propositional logic formula in CNF

Let $p_1, \ldots, p_n$ be Boolean variables (or propositions).

$$\varphi = \bigwedge_{i=1}^{m} \underbrace{\bigvee_{j=1}^{m_i} l_{ij}}_{\text{clause}}, \text{ with literals } l_{ij} \text{ with } l_{ij} = p_k \text{ or } l_{ij} = \neg p_k.$$

Satisfiability Checking

Propositional logic formula in CNF

Let $p_1, \ldots, p_n$ be Boolean variables (or propositions).

$$\varphi = \bigwedge_{i=1}^{m} \underbrace{\bigvee_{j=1}^{m_i} l_{ij}}_{\text{clause}}, \text{ with literals } l_{ij} \text{ with } l_{ij} = p_k \text{ or } l_{ij} = \neg p_k.$$

- φ is satisfiable iff an assignment $\alpha : \{p_1, \ldots, p_n\} \to \{\mathit{true}, \ \mathit{false}\}$ exists such that φ evaluates to *true*

Satisfiability Checking

Propositional logic formula in CNF

Let $p_1, \ldots, p_n$ be Boolean variables (or propositions).

$$\varphi = \bigwedge_{i=1}^{m} \underbrace{\bigvee_{j=1}^{m_i} l_{ij}}_{\text{clause}}, \text{ with literals } l_{ij} \text{ with } l_{ij} = p_k \text{ or } l_{ij} = \neg p_k.$$

- φ is satisfiable iff an assignment $\alpha : \{p_1, \ldots, p_n\} \to \{\mathit{true}, \ \mathit{false}\}$ exists such that φ evaluates to *true*
- satisfiability problem (SAT): is a given φ satisfiable?

Satisfiability Checking

Propositional logic formula in CNF

Let $p_1, \ldots, p_n$ be Boolean variables (or propositions).

$$\varphi = \bigwedge_{i=1}^{m} \underbrace{\bigvee_{j=1}^{m_i} l_{ij}}_{\text{clause}}, \text{ with literals } l_{ij} \text{ with } l_{ij} = p_k \text{ or } l_{ij} = \neg p_k.$$

- φ is satisfiable iff an assignment $\alpha : \{p_1, \ldots, p_n\} \to \{\mathit{true}, \ \mathit{false}\}$ exists such that φ evaluates to *true*
- satisfiability problem (SAT): is a given φ satisfiable?
- SAT is NP-complete, but can often be solved quickly

Satisfiability Checking

Propositional logic formula in CNF

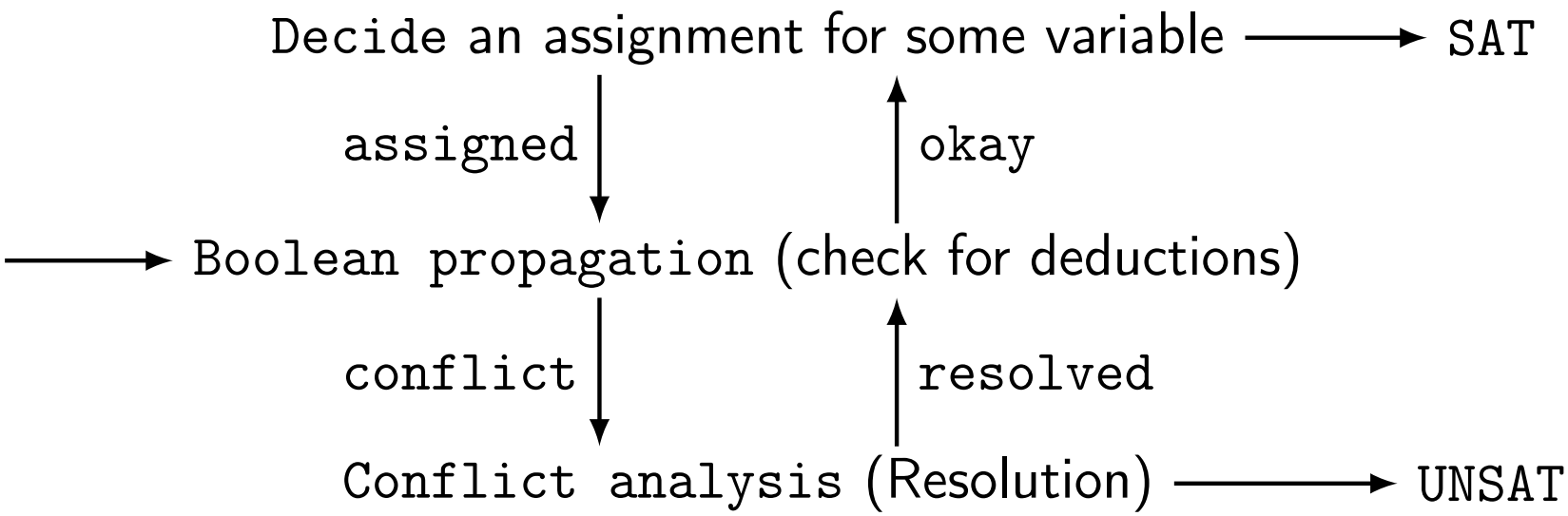
Let $p_1, \ldots, p_n$ be Boolean variables (or propositions).

$$\varphi = \bigwedge_{i=1}^{m} \underbrace{\bigvee_{j=1}^{m_i} l_{ij}}_{\textit{clause}}, \text{ with literals } l_{ij} \text{ with } l_{ij} = p_k \text{ or } l_{ij} = \neg p_k.$$

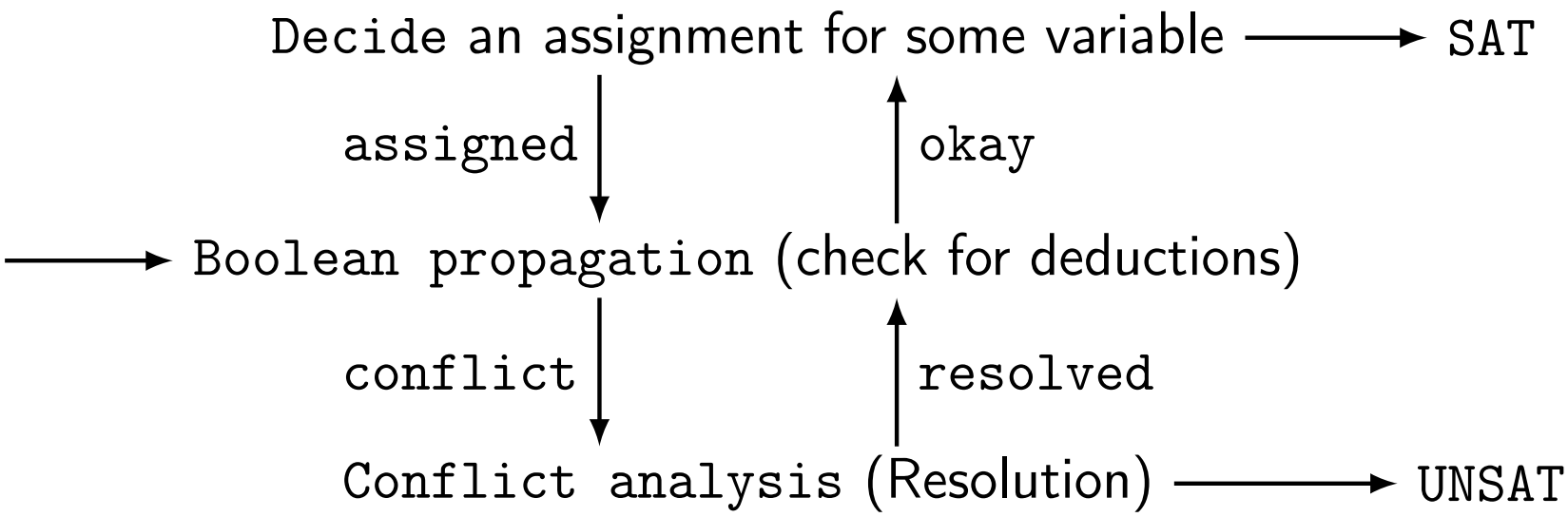
- φ is satisfiable iff an assignment $\alpha : \{p_1, \ldots, p_n\} \to \{\textit{true}, \ \textit{false}\}$ exists such that φ evaluates to *true*
- satisfiability problem (SAT): is a given φ satisfiable?
- SAT is NP-complete, but can often be solved quickly

  $\to$ Davis-Putnam-Logemann-Loveland (DPLL) algorithm
  $\to$ Conflict-Driven-Clause-Learning (CDCL) algorithm

CDCL Algorithm



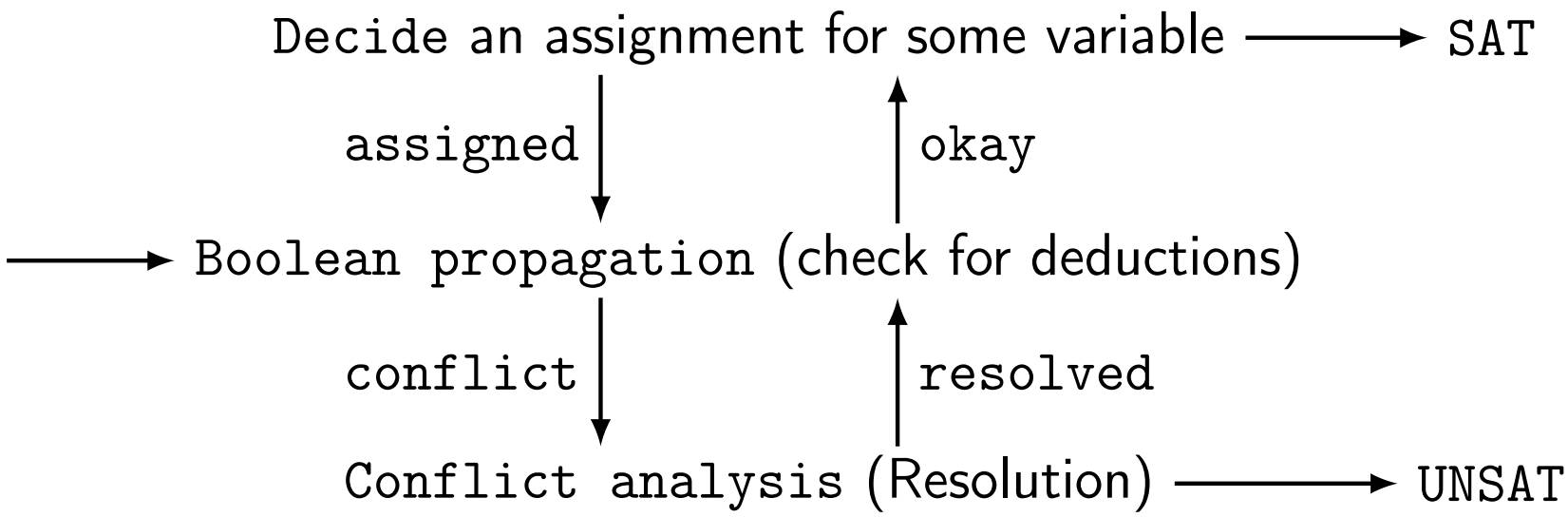
CDCL Algorithm



$$\begin{aligned}\varphi &= c_1 \wedge c_2 \wedge c_3 \wedge c_4 \\ &= p_1 \wedge (p_2 \vee p_3) \wedge (\neg p_3 \vee p_4) \wedge (\neg p_1 \vee p_2 \vee \neg p_4)\end{aligned}$$

$$\alpha = \emptyset$$

CDCL Algorithm

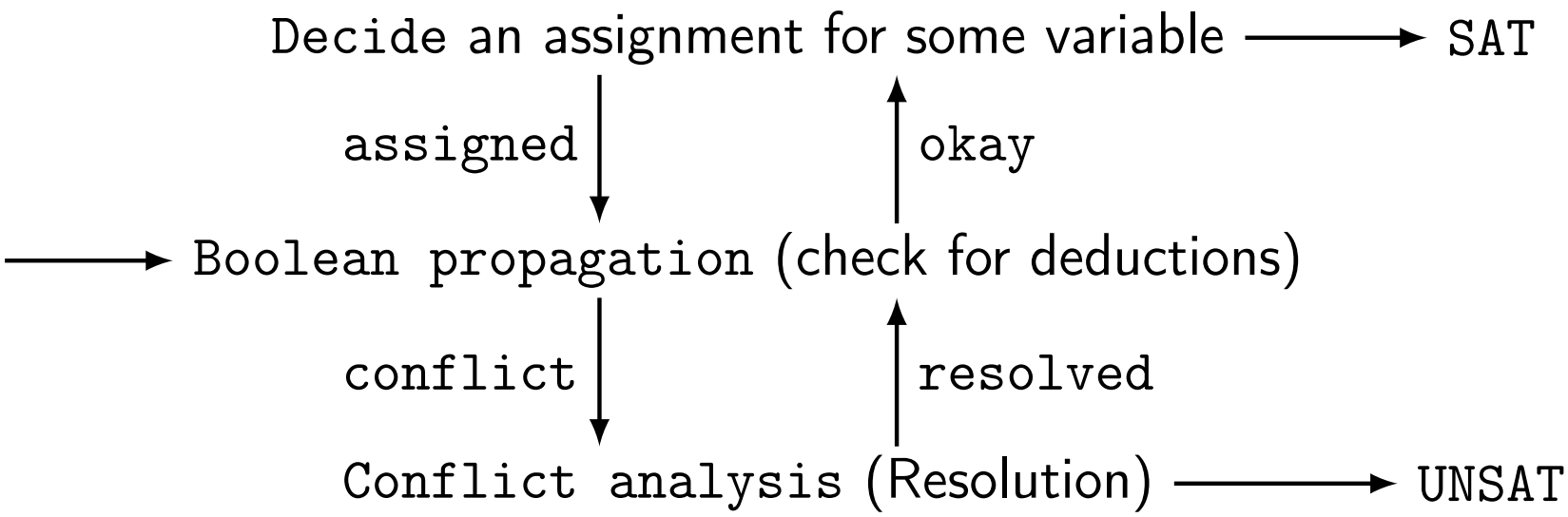


$$\varphi = c_1 \wedge c_2 \wedge c_3 \wedge c_4$$
$$= p_1 \wedge (p_2 \vee p_3) \wedge (\neg p_3 \vee p_4) \wedge (\neg p_1 \vee p_2 \vee \neg p_4)$$

$$\alpha = \{p_1 \mapsto \mathit{true}\}$$

- ▷ Propagate

CDCL Algorithm

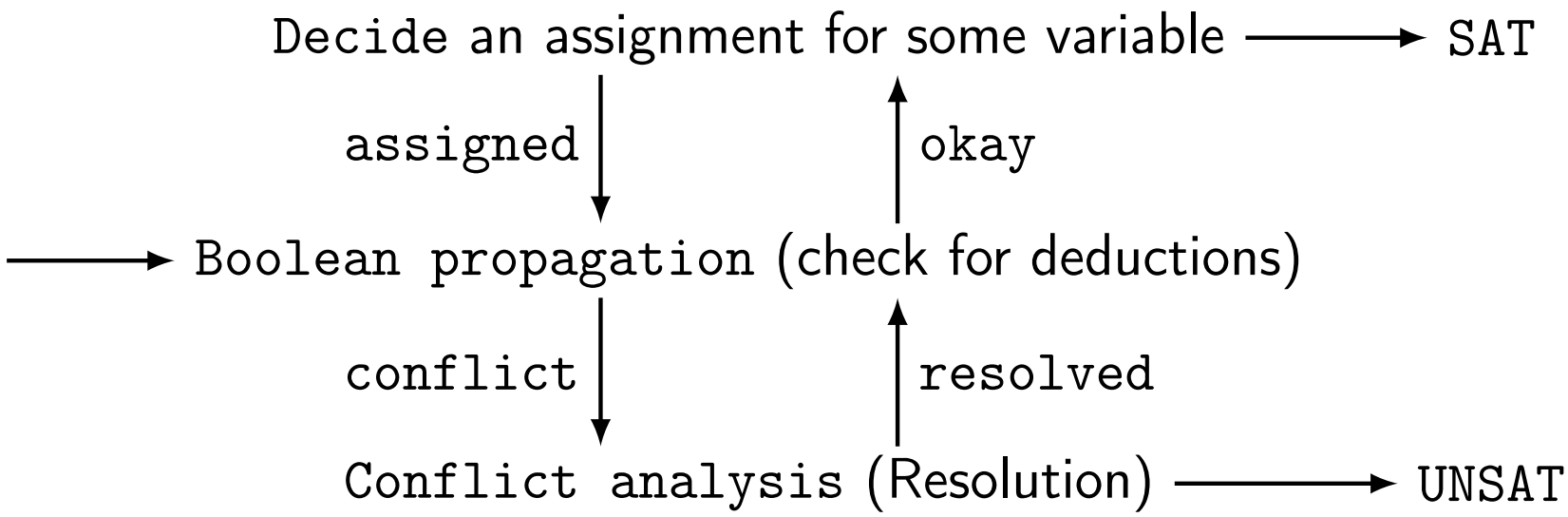


$$\varphi = c_2 \wedge c_3 \wedge c_4$$
$$= (p_2 \vee p_3) \wedge (\neg p_3 \vee p_4) \wedge (p_2 \vee \neg p_4)$$

$$\alpha = \{p_1 \mapsto \mathit{true},\ p_2 \mapsto \mathit{false}\}$$

- ▷ Decide

CDCL Algorithm

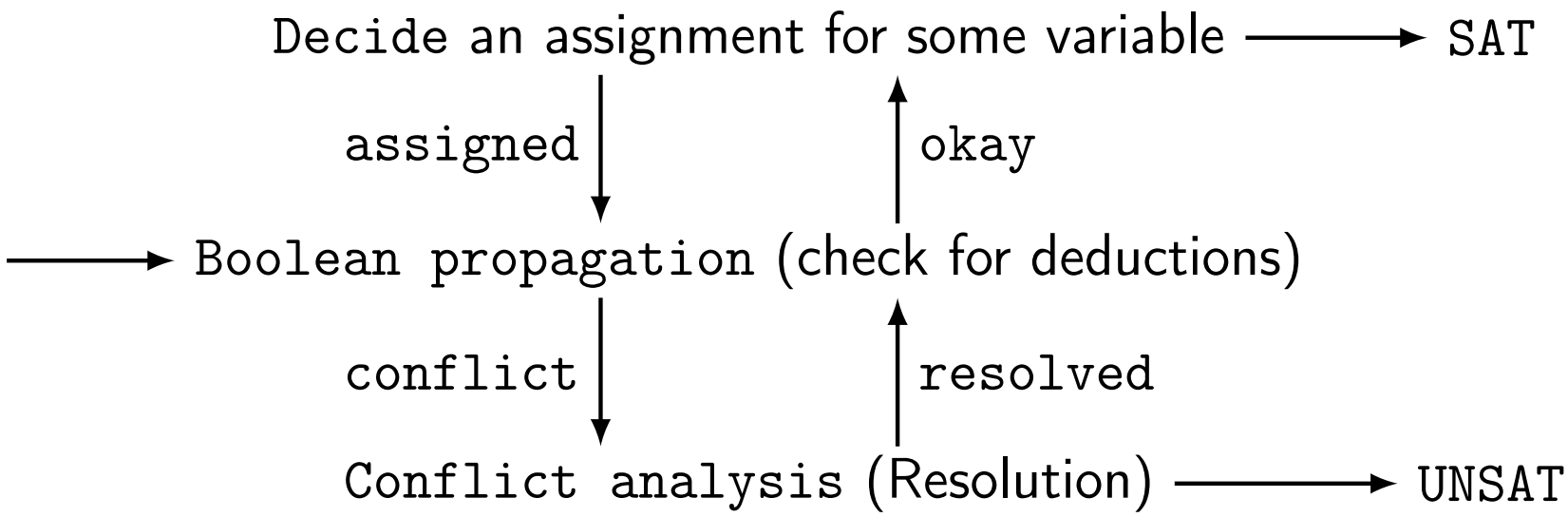


$$\varphi = c_2 \wedge c_3 \wedge c_4$$
$$= (\quad p_3) \wedge (\neg p_3 \vee p_4) \wedge (\quad \neg p_4)$$

$$\alpha = \{p_1 \mapsto \mathit{true},\ p_2 \mapsto \mathit{false},\ p_3 \mapsto \mathit{true}\}$$

- ▷ Propagate

CDCL Algorithm

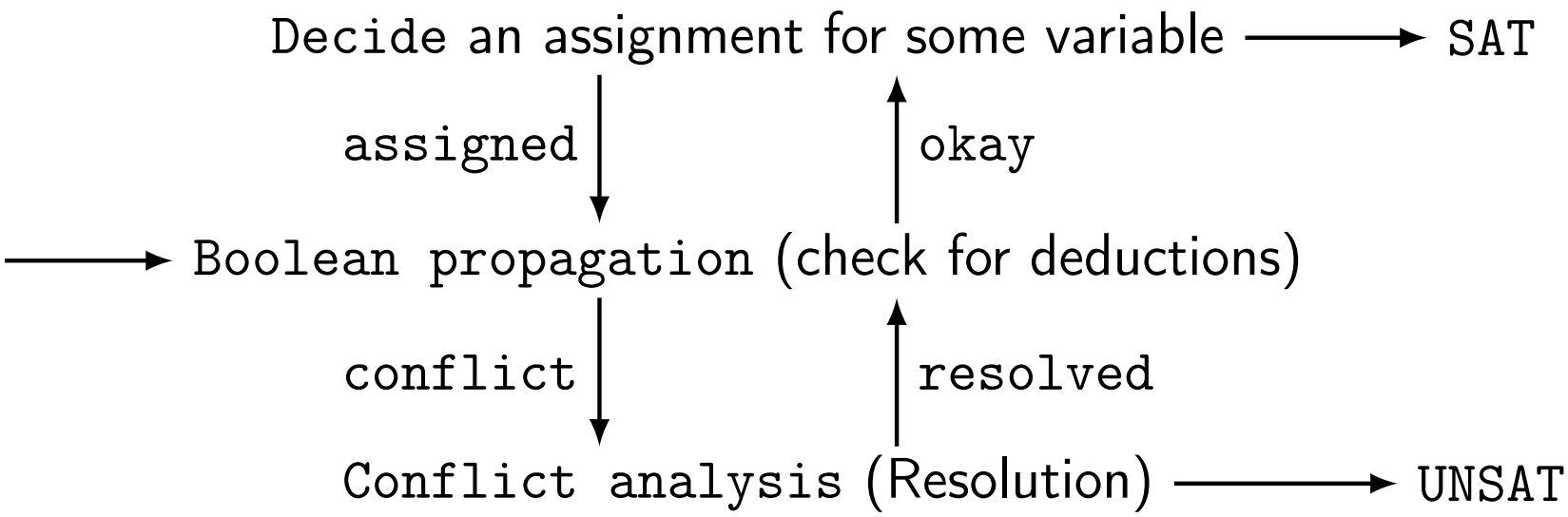


$$\varphi = c_3 \wedge c_4$$
$$= (\quad p_4) \wedge (\quad \neg p_4)$$

$$\alpha = \{p_1 \mapsto \mathit{true},\ p_2 \mapsto \mathit{false},\ p_3 \mapsto \mathit{true},\ p_4 \mapsto \mathit{true}\}$$

- ▷ Propagate

CDCL Algorithm



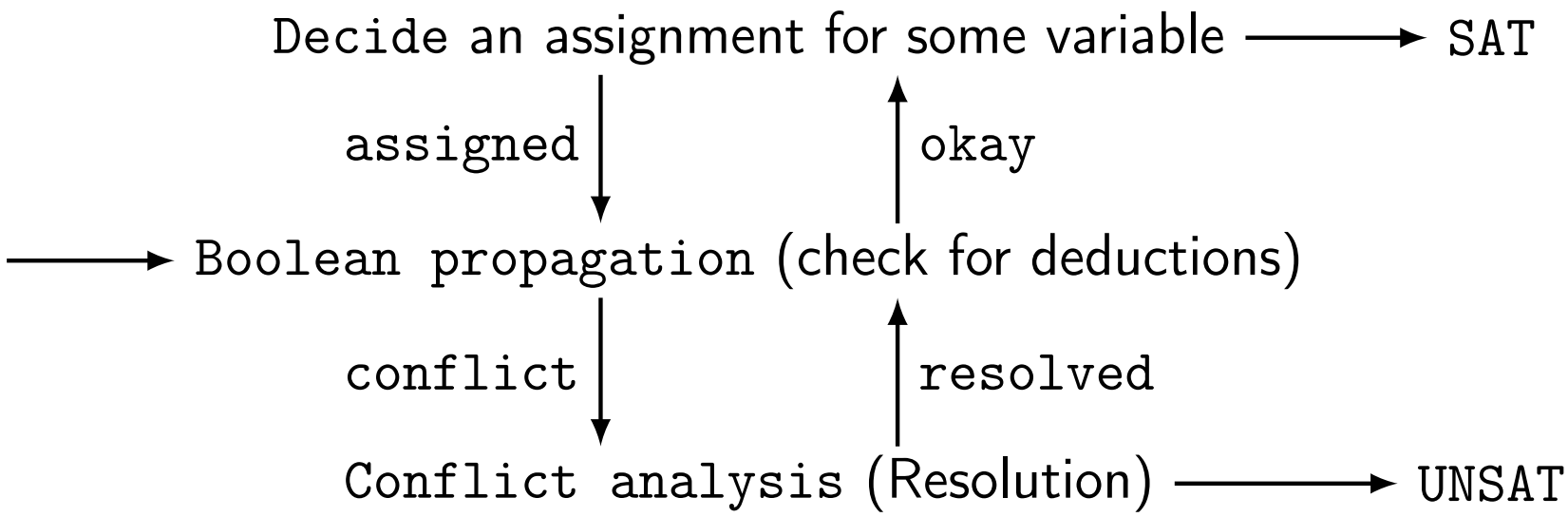
$\varphi \;=\; c_4$

$\;=\; \mathit{false}$

$\alpha \;=\; \{p_1 \mapsto \mathit{true},\; p_2 \mapsto \mathit{false},\; p_3 \mapsto \mathit{true},\; p_4 \mapsto \mathit{true}\}$

▷ `Conflict!` Conflict clause: $c_5 = \texttt{Resolvent}(c_3, c_4) = \neg p_1 \vee p_2 \vee \neg p_3$

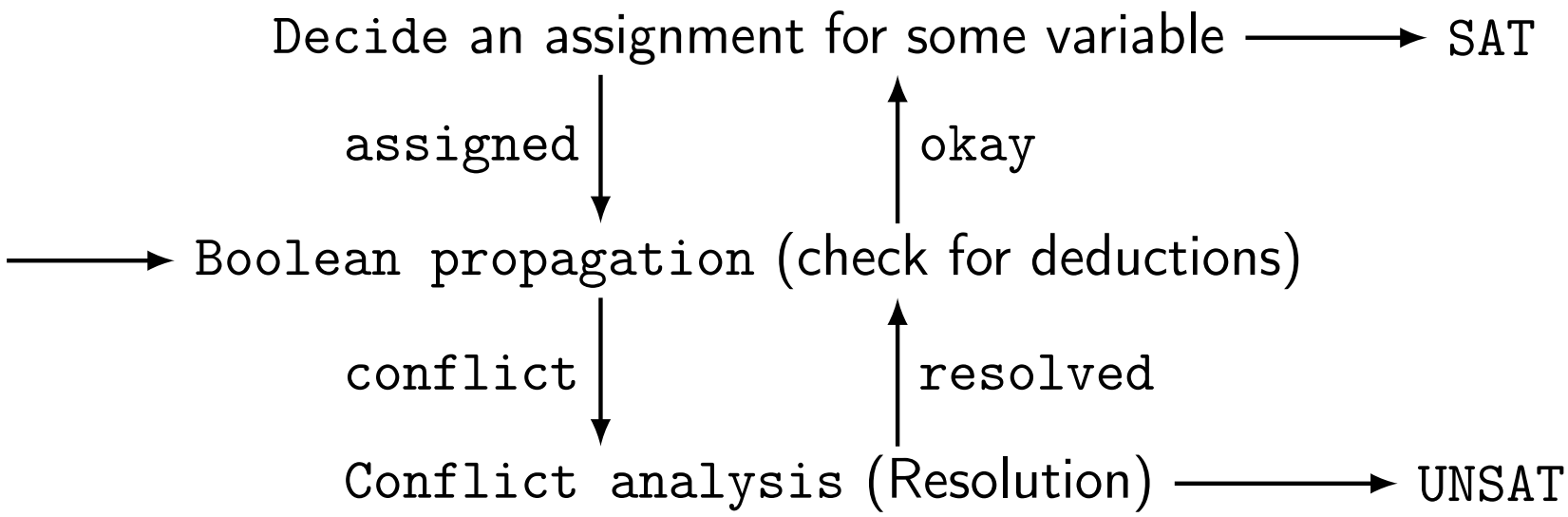
CDCL Algorithm



$$\varphi = c_2 \wedge c_3 \wedge c_4 \wedge c_5$$
$$= (p_2 \vee p_3) \wedge (\quad p_4) \wedge (\quad p_2 \vee \neg p_4)$$
$$\wedge (\neg p_1 \vee p_2 \vee \neg p_3)$$
$$\alpha = \{p_1 \mapsto \mathit{true}\}$$

- ▷ Resolution

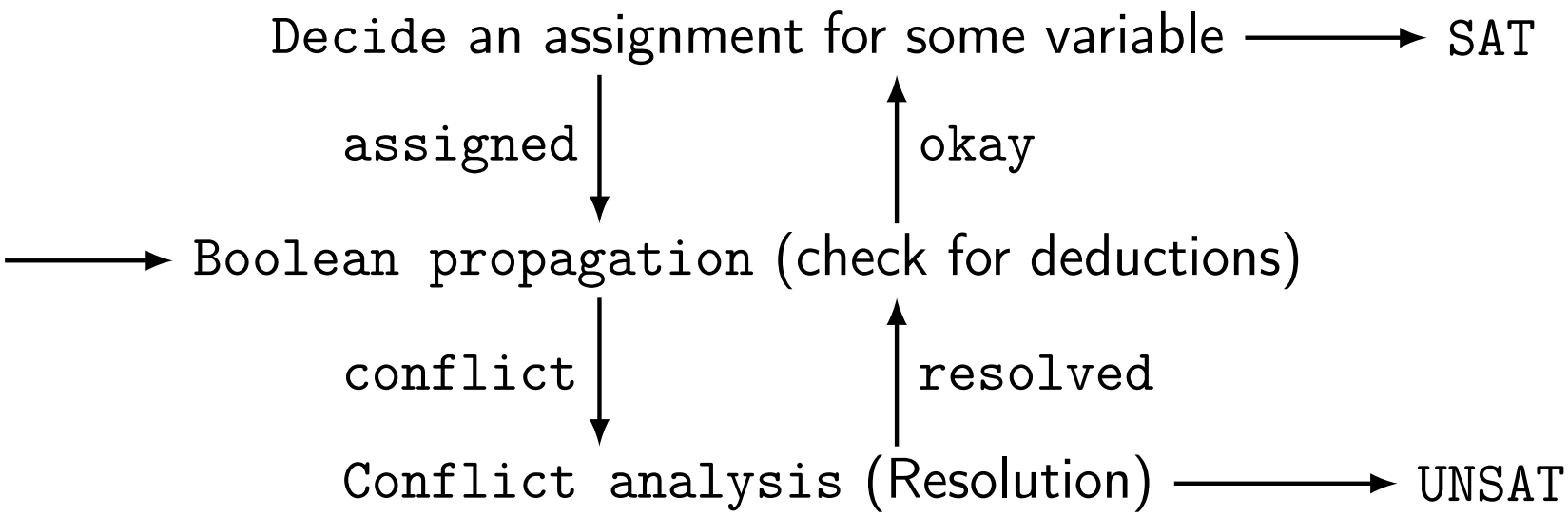
CDCL Algorithm



$$\varphi = c_2 \wedge c_3 \wedge c_4 \wedge c_5$$
$$= (p_2 \vee p_3) \wedge (\quad p_4) \wedge (\quad p_2 \vee \neg p_4)$$
$$\wedge (\neg p_1 \vee p_2 \vee \neg p_3)$$
$$\alpha = \{p_1 \mapsto \mathit{true}\}$$

- ...

CDCL Algorithm



$\varphi = \mathit{true}$

$\alpha = \{p_1 \mapsto \mathit{true},\ p_2 \mapsto \mathit{true},\ p_3 \mapsto \mathit{true},\ p_4 \mapsto \mathit{true}\}$

▷ SAT

Equality Logic

Logic formula in CNF

$$\varphi = \underbrace{\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} c_{ij}}_{clause}, \text{ with constraints } c_{ij}.$$

- all constraints are propositions
  $\rightarrow$ **Propositional logic**

Equality Logic

Logic formula in CNF

$$\varphi = \bigwedge_{i=1}^{n} \underbrace{\bigvee_{j=1}^{m_i} c_{ij}}_{clause}, \text{ with constraints } c_{ij}.$$

- all constraints are propositions
  $\rightarrow$ **Propositional logic**
- constraints can also be equalities between constants and variables with values from some domain, e.g. $\varphi = a = b \wedge b = c \wedge a \neq c, a, b, c \in D$
  $\rightarrow$ **Equality logic**

Equality Logic

Logic formula in CNF

$$\varphi = \bigwedge_{i=1}^{n} \underbrace{\bigvee_{j=1}^{m_i} c_{ij}}_{clause}, \text{ with constraints } c_{ij}.$$

- all constraints are propositions
  $\rightarrow$ **Propositional logic**
- constraints can also be equalities between constants and variables with values from some domain, e.g. $\varphi = a = b \wedge b = c \wedge a \neq c, a, b, c \in D$
  $\rightarrow$ **Equality logic**
- additionally to constants and variables, function symbols can be used, e.g. $\varphi = (f(a) = b \wedge b < f(c)) \Rightarrow f(a) = f(c), a, b, c \in D, f$ function symbol
  $\rightarrow$ **Equality logic with uninterpreted functions**

Uninterpreted Functions

- semantic of function symbols is not specified

Uninterpreted Functions

- semantic of function symbols is not specified
- only constraint on semantics:

Functional congruence

$x = y \Rightarrow f(x) = f(y)$ for all terms x and y and function symbols f.
Terms are constants, variables or functions of terms.

Uninterpreted Functions

- semantic of function symbols is not specified
- only constraint on semantics:

Functional congruence

$x = y \Rightarrow f(x) = f(y)$ for all terms x and y and function symbols f.
Terms are constants, variables or functions of terms.

e.g. $a \neq b \wedge f(a) = f(b)$ is a satisfiable formula but
$a = b \wedge f(a) \neq f(b)$ is unsatisfiable

Satisfiability Modulo Theories

Satisfiability modulo theories formula in CNF

$$\varphi = \bigwedge_{i=1}^{n} \underbrace{\bigvee_{j=1}^{m_i} c_{ij}}_{clause}, \text{ with constraints } c_{ij} \text{ over some theory.}$$

Satisfiability Modulo Theories

Satisfiability modulo theories formula in CNF

$$\varphi = \bigwedge_{i=1}^{n} \underbrace{\bigvee_{j=1}^{m_i} c_{ij}}_{clause}, \text{ with constraints } c_{ij} \text{ over some theory.}$$

- φ is satisfiable iff an assignment for the contained variables in their respective domains exists such that φ evaluates to *true*

Satisfiability Modulo Theories

Satisfiability modulo theories formula in CNF

$$\varphi = \bigwedge_{i=1}^{n} \underbrace{\bigvee_{j=1}^{m_i} c_{ij}}_{clause}, \text{ with constraints } c_{ij} \text{ over some theory.}$$

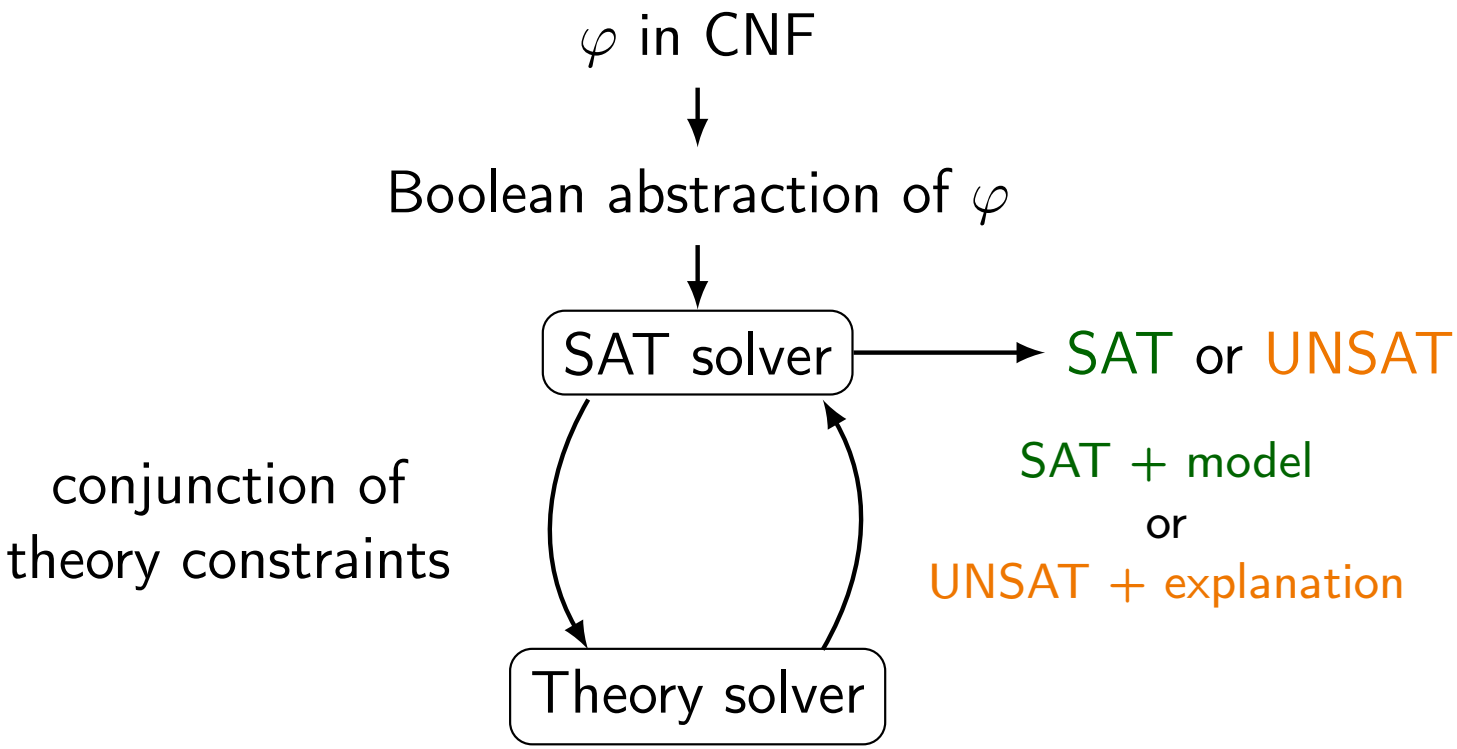
- φ is satisfiable iff an assignment for the contained variables in their respective domains exists such that φ evaluates to *true*
- SAT modulo theories problem (SMT): is a given φ satisfiable?

SMT Solving

Eager: encode SMT problems into SAT problems and use a SAT solver

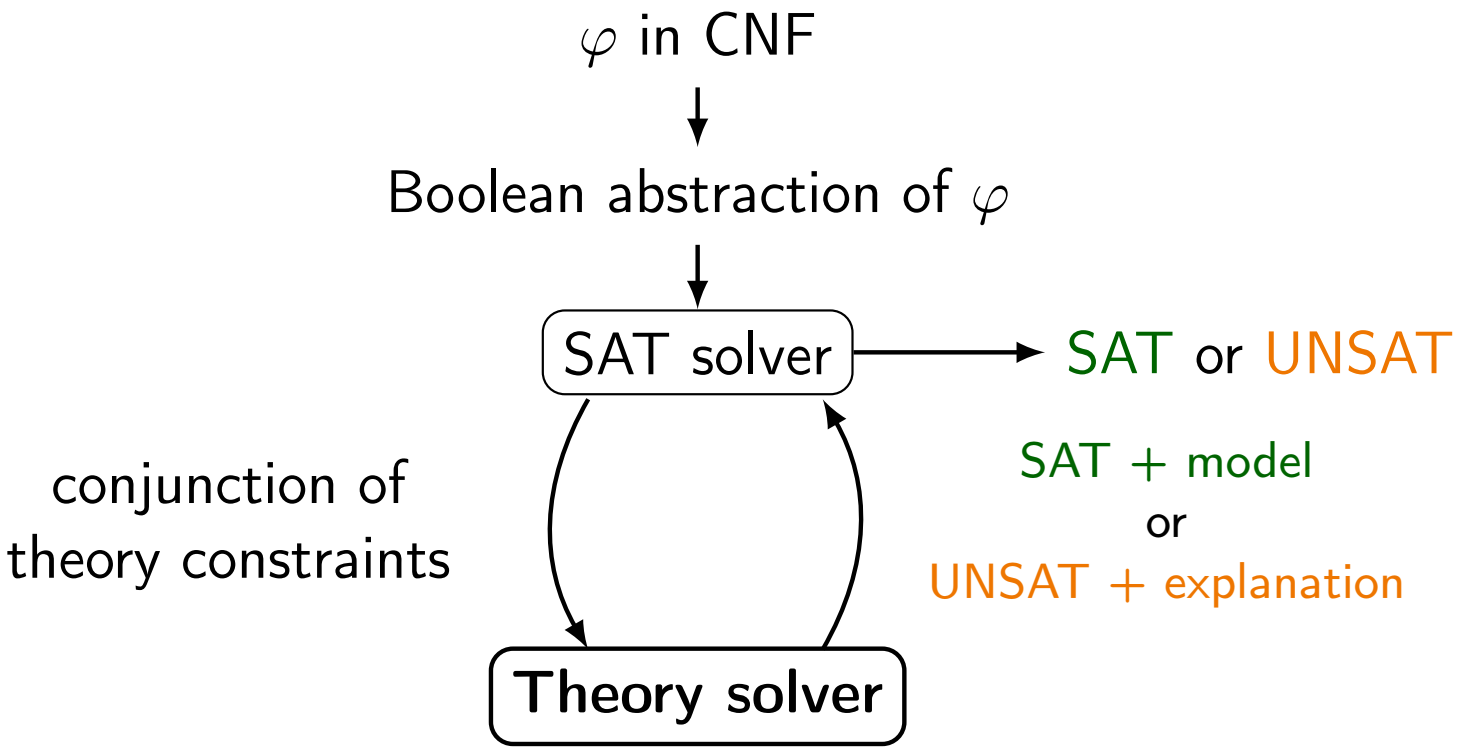
SMT Solving

Eager: encode SMT problems into SAT problems and use a SAT solver
Lazy: combine decision procedures to solve SMT problems.



SMT Solving

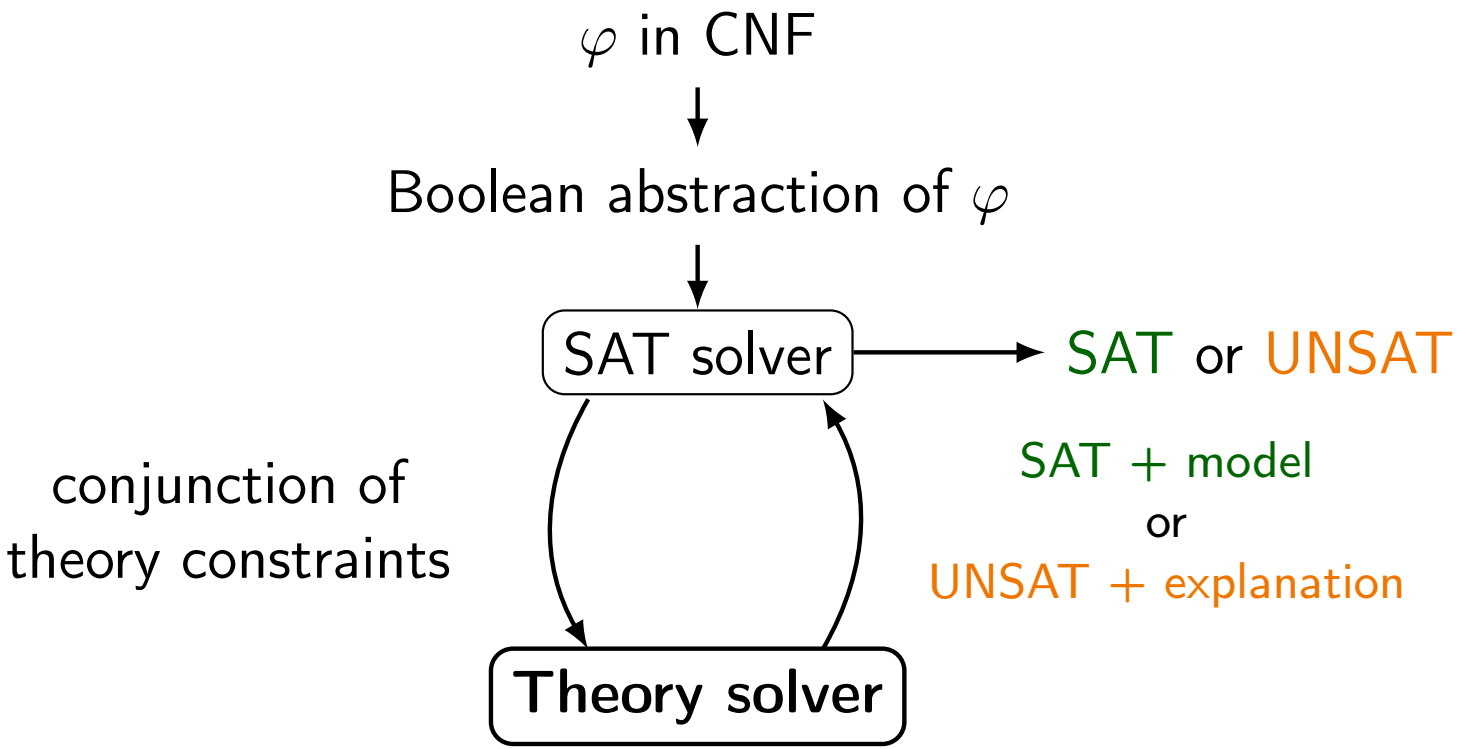
Eager: encode SMT problems into SAT problems and use a SAT solver
Lazy: combine decision procedures to solve SMT problems.



- full lazy: theory calls only for full Boolean solutions

SMT Solving

Eager: encode SMT problems into SAT problems and use a SAT solver
Lazy: combine decision procedures to solve SMT problems.



- full lazy: theory calls only for full Boolean solutions
- **less** lazy: theory call after every conflict-free `Propagate` execution

SMT-RAT

- is written in C++, a tutorial can be found for example at

  `http://www.cplusplus.com/doc/tutorial/`

SMT-RAT

- is written in C++, a tutorial can be found for example at

  `http://www.cplusplus.com/doc/tutorial/`

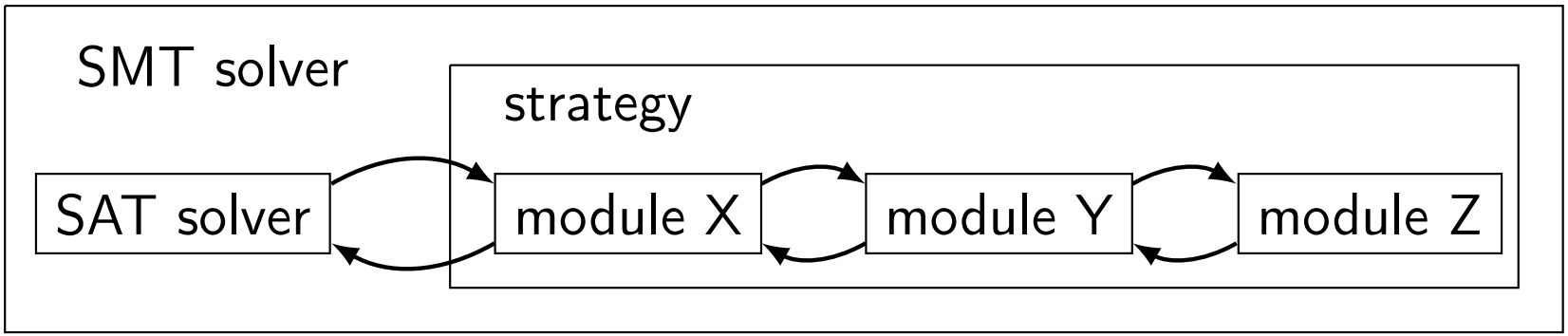
- consists of **modules**

SMT-RAT

- is written in C++, a tutorial can be found for example at

  `http://www.cplusplus.com/doc/tutorial/`

- consists of **modules**
- modules can be composed to an SMT solver according to a **strategy**



Module

- can be informed about any constraint it could receive eventually: `informCore`

Module

- can be informed about any constraint it could receive eventually: `informCore`
- gets sets of formulas: `addCore` (and `removeCore`)

Module

- can be informed about any constraint it could receive eventually: `informCore`
- gets sets of formulas: `addCore` (and `removeCore`)
- checks these sets for satisfiability: `checkCore`

Module

- can be informed about any constraint it could receive eventually: `informCore`
- gets sets of formulas: `addCore` (and `removeCore`)
- checks these sets for satisfiability: `checkCore`
- can give sets of formulas to other modules according to the strategy: `addSubformulaToPassedFormula`, `removeSubformula` and `runBackends`

Theory Solver

- for *conjunctions / sets* of equations and disequations over terms

Theory Solver

- for *conjunctions / sets* of equations and disequations over terms

Theory solver input

$\Phi = \{c \mid \alpha(X_c) = \mathit{true}\} \cup \{\neg c \mid \alpha(X_c) = \mathit{false}\}$,
for a partial assignment α

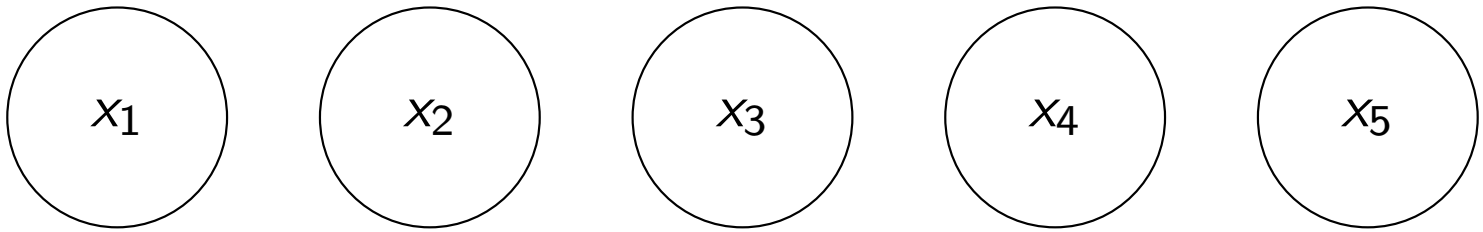
Theory Solver

- for *conjunctions / sets* of equations and disequations over terms

Theory solver input

$\Phi = \{c \mid \alpha(X_c) = \mathit{true}\} \cup \{\neg c \mid \alpha(X_c) = \mathit{false}\}$,
for a partial assignment α

Equality logic: $\varphi^E = x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge x_5 \neq x_1$
We assume an infinite domain. Assure reflexivity



all subterms

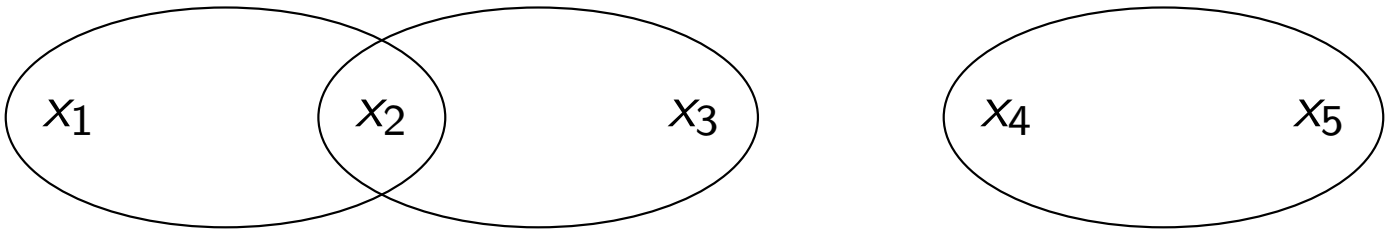
Theory Solver

- for *conjunctions / sets* of equations and disequations over terms

Theory solver input

$\Phi = \{c \mid \alpha(X_c) = \mathit{true}\} \cup \{\neg c \mid \alpha(X_c) = \mathit{false}\}$,
for a partial assignment α

Equality logic: $\varphi^E = x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge x_5 \neq x_1$
We assume an infinite domain. Assure reflexivity, symmetry



consider equalities

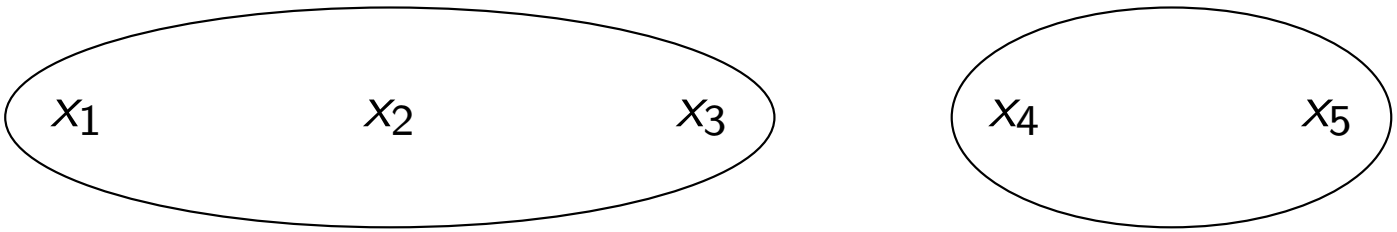
Theory Solver

- for *conjunctions / sets* of equations and disequations over terms

Theory solver input

$\Phi = \{c \mid \alpha(X_c) = \mathit{true}\} \cup \{\neg c \mid \alpha(X_c) = \mathit{false}\}$,
for a partial assignment α

Equality logic: $\varphi^E = x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge x_5 \neq x_1$
We assume an infinite domain. Assure reflexivity, symmetry and transitivity.



partition of the subterms

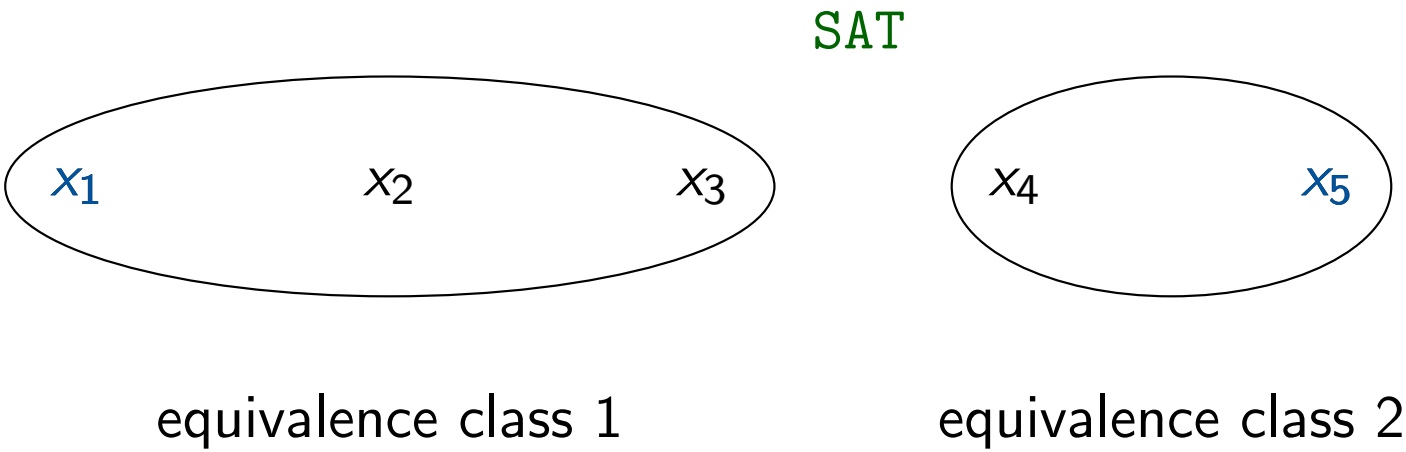
Theory Solver

- for *conjunctions / sets* of equations and disequations over terms

Theory solver input

$\Phi = \{c \mid \alpha(X_c) = \mathit{true}\} \cup \{\neg c \mid \alpha(X_c) = \mathit{false}\}$,
for a partial assignment α

Equality logic: $\varphi^E = x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge x_5 \neq x_1$
We assume an infinite domain. Assure reflexivity, symmetry and transitivity.



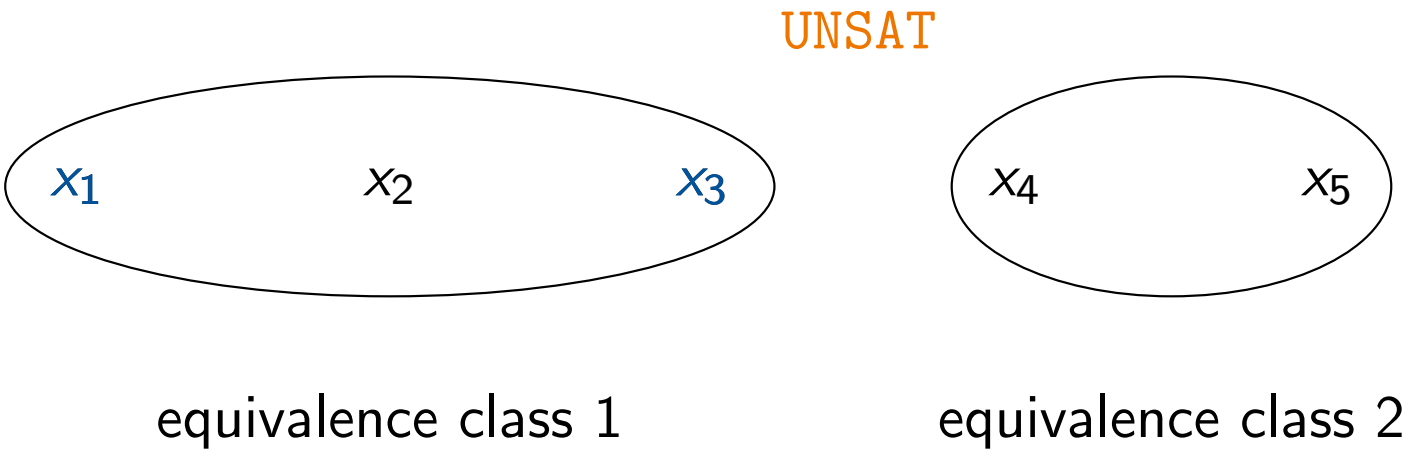
Theory Solver

- for *conjunctions / sets* of equations and disequations over terms

Theory solver input

$\Phi = \{c \mid \alpha(X_c) = \mathit{true}\} \cup \{\neg c \mid \alpha(X_c) = \mathit{false}\}$,
for a partial assignment α

Equality logic: $\varphi^E = x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge x_1 \neq x_3$
We assume an infinite domain. Assure reflexivity, symmetry and transitivity.



Theory Solver

- for *conjunctions / sets* of equations and disequations over terms

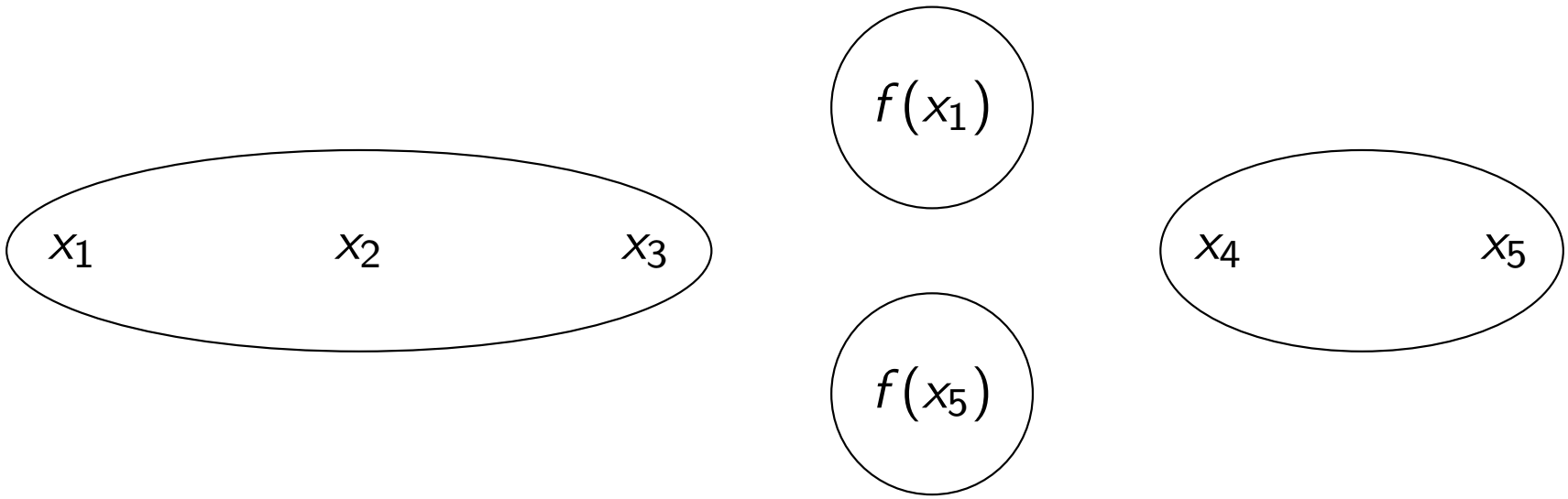
Theory solver input

$\Phi = \{c \mid \alpha(X_c) = \mathit{true}\} \cup \{\neg c \mid \alpha(X_c) = \mathit{false}\}$,
for a partial assignment α

EL + uninterpreted functions:

$x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge f(x_5) \neq f(x_1)$

We assume an infinite domain.



Theory Solver

- for *conjunctions / sets* of equations and disequations over terms

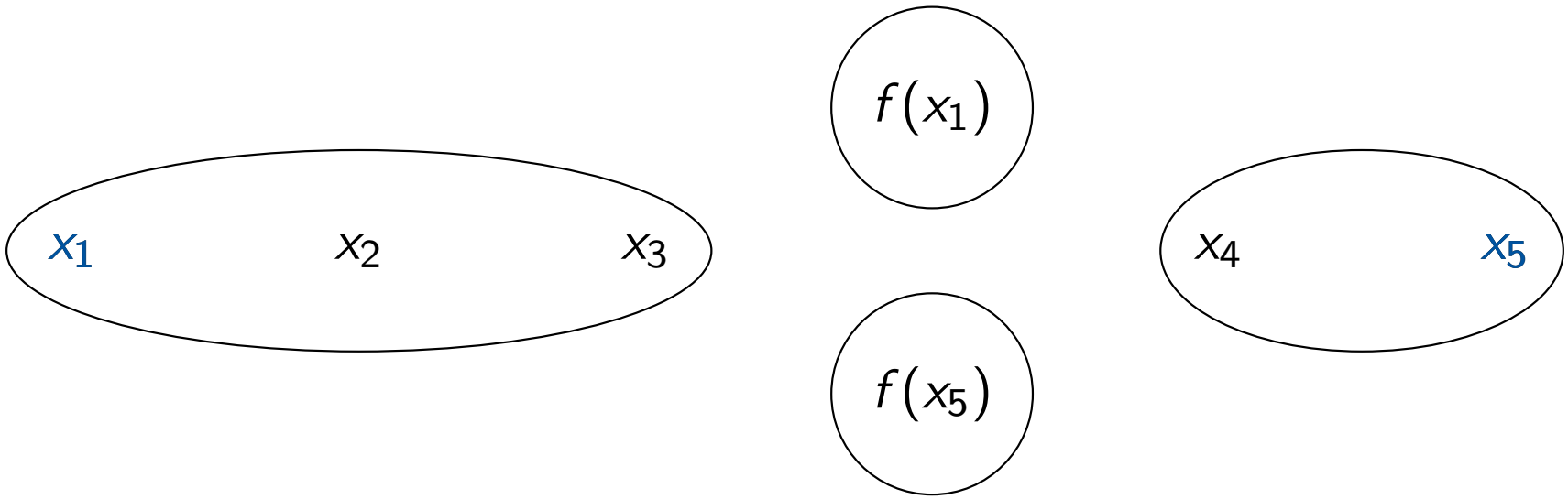
Theory solver input

$\Phi = \{c \mid \alpha(X_c) = \mathit{true}\} \cup \{\neg c \mid \alpha(X_c) = \mathit{false}\}$,
for a partial assignment α

EL + uninterpreted functions:

$x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge f(x_5) \neq f(x_1)$

We assume an infinite domain. Assure functional congruence.



Theory Solver

- for *conjunctions / sets* of equations and disequations over terms

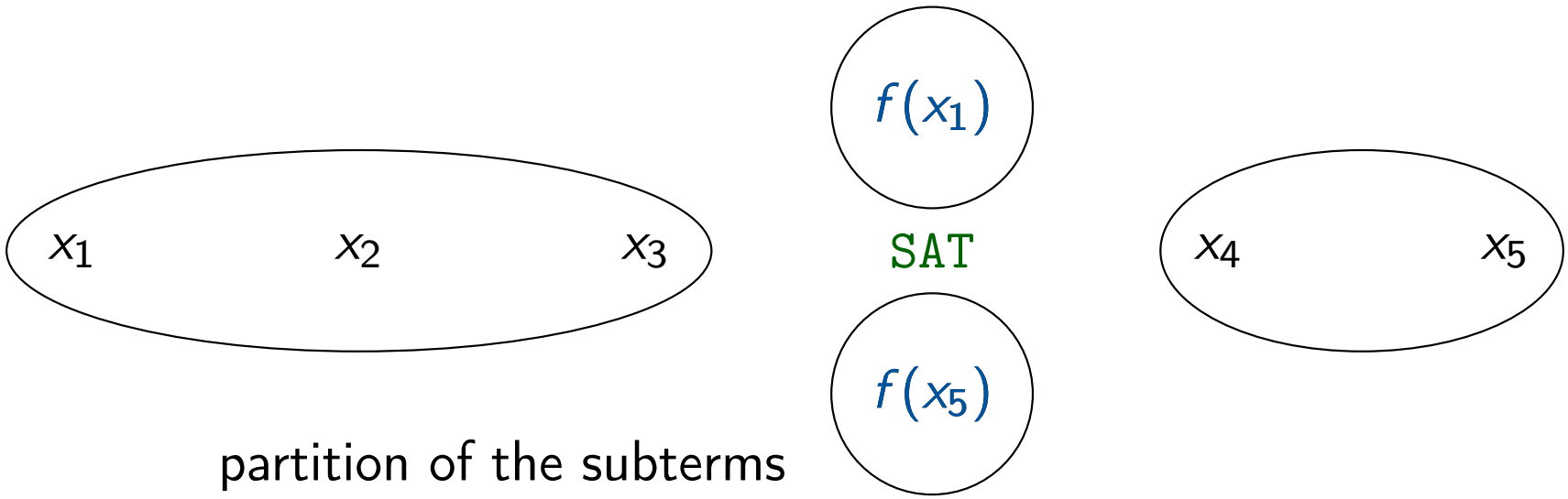
Theory solver input

$\Phi = \{c \mid \alpha(X_c) = \mathit{true}\} \cup \{\neg c \mid \alpha(X_c) = \mathit{false}\}$,
for a partial assignment α

EL + uninterpreted functions:

$x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge f(x_5) \neq f(x_1)$

We assume an infinite domain. Assure functional congruence.



partition of the subterms

Theory Solver

- for *conjunctions / sets* of equations and disequations over terms

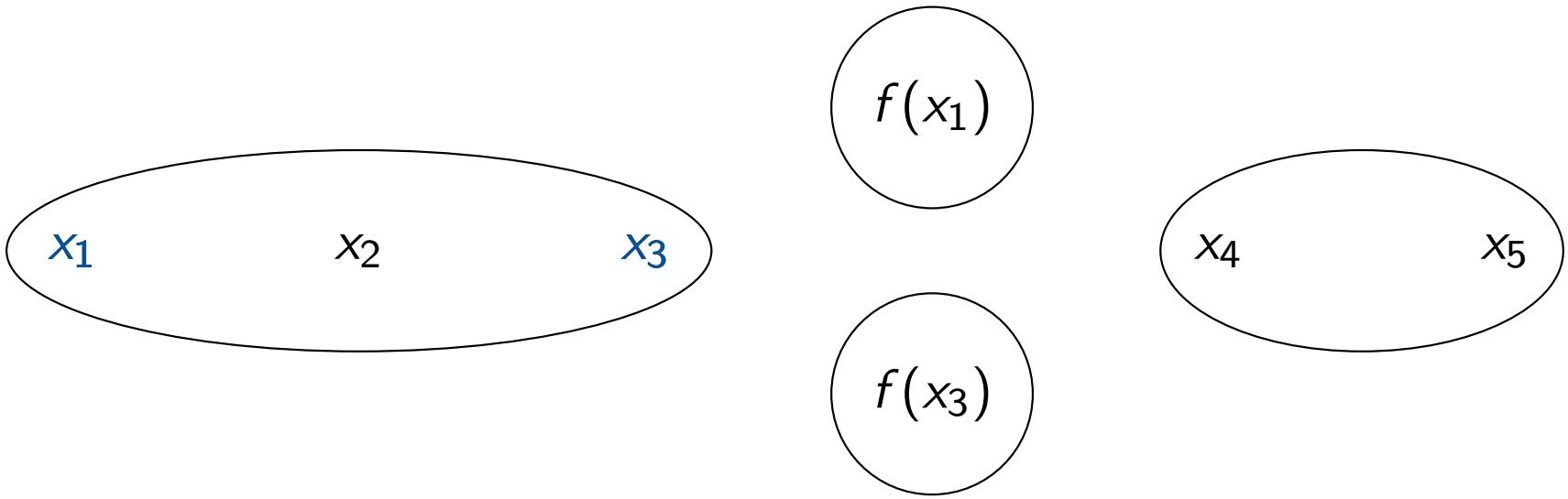
Theory solver input

$\Phi = \{c \mid \alpha(X_c) = \mathit{true}\} \cup \{\neg c \mid \alpha(X_c) = \mathit{false}\}$,
for a partial assignment α

EL + uninterpreted functions:

$x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge f(x_1) \neq f(x_3)$

We assume an infinite domain. Assure functional congruence.



Theory Solver

- for *conjunctions / sets* of equations and disequations over terms

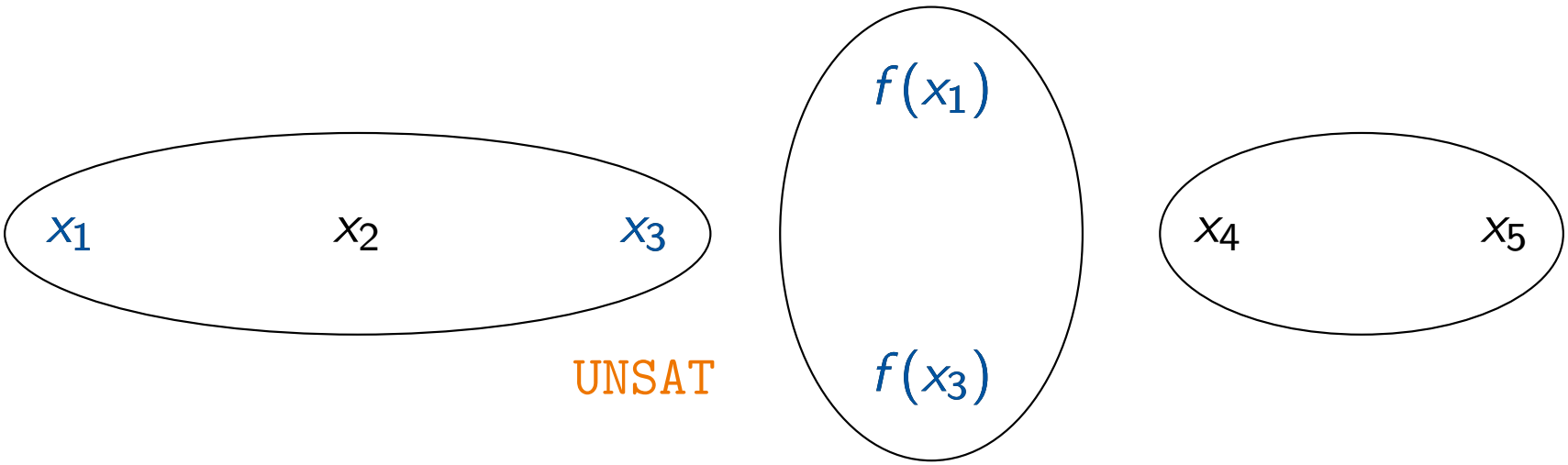
Theory solver input

$\Phi = \{c \mid \alpha(X_c) = \mathit{true}\} \cup \{\neg c \mid \alpha(X_c) = \mathit{false}\}$,
for a partial assignment α

EL + uninterpreted functions:

$x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge f(x_1) \neq f(x_3)$

We assume an infinite domain. Assure functional congruence.



Initialization

Algorithm: `initialize`

Input: Set T of all subterms that can be used in (dis)equations
Output: -

1. no constraints received yet:
   $E := \emptyset$;
2. initial partition over T:
   $\mathcal{C} := \{\{t\} \mid t \in T\}$;
3. current state of satisfiability:
   $state := SAT$;

Equations

Algorithm: `addEquation`

Input: Equation $t_1 = t_2$ with subterms from T
Output: Satisfiability of the conjunction of all received (and not yet removed) equations and disequations

1. remember equation: $E := E \cup \{t_1 = t_2\}$;
2. `if` $(state = UNSAT)$ `then return UNSAT;`
3. update partition and re-check satisfiability of disequations:

```
if ([t1] ≠ [t2]) then
  C := (C \ {[t1], [t2]}) ∪ {[t1] ∪ [t2]};
  while ∃ F(t), F(t') ∈ T : [t]=[t'] ∧ [F(t)]≠[F(t')]
    C := (C \ {[F(t)], [F(t')]}) ∪ {[F(t)] ∪ [F(t')]};
  for each (t ≠ t') ∈ E
    if ([t] = [t']) then {state := UNSAT; return UNSAT;}
return SAT;
```

Disequations

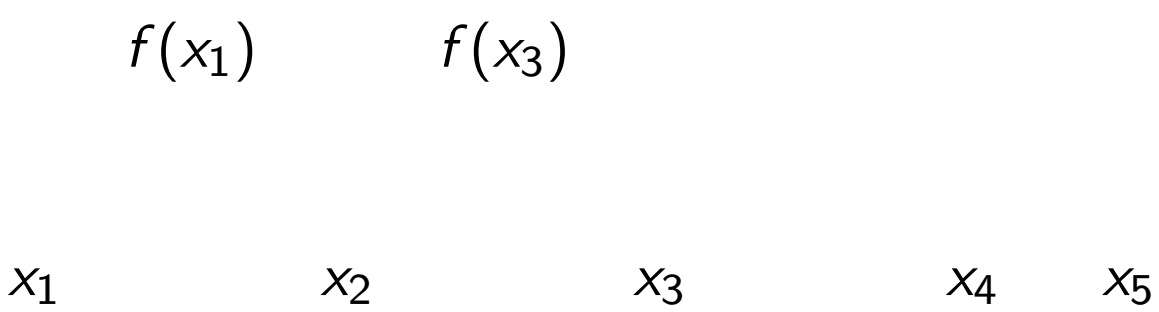
Algorithm: `addDisequation`

Input: Disequation $t_1 \neq t_2$ with subterms from T
Output: Satisfiability of the conjunction of all received (and not yet removed) equations and disequations

1. remember disequation:
   $E := E \cup \{t_1 \neq t_2\}$;
2. `if` $(e = \mathit{UNSAT})$ `then return UNSAT;`
3. check satisfiability of disequation:
   `if` $([t_1] = [t_2])$ `then`
   $\mathit{state} := \mathit{UNSAT}$;
   `return UNSAT;`
   `return SAT;`

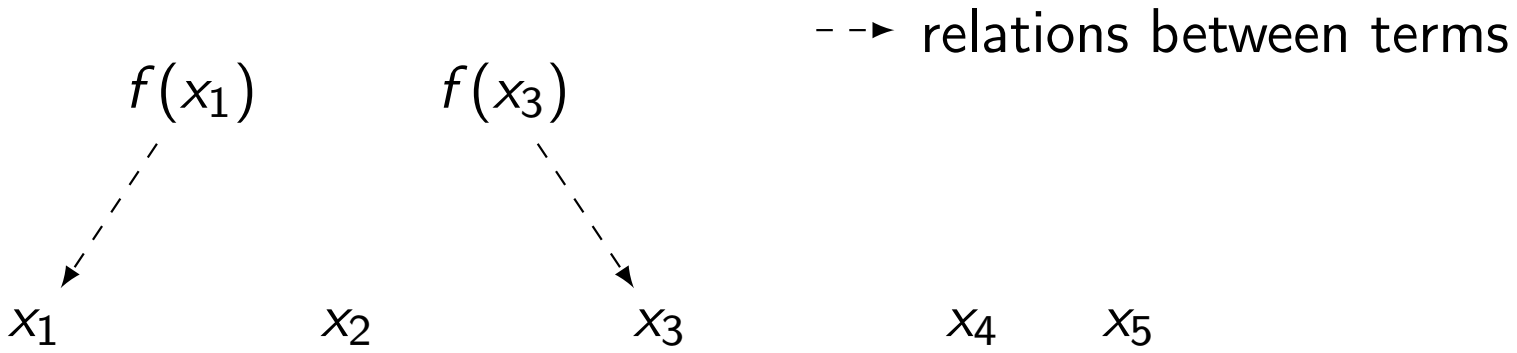
E-Graph

- $x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge f(x_1) \neq f(x_3)$



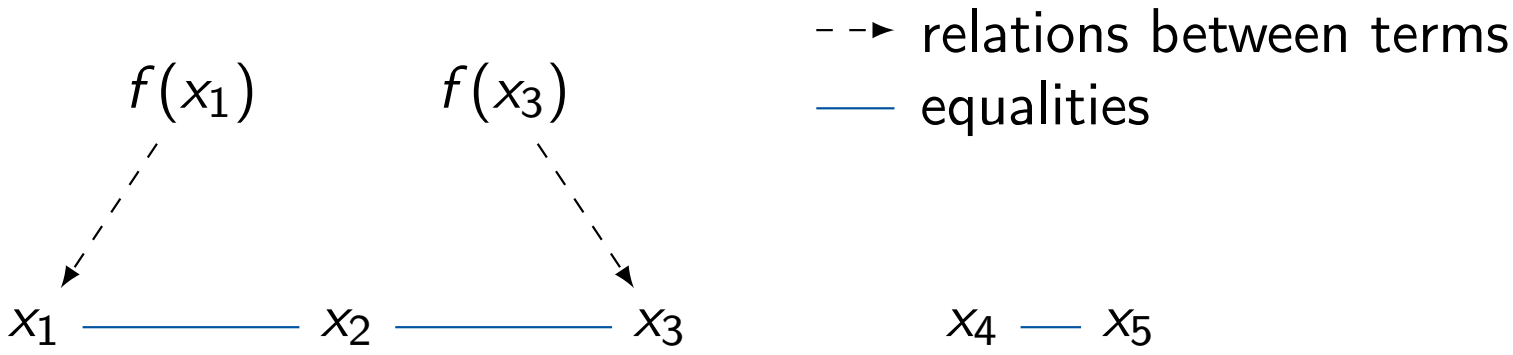
E-Graph

- $x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge f(x_1) \neq f(x_3)$



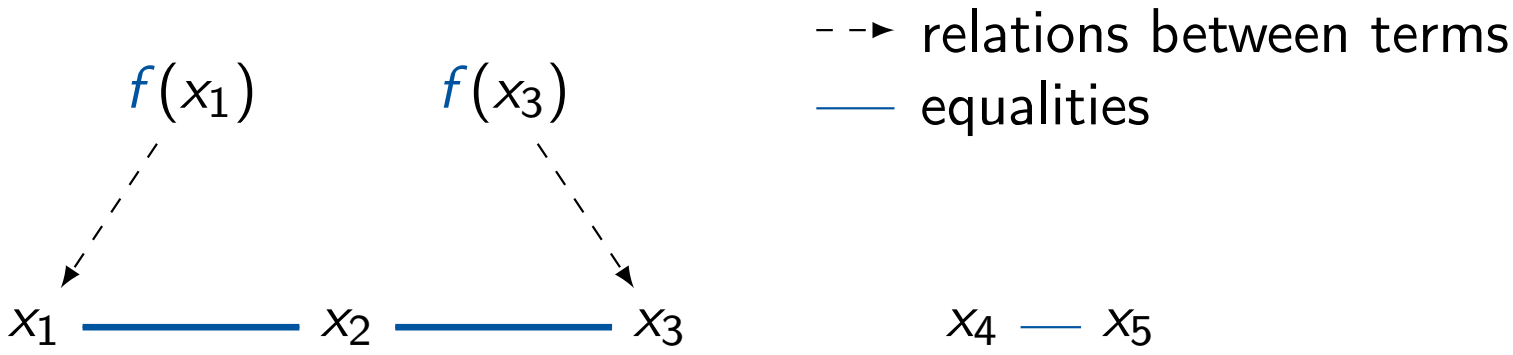
E-Graph

- $x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge f(x_1) \neq f(x_3)$



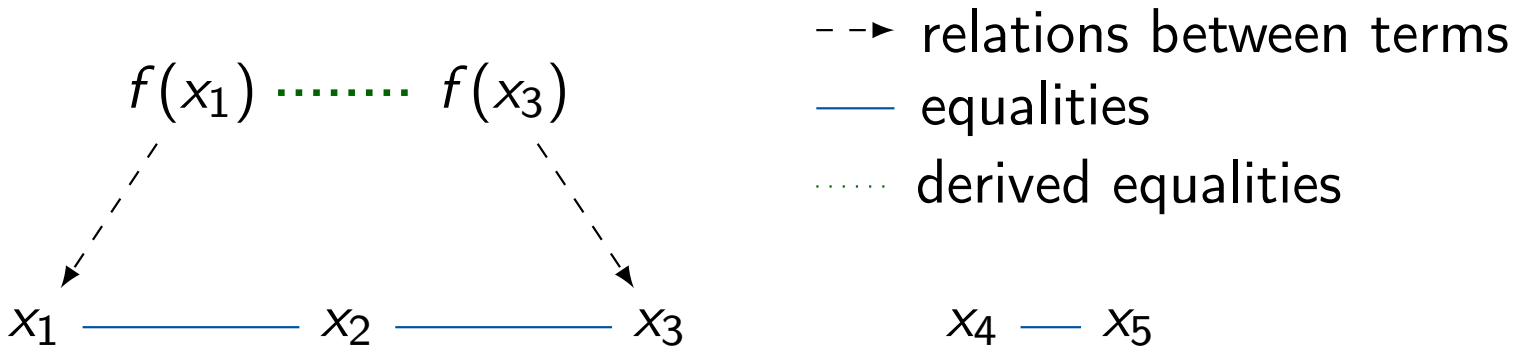
E-Graph

- $x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge f(x_1) \neq f(x_3)$



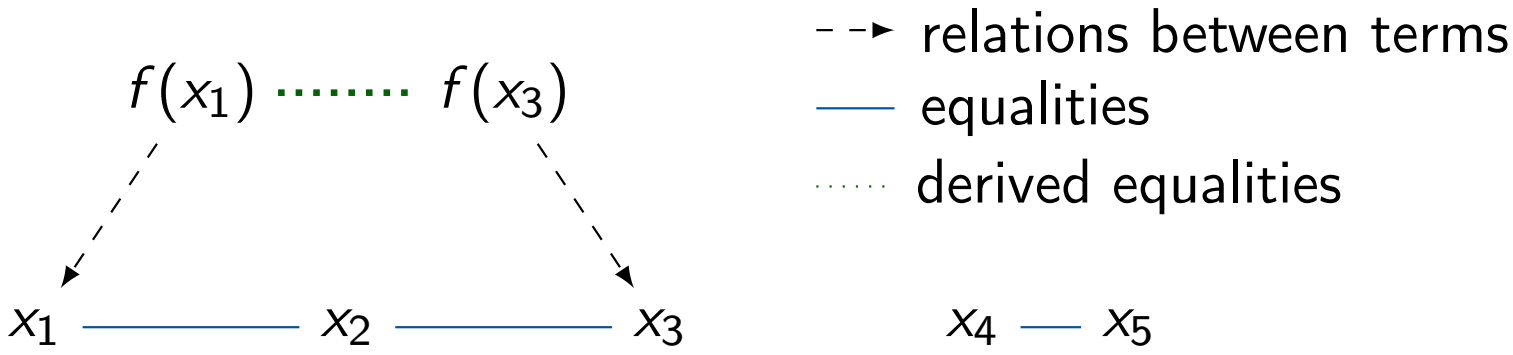
E-Graph

- $x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge f(x_1) \neq f(x_3)$

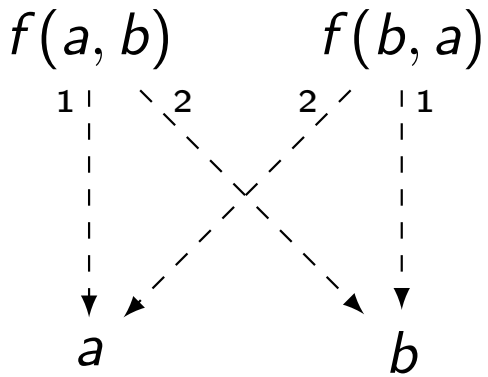


E-Graph

- $x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge f(x_1) \neq f(x_3)$



- Consider that $f(a, b) \neq f(b, a)$ is satisfiable!



Boost Graph Library

- C++ library for graphs, see

  `https://www.boost.org/doc/libs/1_68_0/libs/graph/doc/index.html`

Boost Graph Library

- C++ library for graphs, see
  `https://www.boost.org/doc/libs/1_68_0/libs/graph/doc/index.html`
- you may use other libraries or write your own graph data structure

Boost Graph Library

- C++ library for graphs, see
  `https://www.boost.org/doc/libs/1_68_0/libs/graph/doc/index.html`
- you may use other libraries or write your own graph data structure
- recommendation for sparse graphs: `adjacency_list`;
  for dense graphs: `adjacency_matrix`

Boost Graph Library

- C++ library for graphs, see
  `https://www.boost.org/doc/libs/1_68_0/libs/graph/doc/index.html`
- you may use other libraries or write your own graph data structure
- recommendation for sparse graphs: `adjacency_list`;
  for dense graphs: `adjacency_matrix`
- examples for both can be found on the respective pages in the above linked documentation

Models

Models (satisfying assignments)

- can be inferred from the E-graph

Models

Models (satisfying assignments)

- can be inferred from the E-graph
- are called `mModel` in SMT-RAT and can be updated with `updateModel`

Models

Models (satisfying assignments)

- can be inferred from the E-graph
- are called `mModel` in SMT-RAT and can be updated with `updateModel`
- must be complete, that is all variables and uninterpreted functions occurring in the received formula must be assigned to a value of their corresponding domain

  (domains are in general assumed to be infinite)

Models

Models (satisfying assignments)

- can be inferred from the E-graph
- are called `mModel` in SMT-RAT and can be updated with `updateModel`
- must be complete, that is all variables and uninterpreted functions occurring in the received formula must be assigned to a value of their corresponding domain

  (domains are in general assumed to be infinite)

Models enumerate equivalence classes and assign each variable and uninterpreted function the number of their corresponding equivalence class.

Domains

Different domains may occur.

Domains

Different domains may occur.

- Undefined domains (`Sorts`) are assumed to be infinite, `Int` and `Real` are infinite domains.

Domains

Different domains may occur.

- Undefined domains (`Sorts`) are assumed to be infinite, `Int` and `Real` are infinite domains.
- `Bool` is a finite domain!
  E.g. $a \neq b \wedge b \neq c \wedge c \neq a$ is unsatisfiable for $a,\ b,\ c \in$ `Bool`, although there is no explicit equality.

Infeasible Subset

- set of constraints that are unsatisfiable

Infeasible Subset

- set of constraints that are unsatisfiable
- is called `mInfeasibleSubset` in SMT-RAT

Infeasible Subset

- set of constraints that are unsatisfiable
- is called `mInfeasibleSubset` in SMT-RAT
- the trivial infeasible subset can be obtained with `generateTrivialInfeasibleSubset`

Infeasible Subset

- set of constraints that are unsatisfiable
- is called `mInfeasibleSubset` in SMT-RAT
- the trivial infeasible subset can be obtained with `generateTrivialInfeasibleSubset`

- later more about *minimal* infeasible subsets

Backtracking

This is just a rather naive (and informal) solution for backtracking. More optimal solutions need smart data structures for efficient book-keeping.

Backtracking

This is just a rather naive (and informal) solution for backtracking. More optimal solutions need smart data structures for efficient book-keeping.

Algorithm: `backtrack`

Input: Equation and disequation set S
Output: -

1. remove the equations and disequations: $E := E \setminus S$;
2. compute satisfiability for T and the conjunction of all equations and disequations from E
3. remember satisfiability result in `state`
4. return satisfiability result

References

Florian Corzilius, Gereon Kremer, Sebastian Junges, Stefan Schupp, and Erika Ábrahám.

SMT-RAT: An Open Source C++ Toolbox for Strategic and Parallel SMT Solving.

In *Theory and Applications of Satisfiability Testing – SAT 2015*, pages 360–368. Springer International Publishing, 2015.

David Detlefs, Greg Nelson, and James B. Saxe.

Simplify: A theorem prover for program checking.

J. ACM, 52(3):365–473, May 2005.

Daniel Kroening and Ofer Strichman.

Decision Procedures: An Algorithmic Point of View, pages 59–110.

Springer, 2008.

Greg Nelson and Derek C. Oppen.

Fast decision procedures based on congruence closure.

J. ACM, 27(2):356–364, April 1980.